

SCRUM

ГИБКАЯ РАЗРАБОТКА ПО

*ОПИСАНИЕ ПРОЦЕССА УСПЕШНОЙ ГИБКОЙ
РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
С ИСПОЛЬЗОВАНИЕМ SCRUM*

SUCCEEDING WITH AGILE

SOFTWARE DEVELOPMENT USING SCRUM

Mike Cohn



ADDISON-WESLEY

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokio • Singapore • Mexico City

SCRUM ГИБКАЯ РАЗРАБОТКА ПО

*ОПИСАНИЕ ПРОЦЕССА УСПЕШНОЙ ГИБКОЙ
РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
С ИСПОЛЬЗОВАНИЕМ SCRUM*

Майк Кон



Москва • Санкт-Петербург • Киев
2011

ББК 32.973.26-018.2.75

К64

УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией С.Н. Тригуб

Перевод с английского и редакция канд. техн. наук И.В. Красикова

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:

info@williamspublishing.com, http://www.williamspublishing.com

Кон, Майк.

K64 Scrum: гибкая разработка ПО. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2011. — 576 с.
: ил. — Парал. тит. англ.

ISBN 978-5-8459-1731-7 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Authorized translation from the English language edition published by Addison-Wesley Publishing Company, Inc,
Copyright © 2010 Mike Cohn

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2011

Научно-популярное издание

Майк Кон

SCRUM

гибкая разработка ПО

Литературный редактор *Л.Н. Красножон*

Верстка *Л.В. Чернокозинская*

Художественный редактор *В.Г. Павлютин*

Корректор *Л.А. Гордиенко*

Подписано в печать 04.04.2011. Формат 70x100/16.

Гарнитура Times. Печать офсетная.

Усл. печ. л. 46,44. Уч.-изд. л. 38,70.

Тираж 1000 экз. Заказ № 0000.

Отпечатано по технологии CtP

в ОАО “Печатный двор” им. А. М. Горького

197110, Санкт-Петербург, Чкаловский пр., 15.

ООО “И. Д. Вильямс”, 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1731-7 (рус.)

ISBN 978-0-321-57936-2 (англ.)

© Издательский дом “Вильямс”, 2011

© Mike Cohn, 2010

Оглавление

Предисловие	21
Благодарности	23
Об авторе	27
Введение	29
Часть I. Приступаем к освоению Scrum	35
Глава 1. Гибкая методология разработки: нелегко, но перспективно	37
Глава 2. ADAPТация к Scrum	57
Глава 3. Модели внедрения Scrum	83
Глава 4. Движение в направлении гибкости	103
Глава 5. Ваши первые проекты	127
Часть II. Люди	143
Глава 6. Преодоление сопротивления	145
Глава 7. Новые роли	167
Глава 8. Изменившиеся роли	189
Глава 9. Технические приемы	211
Часть III. Команды	235
Глава 10. Структура команды	237
Глава 11. Организация коллективного труда	265
Глава 12. Управление самоорганизующимся коллективом	287
Глава 13. Журнал запросов на выполнение работ	305
Глава 14. Спринты	329
Глава 15. Планирование	363
Глава 16. Качество	389

Часть IV. Организация	411
Глава 17. Изменение масштаба Scrum	413
Глава 18. Рассредоточенные коллективы	445
Глава 19. Сосуществование с другими подходами	485
Глава 20. Кадры, техническое обеспечение и отдел управления проектами	503
Часть V. Что дальше	529
Глава 21. Как далеко вы продвинулись в деле внедрения Scrum	531
Глава 22. Впереди еще много работы	551
Список литературы	553
Предметный указатель	565

Содержание

Предисловие	21
Благодарности	23
Об авторе	27
Введение	29
Часть I. Приступаем к освоению Scrum	35
Глава 1. Гибкая методология разработки: нелегко, но перспективно	37
Чем обусловлены трудности перехода к использованию Scrum	39
Успешное изменение не осуществляется исключительно по принципу “сверху вниз” или “снизу вверх”	40
Конечное состояние перехода к Scrum непредсказуемо	41
Scrum проникает буквально во все поры организации	42
Scrum — очень своеобразная методология	43
Перемены происходят быстрее, чем прежде	44
Передовые методы таят в себе опасность	45
Почему переход к использованию Scrum стоит затрачиваемых на него усилий	46
Повышение производительности и снижение издержек	47
Повышение заинтересованности работников в выполняемой ими работе	48
и более высокая удовлетворенность ею	50
Сокращение времени вывода новых продуктов на рынок	51
Повышение качества	52
Повышение удовлетворенности всех, кто так или иначе связан с разработкой новых продуктов	53
Неудовлетворенность прежними методами работы	54
Что дальше	55
Дополнительная литература	56
Глава 2. ADAPТация к Scrum	57
Осознание	59
Инструменты, способствующие осознанию	61
Желание	63
Инструменты, способствующие возникновению желания перейти к Scrum	64
Способность	68
Инструменты, позволяющие выработать способность	69
Продвижение	72
Инструменты продвижения Scrum	73
Распространение	76
Источники “организационной гравитации”	77
Взглянем на картину в целом	80
Дополнительная литература	81

Глава 3. Модели внедрения Scrum	83
Массовость: за и против	83
Почему следует отдать предпочтение постепенному переходу	85
Почему следует отдать предпочтение одновременному переходу	86
Выбор между одновременным и постепенным переходами	87
Открытый и скрытый варианты перехода	88
Причины, по которым предпочтение отдается открытому варианту перехода	89
Причины, по которым предпочтение отдается скрытому варианту перехода	90
Выбор между скрытым и открытым вариантами перехода к использованию Scrum	91
Варианты распространения опыта использования Scrum	92
Метод “разделить и засеять”	92
Метод “нарастить и разделить”	93
Внутреннее наставничество	94
Преимущества метода “разделить и засеять”	94
Преимущества метода “нарастить и разделить”	95
Преимущества внутреннего наставничества	95
Выберите свой подход	96
Внедрение новых технических приемов	97
Причины раннего опробования новых приемов	98
Причины позднего опробования новых приемов	99
Последнее соображение	99
Дополнительная литература	101
Глава 4. Движение в направлении гибкости	103
Журнал совершенствования	105
Сообщество в поддержку перехода к Scrum	107
Спринты ЕТС	108
Обязанности ЕТС	111
Сообщества в поддержку усовершенствований	114
Катализаторы усовершенствований	116
Спринт сообщества в поддержку усовершенствований	117
Сосредоточьтесь на целях, имеющих непосредственное отношение к практике	119
Члены сообщества в поддержку усовершенствований	120
Роспуск сообщества	122
Универсального рецепта не существует	123
Что дальше	123
Дополнительная литература	124
Глава 5. Ваши первые проекты	127
Выбор pilotного проекта	128
Четыре характеристики идеального pilotного проекта	128
Выбор времени для внедрения Scrum	131
Когда приближается час расплаты	131
Выбор pilotной команды	133
Если pilotный проект неудачен	134
Формулировка ожиданий и управление ими	135
Ожидания, касающиеся прогресса	136
Ожидания, касающиеся предсказуемости	138

Ожидания, касающиеся отношения к Scrum	139
Ожидания, касающиеся участия всех заинтересованных сторон	139
Речь идет о пилотном проекте	140
Дополнительная литература	140
Часть II. Люди	143
Глава 6. Преодоление сопротивления	145
Можно ли предвидеть возникновение сопротивления	146
Кто будет сопротивляться	146
Заблуждения и фобии	148
Информация о переменах	150
Послания лидеров	150
Мнение коллег	151
Тонкости индивидуального сопротивления	152
Сkeptики	155
Саботажники	158
Консерваторы	160
Последователи	162
Сопротивление как полезный предупреждающий сигнал	164
Дополнительная литература	165
Глава 7. Новые роли	167
Роль Scrum-мастера	167
Качества хорошего Scrum-мастера	168
Технические лидеры в качестве Scrum-мастеров	171
Scrum-мастера: собственные или приглашенные	173
Ротация Scrum-мастеров	173
Преодоление типичных проблем	174
Владелец продукта	176
Обязанности владельца продукта	176
Каждой команде нужен только один владелец продукта	179
Качества хорошего владельца продукта	182
Scrum-мастер как владелец продукта	183
Преодоление типичных проблем	184
Новые роли, прежние обязанности	187
Дополнительная литература	187
Глава 8. Изменившиеся роли	189
Аналитики	190
Руководители проектов	192
Почему изменяется название должности	195
Архитекторы	196
Архитектор, не занимающийся кодированием	197
Функциональные менеджеры	197
Лидерская роль функционального менеджера	198
Обязанности, связанные с управлением кадрами	199
Программисты	200
Администраторы базы данных	202

Тестеры	202
Проектировщики пользовательского опыта	206
Три общие темы	208
Дополнительная литература	209
Глава 9. Технические приемы	211
Стремление к техническому совершенству	212
Проектирование на основе тестирования	212
Рефакторинг	215
Коллективное владение	217
Непрерывная интеграция	219
Парное программирование	221
Проектирование: целенаправленное и спонтанное	224
Привыкайте к жизни без крупномасштабного проектирования	225
Управление проектированием	227
Обязательное совершенствование технических приемов	230
Дополнительная литература	231
Часть III. Команды	235
Глава 10. Структура команды	237
Накормите команду двумя пиццами	238
Почему достаточно двух пицц	239
Производительность небольшой команды	241
Команды, специализирующиеся на определенных функциональных возможностях	244
Используйте компонентные команды экономно	246
Кто принимает эти решения	250
То, что правильно сегодня, может оказаться неправильным завтра	251
“Самоорганизующаяся” не означает “случайно сформированная”	251
Как подобрать подходящих членов команды	252
Один человек — один проект	254
Когда задач слишком много, время на их выполнение сокращается	256
Когда многозадачность является подходящим вариантом	257
Корпоративная форма многозадачности	258
Исключите однообразный механический труд	259
Рекомендации по выбору структуры команды	260
Что дальше	262
Дополнительная литература	263
Глава 11. Организация коллективного труда	265
Воспитание чувства коллективной ответственности	266
Воспитание у команды чувства приверженности поставленной перед ней цели	268
Полагайтесь на специалистов, не злоупотребляя ими	269
Старайтесь понемногу выполнять разные работы	271
Чтобы завершить все работы, не нужно ожидать окончания спринта	273
Балансируйте размеры элементов журнала запросов	273
Способствуйте обучению команды	274
Позаботьтесь о создании условий для обучения	275
Избегайте потерь знаний	280

Стимулируйте сотрудничество, напоминая о совместной цели	282
А теперь все вместе	285
Дополнительная литература	286
Глава 12. Управление самоорганизующимся коллективом	287
Влияние на самоорганизацию	288
Контеинеры, различия и обмены	289
Влияние на эволюцию команды или компании	296
Выбирайте внешнее окружение	297
Сформулируйте, что такое успешное функционирование	298
Управляйте смыслом	298
Внедряйте замещающие системы отбора	299
Подзаряжайте систему энергией	300
Лидерство — это не только покупка пиццы	302
Дополнительная литература	303
Глава 13. Журнал запросов на выполнение работ	305
Переход от документов к обсуждениям	306
Не выплесните ребенка вместе с документацией	309
Используйте в журнале запросов на выполнение работ историю пользователей	309
Постепенное уточнение требований	312
Спонтанно возникающие требования	312
Айсберг журнала запросов на выполнение работ	314
Зачем нужно постепенно уточнять требования	316
Постепенное уточнение историй пользователя	317
Учитесь начинать без спецификации	321
Создание спецификаций на основе примеров	322
Межфункциональные команды снижают потребность в документации	325
Критерии DEEP для журнала запросов	326
Не забывайте обсуждать	327
Дополнительная литература	327
Глава 14. Спринты	329
Каждый спринт должен заканчиваться созданием работоспособного программного обеспечения	330
Определение понятия “потенциально готов”	331
Рекомендации по определению понятия “потенциально готов”	333
Создавайте к окончанию каждого спринта что-то значимое	336
Готовьтесь в текущем спринте к следующему	340
Спринты типа “бильярдный шар”	340
Включайтe в спринт только реальный объем работы	341
Работайте вместе	343
Избегайте спринтов, посвященных определенным видам деятельности	345
Замените отношения “от окончания к началу” отношениями “от окончания к окончанию”	346
Распараллеливание проектирования пользовательского опыта	347
Мыслите целостно, работайте инкрементно	349
Архитектура и проектирование баз данных	350

Временные рамки: неизменные и строгие	352
Никогда не продлевайте спринт	354
Не меняйте цель	356
Откажитесь от перенацеливания команды	359
Получайте отзывы, учитесь и приспосабливайтесь	360
Дополнительная литература	361
Глава 15. Планирование	363
Постепенно уточняйте свои планы	364
Ни каких сверхурочных для спасения плана	366
Обходиться без сверхурочной работы трудно	368
Как прийти к финишу с наилучшим результатом	369
Если не сверхурочные, то что?	371
По возможности предпочтите изменение масштаба	372
Альтернативные варианты	373
Важность контекста конкретного проекта	376
Оценки и обязательства — это не одно и то же	378
Какими данными нужно располагать	378
Переход от оценки к обязательству	381
Исторически сложившаяся скорость создает основу для обязательств	383
Резюме	387
Дополнительная литература	388
Глава 16. Качество	389
Включайте тестирование в процесс разработки	390
Почему тестирование в конце не дает нужного результата	391
Как выглядит встраивание качества в процесс разработки на практике	392
Автоматизируйте на разных уровнях	394
Другие роли тестов пользовательского интерфейса	397
Роль тестирования вручную	397
Автоматизируйте во время спрингта	398
Выгоды от автоматизации тестов	400
Выполняйте разработку на основе приемочных тестов	401
Наиболее подходящий уровень детализации	403
Выплатите технический долг	405
Выплата технического долга в три этапа	406
Качество зависит от команды	407
Дополнительная литература	408
Часть IV. Организация	411
Глава 17. Изменение масштаба Scrum	413
Изменение масштаба роли владельца продукта	414
Совместная ответственность и разделение функциональности	415
Ведение большого журнала запросов на выполнение работ	416
Один продукт — один журнал	417
Поддерживайте разумный размер журнала	419
Упреждающий режим управления зависимостями	421
Планирование с накатом на несколько последующих спринтов	422

Проведение организационных совещаний, предваряющих очередной релиз-цикл	425
Совместное использование специалистов командами	426
Использование специализированной команды интеграции	426
Координация работы команд	429
Объединенное Scrum-совещание	430
Синхронизация спринтов	433
Изменение масштаба совещания по планированию спринтов	435
Смещение на один день	435
Большая комната	436
Создание сообществ практиков	438
Формальные или неформальные	439
Создание условий для формирования и процветания сообществ практиков	440
Участие	442
Scrum и масштаб проекта	443
Дополнительная литература	443
Глава 18. Рассредоточенные коллективы	445
Решите, как рассредоточить несколько команд	446
Добейтесь спаянности команды	449
Учитывайте культурные различия	450
Учитывайте незначительные культурные различия	452
Укрепляйте функциональную и командную субкультуры	453
Встречайтесь чаще	459
Визиты знакомства	459
Визиты с целью поддержания контактов	461
Путешествующие посланники	462
Измените способ своего общения	464
Объем документации растет	465
Добавление подробностей в журнал запросов на выполнение работ	466
Поощрение горизонтальных связей	466
Совещания	468
Советы общего характера	469
Совещание, посвященное планированию спринта	472
Ежедневное совещание разработчиков	475
Объединенные Scrum-совещания	479
Обзоры и ретроспективы спринтов	480
Действуйте с осторожностью	481
Дополнительная литература	482
Глава 19. Сосуществование с другими подходами	485
Совмещение Scrum и последовательной разработки	486
Три сценария взаимодействия	486
Три сферы конфликта	487
Могут ли Scrum и последовательная разработка сосуществовать вечно	489
Надзор за выполнением проекта	490
Выполнение Scrum-проектов, когда надзор не связан с гибкой методологией разработки	492

Соответствие стандартам	494
ISO 9001	494
Интегрированная модель технологической зрелости организации (CMMI)	496
Обеспечение соответствия	498
Что дальше	500
Дополнительная литература	500
Глава 20. Кадры, техническое обеспечение и отдел управления проектами	503
Отдел кадров	504
Структуры подчиненности	505
Периодические аттестации работников	506
Увольнение членов команды	509
Пути карьеры	510
Команда состоит из людей, а между людьми всегда возникают проблемы	511
Группа технического обеспечения	512
Физическое пространство	512
Мебель	516
О том, что должно быть хорошо видно в вашем рабочем пространстве	517
Отдел управления проектами	521
Люди	521
Проекты	522
Процесс	523
Переименование РМО	525
Подведем итоги	525
Дополнительная литература	526
Часть V. Что дальше	529
Глава 21. Как далеко вы продвинулись в деле внедрения Scrum	531
Цель измерения	531
Универсальные оценки освоения гибкой методологии разработки	532
Опрос на соответствие уровню “шодан”	534
Схема оценки степени освоения командой гибкой методологии разработки	535
Сравнительная оценка степени освоения командой гибкой методологии разработки	537
Разработка собственной оценки	540
Сбалансированная система показателей Scrum-команд	542
Построение сбалансированной системы показателей	543
Отдавайте предпочтение простым показателям	545
Должно ли это нас волновать	546
Дополнительная литература	548
Глава 22. Впереди еще много работы	551
Список литературы	553
Предметный указатель	565

“Недостаточно лишь понимать сам по себе механизм гибкой методологии разработки. Майк Кон предлагает вниманию читателей исчерпывающее собрание практических рекомендаций, которые помогут отдельным специалистам и коллективам в решении сложной задачи внедрения и адаптации процессов гибкой методологии разработки к специфическим задачам, которые приходится решать этим специалистам и коллективам. Эта книга станет авторитетным пособием для организаций, внедряющих у себя гибкую методологию разработки”.

— **Колин Берд (Colin Bird)**, Global Head of Agile, EMC Consulting

“Опыт Майка Кона, приобретенный им в процессе долговременного сотрудничества с множеством разных организаций в деле внедрения гибкой методологии разработки, проявляется в предлагаемых им практических подходах и принадлежащих ему ценных идеях. Если вы действительно хотите освоить гибкую методологию разработки, тогда эта книга — для вас”.

— **Джефф Хониус (Jeff Honious)**, вице-президент по инновациям, компания Reed Elsevier

“Майку Кону это удалось еще раз. Выщенная им книга основывается на его собственном (а также на всем нашем) опыте применения гибкой методологии разработки, приобретенном к настоящему времени. Майк охватывает в своей книге все стадии разработки проекта и предлагает ценные советы отдельным специалистам, коллективам и предприятию в целом. Какова бы ни была ваша роль в цикле гибкой методологии разработки, эта книга обязательно принесет вам пользу!”

— **Рон Джейферс (Ron Jeffers)**, www.Xprogramming.com

“Если вы хотите освоить гибкую методологию разработки программного обеспечения или повысить свою квалификацию в этой области, то эта книга — для вас. В ней рассматриваются проблемы, эффективные решения и полезные рекомендации, которые пригодятся вам при расширении масштаба проектов, выполняемых с помощью гибкой методологии разработки. Мы широко использовали рекомендации, приведенные в этой книге, при внедрении гибкой методологии разработки в одном из крупных подразделений, регулируемых Управлением по контролю за продуктами и лекарствами (США)”.

— **Крист Врайенс (Christ Vriens)**, глава подразделения MiPlaza, входящего в состав Philips Research

“Тем, у кого переход к использованию гибкой методологии разработки вызывает серьезные трудности, эта книга поможет понять, в чем тут дело. Майк Кон предлагает авторитетные и серьезные рекомендации по превращению вашей организации в высокоэффективное, новаторское и конкурентоспособное предприятие”.

— **Стиви Грини (Steve Greene)**, старший директор по управлению разработкой программного обеспечения и гибкой методологией разработки, www.salesforce.com

“Майк Кон — великий консультант в области эффективной трансформации организаций, занимающихся разработкой программного обеспечения. В этой книге представлена квинтэссенция опыта, приобретенного Майком за годы сотрудничества с

компаниями, которые пытаются достичь большей гибкости и маневренности. Если ваша организация ставит перед собой такую же цель, тогда эта книга — для вас”.

— **Кристофер Фрай (Christopher Fry)**, вице-президент по разработкам, www.salesforce.com

“Каким бы ни был уровень вашей квалификации в области Scrum, в книге Майка Кона вы найдете всю необходимую информацию для непрерывного совершенствования своих познаний в области гибкой методологии разработки. В этой книге теоретические концепции подкрепляются полезными советами, почерпнутыми автором из его повседневной практики. В частности, Майк советует, как отвечать на возражения колеблющихся сотрудников, и предлагает методы, которые можно (и нужно) испытать на практике сию минуту и которые активизируют мышление читателя. Обширный перечень рекомендованной литературы является последним доводом, который позволяет рассматривать эту книгу как настольное пособие для каждого специалиста, занимающегося разработкой программного обеспечения”.

— **Никки Ром (Nikki Rohm)**, директор студии управления проектами и ресурсами, Electronics Arts

“Первые шаги к совершенствованию вашего процесса разработки программного обеспечения посредством Scrum будут нелегки, и на каждом таком шаге вы будете сталкиваться с новыми проблемами. В своей книге Майк Кон показывает, как их преодолели другие организации и как вы можете воспользоваться их опытом, чтобы успешно внедрить у себя Scrum и выйти на путь постоянного совершенствования”.

— **Йоханес Бродуолл (Johanes Brodwall)**, главный научный сотрудник, Steria Norway

“Я начала рекомендовать всем своим знакомым специалистам новую книгу Майка Кона сразу же после того, как ее просмотрела. Как только кто-либо задавал мне вопрос по гибкой методологии разработки, я сразу же вспоминала ценные соображения на эту тему, недавно прочитанные мною в книге Кона. Я так рада, что эта книга, наконец-то, появилась в продаже, что повторяю буквально на каждом шагу: «Прочтите новую книгу Майка Кона — в ней вы найдете ответы на все свои вопросы!»

— **Линда Райзинг (Linda Rising)**, соавтор Мэри Линн Маннс (Mary Lynn Manns) по книге *Fearless Change: Patterns for Introducing New Ideas*

“Название этой книги говорит само за себя: это чрезвычайно проницательное и практическое руководство для тех, кто желает достичь успеха в разработке программного обеспечения с помощью гибкой методологии разработки. Если вы хотите знать все об этой методологии, тогда книга Майка Кона — для вас. Сейчас я готов рекомендовать ее всем своим клиентам!”

— **Хенрик Кнайберг (Henrik Kniberg)**, наставник по гибкой методологии разработки, член совета Agile Alliance, автор книги *Scrum and XP from the Trenches*

“В своей новой книге Майк Кон предлагает вниманию читателей сочетание теоретических сведений и ценных практических методов. Это еще одна великолепная книга Майка Кона, посвященная гибкой методологии разработки. Она окажет неоценимую

помощь вашей группе, вашему отделу и всей вашей организации и наверняка принесет вам успех”.

— **Мэтт Траксоу (Matt Truxaw)**, менеджер по поставкам приложений, Kaiser Permanente IT, дипломированный руководитель Scrum-проектов

“Новая книга Майка Кона представляет собой авторитетное руководство для компаний, внедряющих у себя Scrum. Представленный в ней материал тесно связан с практикой и доступен для понимания. Купите ее, прочтите ее и примените полученные знания на практике!”

— **Роман Пихлер (Roman Pichler)**, автор книги *Agile Product Management with Scrum*

“Эта книга не только чрезвычайно практична, но и заключает в себе очень глубокие мысли, а ее чтение доставляет истинное удовольствие. В ней сочетаются замечательные идеи и истории и примеры из деятельности организаций, занимающихся разработкой программного обеспечения. Эта книга рассчитана на широкий круг читателей, начиная с тех, кто желает приступить к внедрению в своей организации гибкой методологии разработки, и заканчивая разработчиками, стремящимися усовершенствовать способ, с помощью которого группа реализует отдельно взятый проект”.

— **Эндрю Стеллман (Andrew Stellman)**, разработчик, проект-менеджер и автор *Head First PMP, Beautiful Teams, Applied Software Project Management*

“Внедрение гибкой методологии разработки в небольшой компании, занимающейся разработкой проектов веб-приложений, на которые не накладываются какие-либо ограничения, связано со значительными трудностями. Но гораздо более сложное дело — трансформация целого предприятия. В этой книге рассказывается о проблемах, подобных тем, с которыми нам приходилось сталкиваться, и предлагаются не только объяснения этих проблем, но и, что гораздо важнее, практические подходы к их решению”.

— **Майкл Уоллин (Michael Wollin)**, старший менеджер по разработке, Broadcast Production Systems, CNN

“Майк Кон написал фантастическую книгу рекомендаций, пользуясь которыми, можно не только приступить к внедрению Scrum, но и превратить всю корпорацию в коллектив энтузиастов гибкой методологии разработки. Я уже реализовал многие из рекомендаций, содержащихся в этой книге, и вижу, как в нашей организации буквально каждый день увеличивается число сторонников Scrum”.

— **Джеймс Тишарт (James Tischart)**, CSM, CSP, CTFL, вице-президент по поставкам продукции, Mx Logic, Inc.

“В своей книге Майк Кон отразил не только собственный опыт, приобретенный в ходе выполнения огромного числа проектов разными группами и организациями, с которыми лично ему приходилось сотрудничать, но и коллективный опыт множества других организаций. Его книга изобилует многочисленными историями из практики, полезными данными и результатами исследований, бесценными соображениями относительно эффективности или неэффективности тех или иных способов внедрения, адаптации и наращивания масштабов применения Scrum. Лично мне в этой книге больше всего нравятся те места, где Майк высказывает собственные соображения по

поводу нескольких альтернативных подходов, а также обстоятельств, в которых каждый из этих подходов может оказаться наиболее эффективным”.

— **Брэд Эпплтон (Brad Appleton)**, консультант по гибкой методологии разработки в одной из телекоммуникационных компаний, входящих в перечень “*Fortune 100*”

“Я полагаю, что книга Майка Кона ответит на многие вопросы, с которыми приходится сталкиваться отдельным специалистам и целым коллективам и которые касаются способов улучшения сотрудничества, взаимодействия, качества и коллективной производительности. Мне особенно понравилось утверждение Майка о том, что в процессе, который предполагает непрерывное совершенствование, не может быть конечной точки. Это — нелегкий труд, требующий упорства, слаженности действий и проявления добной воли со стороны многих людей. Я хочу, чтобы все сотрудники моей организации прочитали эту книгу (точно так же, как они прочли книгу *Agile Estimating and Planning*”).

— **Скотт Спенсер (Scott Spencer)**, вице-президент по инжинирингу, First American CoreLogic, Inc.

“Майк Кон снова сделал это! В его всестороннем исследовании гибкой методологии разработки программного обеспечения описаны многочисленные приемы и методологии, позволяющие достичь успеха. Я рекомендую эту книгу каждому, кто желает приступить к применению гибкой методологии разработки или усовершенствовать используемый им процесс разработки программного обеспечения”.

— **Бенуа Хоули (Benoit Houle)**, старший менеджер по разработке, BioWare (подразделение Electronic Arts)

“Не приходится сомневаться в том, что новая книга Майка Кона станет настольной книгой по управлению проектами, связанными с разработкой программного обеспечения с применением Scrum. Материал книги скрупулезно выверен. Автор всеми силами избегал простых однозначных указаний по решению всех проблем. Несмотря на то что этот труд посвящен главным образом Scrum, Майк рассказывает и о других методах, и поэтому его книга стала всесторонним и полным пособием для специалистов по разработке программного обеспечения. Представленная информация заслуживает полного доверия, являясь отражением обширного личного опыта ее автора во всех затрагиваемых им вопросах”.

— **Филипп Крухтен (Philippe Kruchten)**, профессор в области проектирования программного обеспечения из Университета Британской Колумбии

“В этой книге вы найдете множество полезных советов по внедрению в вашей организации гибкой методологии разработки. Это руководство будет очень полезно для наставников и агентов изменений, которые сталкиваются с вызовами реального мира, такими как применение гибкой методологии разработки для рассредоточенных коллективов, и которые хотели бы внедрить эту методологию в масштабе всей организации. Мне нравится стиль Майка Кона, его стремление рассказать читателям о ситуациях, с которыми ему лично пришлось столкнуться на практике, и подкрепить свои соображения и выводы результатами соответствующих исследований. Из этой книги я узнала немало нового. Я уверена, что каждый, кто прочитает эту книгу, также существенно пополнит багаж своих знаний”.

— **Рейчел Дэвис (Rachel Davies)**, соавтор книги *Agile Coaching*

*Лоре, Саванне и Делани,
благодаря которым я могу с полным правом
называть себя знающим человеком*

Предисловие

Я постоянно слышу о том, что проекты, связанные с разработкой программного обеспечения, нужно рассматривать как “путешествия”. Однако, как мне кажется, такое определение подразумевает, что проекты следует рассматривать не просто как “путешествия”, а как “путешествия в неизведанное”. Мы ищем источник финансирования (спонсора проекта), формируем команду энтузиастов, определяем оптимальное, на наш взгляд, направление действий, а все остальное — это одиссея, путешествие в мир неизведенного. В ходе этого путешествия мы столкнулись с тем, что превосходно описано в легендах о смелом Одиссее — в легендах о лотофагах, циклопах, Цирцеи, сиренах, Сцилле, Калипсо... Наш успех зависит лишь от расположения богов. Как все это романтично — и до чего же глупо!

Лично мне кажется, что более подходящей аналогией проекта является экспедиция. У нас есть определенная цель (или краткий перечень целей). У нас есть достаточно надежные (или не совсем надежные) карты местности. Мы располагаем рекомендациями и путевыми заметками тех, кому уже приходилось бывать в этих краях.

Таким образом, выйдя за порог дома, мы отнюдь не попадаем в совершенно неизведанный мир. В то же время перед нами неминуемо возникают большие вопросительные знаки, а из этого следует, что нам не избежать крупного риска. Тем не менее мы идем на этот риск, поскольку в случае удачной экспедиции нас ожидает серьезное вознаграждение. Мастерства нам не занимать, но ситуация осложняется тем, что нам приходится действовать в условиях неопределенности.

Как же нам быть? Я посоветовал бы читателям вспомнить, что представляла собой примерно 300 лет тому назад Йоркская фактория на берегу Гудзонова залива (Канада). В то время Йоркская фактория была штаб-квартирой компании Hudson Bay Company. Основным направлением бизнеса Hudson Bay Company были поставки всего необходимого снаряжения и провианта для экспедиций, которые предпринимались охотниками на пушного зверя, промышлявшими на берегах Гудзонова залива. Охотники придумали великолепный способ начинать экспедиции, который они называли “Hudson Bay Start”. Совершив все необходимые закупки в Hudson Bay Company, они отправлялись в путь. Пройдя милю-другую, охотники останавливались и разбивали лагерь. Зачем? Конечно же, не для того, чтобы расставить капканы на зверя. Просто они пытались вспомнить, не забыли ли захватить с собой что-нибудь важное, пока еще не поздно вернуться в город и исправить обнаруженные упущения. Столь изощренный специалист по выполнению проектов, как вы, наверняка знаете, что для профессионального охотника на пушного зверя такие экспедиции являются не эпизодическим или случайным, а регулярным событием.

Какое отношение имеет вся эта история к книге, которую вы сейчас держите руках? Книга Майка Кона представляет собой своего рода “Hudson Bay Start” для тех, кому в наши дни приходится реализовывать проекты, связанные с разработкой программного обеспечения. Вот и все. Автор книги — прожженный “охотник на пушного зверя”. Он составил для вас подробный список необходимого снаряжения, сверившись с которым, вы можете смело отправляться в свою очередную экспедицию. Читая эту книгу, вы не раз убедитесь в том, что Майк Кон предусмотрел в своем списке буквально все — даже то, о чем вы и не помышляли. Он дает советы, как выходить из тех или иных затруднительных ситуаций, и помогает определить новые роли для членов команды.

Желательно, чтобы эту книгу прочитали не только вы, но и все члены вашей команды: когда речь идет о самоорганизующихся коллективах, каждый из членов такого коллектива должен быть готов к тому, что в любой момент ситуация может сложиться так, что ему придется выступить в роли лидера команды. Эта книга наверняка породит очень интересные дискуссии в вашем коллективе. Более того, я гарантирую это.

Меня несколько беспокоит следующее обстоятельство: на основании сказанного мною вам может показаться, будто своей книгой Майк не оставляет вам никакого выбора. Между тем с самого начала он указывает (а впоследствии неоднократно повторяет), что вам неминуемо придется делать выбор, связанный с индивидуальными, коллективными и организационными проблемами.

Книга, которую вы держите в руках, — не руководство по успешному выполнению какого-либо отдельно взятого проекта. Это книга о том, как совершить успешную карьеру “охотника на пушного зверя”.

Если вы все еще сомневаетесь в способности Майка Кона быть опытным руководителем охотничьей экспедиции, прошу обратить внимание на то, что его компания называется Mountain Goat Software.¹

Тим Листер (Tim Lister),
глава The Atlantic Systems Guild, Inc.
New York City

¹ Дословно — “Программное обеспечение «Горный козел»”. — Примеч. пер.

Благодарности

Я в огромном долгу перед своими официальными рецензентами: Брэдом Эпплтоном (Brad Appleton), Иоанном Бродуоллом (Johannes Brodwall), Рейчел Дэйвис (Rachel Davies), Роном Джейфрисом (Ron Jeffries), Брайаном Мариком (Brian Marick) и Линдой Райзинг (Linda Rising). Они прочитали и прокомментировали всю рукопись моей книги. Иногда им приходилось делать это по нескольку раз. Каждый из них высказал чрезвычайно ценные и глубокие соображения по поводу прочитанного, что, несомненно, способствовало повышению качества книги.

Особую благодарность мне хотелось бы выразить Тоду Голдингу (Tod Golding), Кенни Рубину (Kenny Rubin), Ребекке Тройгер (Rebecca Traeger) и моей жене Лоре, которая потратила не один час на обсуждение со мной оглавления этой книги. (Иногда нам казалось, что этим обсуждениям не будет конца.)

Я бесконечно благодарен Ребекке Тройгер. Она оказалась непревзойденным редактором, консультантом и “резонатором”. Как бывший редактор *Agile Alliance* и *Scrum Alliance* я готов утверждать, что Ребекка Тройгер — самая яркая индивидуальность в мире гибких методологий разработки, не говоря уже о том, что она — самый лучший на свете редактор. Она совершила с этой книгой буквально чудеса, действуя, как кулинар экстра-класса, готовящий какое-нибудь изысканное блюдо. Если бы не ее участие, эта книга потеряла бы многие из своих достоинств.

А предисловие Тима Листера (Tim Lister)! Написав его, он оказал мне особую честь. Я знаком с Тимом много лет и потому счел себя вправе попросить его написать предисловие к моей книге. Обратившись к нему по электронной почте, я не знал, что в это время он был в отпуске. Получив спустя неделю ответ от него на свой мобильный телефон, я почувствовал сильное волнение: мне очень хотелось, чтобы Тим согласился выполнить мою просьбу. Я буквально подпрыгнул до потолка от радости, когда узнал, что он согласен написать предисловие. А когда я ознакомился с текстом предисловия, я был необычайно польщен его словами в адрес моей книги. Спасибо тебе, Тим!

Моя помощница Дженинфер Раи (Jennifer Rai) оказывала мне неоценимую помощь в ходе реализации этого проекта. Она проделала огромную работу, начиная с отслеживания ссылок на литературу и заканчивая получением всевозможных разрешений и организацией моих исследований. Я высоко ценю ее прилежание, профессионализм и безусловную ответственность за порученное ей дело. Ничего большего я и не мог бы требовать от своего помощника.

Последние два года я выкладывал главы этой книги на специальном веб-сайте (www.SucceedingWithAgile.com). Мне посчастливилось, что нашлась группа энтузиастов, которые знакомились с содержимым этих глав и отправляли мне свои

комментарии по поводу прочитанного. Я хотел бы поблагодарить этих людей за чтение черновых набросков глав, выложенных мною на указанном сайте, и за то, что они поделились со мной собственными соображениями, многие из которых нашли свое отражение в моей книге. Вот эти люди: Фритьоф Альсведе (Fridtjof Ahlswede), Питер Альфвин (Peter Alfvin), Оле Андерсен (Ole Andersen), Джошуа Бойлтер (Joshua Boelter), Микаэл Боман (Mikael Boman), Роувен Баннинг (Rowan Bunning), Баттерскотч (Butterscotch), Билл Кэмпбелл (Bill Campbell), Мун-Вай Чанг (Mun-Wai Chung), Скотт Коллинз (Scott Collins), Джей Конни (Jay Conne), Джон Корнелл (John Cornell), Лайза Криспин (Lisa Crispin), Алан Дейли (Alan Dayley), Кен Делонг (Ken DeLong), Скотт Дункан (Scott Duncan), Зигфрид Даски (Sigfrid Dusci), Майк Дуайер (Mike Dwyer), Пабло Родригес Факал (Pablo Rodriguez Facal), Эбби Фichtner (Abby Fichtner), Хиллел Глейзер (Hillel Glazer), Керин Гривс (Karen Greaves), Джанет Грегори (Janet Gregory), Рата Граймс (Ratha Grimes), Гейр Хедемарк (Geir Hedemark), Фредрик Хедман (Fredrik Hedman), Бен Хоган (Ben Hogan), Мэтт Холмс (Matt Holmes), Сью Хольстад (Sue Holstad), Бенуа Хаули (Benoit Houle), Эрик Джимминк (Eric Jimmink), Куинн Джоунз (Quinn Jones), Мартин Кирнз (Martin Kearns), Джейфф Лангр (Jeff Langr), Пол Лиер (Paul Lear), Лоуэлл Линдстром (Lowell Lindstrom), Кэтрин Луис (Catherine Louis), Руни Маи (Rune Mai), Артем Марченко (Artem Marchenko), Кент Макдональд (Kent McDonald), Сьюзен Макинтош (Susan McIntosh), Алисия Маклейн (Alicia McLain), Ула Мерц (Ulla Merz), Ральф Майнер (Ralph Miner), Брайан Льюис Пейт (Brian Lewis Pate), Тронд Педерсен (Trond Pedersen), Дэвид Петерсон (David Peterson), Роман Пихлер (Roman Pichler), Уолтер Райес (Walter Ries), Адам Роджерс (Adam Rogers), Рене Розендал (Rene Rosendahl), Кенни Рубин (Kenny Rubin), Майк Рассел (Mike Russell), Майкл Сахота (Michael Sahota), Джордж Шлиз (George Schlitz), Лори Шубринг (Lori Schubring), Раффи Симонян (Raffi Simonian), Джейми Тишарт (Jamie Tischart), Райан Тун (Ryan Toone), Мэтт Траксоу (Matt Truxaw), Дж. Ф. Ансон (J. F. Unson), Сринивас Вадри (Srinivas Vaduri), Стефан ван ден Оорд (Stefan van den Oord), Бас Водде (Bas Vodde), Билл Уэйк (Bill Wake), Дэниел Вильдт (Daniel Wildt), Тронд Вингард (Trond Wingard), Редигер Вольф (Rudiger Wolf), Элизабет Вудворд (Elizabeth Woodward), Ник Ксидис (Nick Xidis), Алисия Яник (Alicia Yanik) и Маурицио Замора (Mauricio Zamora).

Спасибо Джейфу Шайху (Jeff Schaich), который создал великолепные иллюстрации к этой книге. Во момент знакомства с ним мне сказали, что он, так же, как и я, стремится во всем добиться совершенства. То, что это утверждение справедливо по крайней мере в отношении Джейфа Шайха, читатели могут убедиться сами.

Стивен Уилберс (Stephen Wilbers), автор книги *Keys to Great Writing* (“Как стать великим писателем”), с самого начала работы над этим проектом давал мне ценные советы и оказывал помощь в редактировании. Я благодарен Стивену за его советы и дружескую поддержку.

Как всегда, я испытывал огромное удовольствие, работая с сотрудниками издательства Pearson. Крис Гузиковски (Chris Guzikowski) проявил по отношению ко мне недюжинное терпение, особенно в самом начале нашей совместной работы, когда я не укладывался ни в один из сроков, установленных для меня издательством. Крис Зан (Chris Zahn) был моим бесценным наставником в те первые дни работы над книгой, когда я пытался каким-то образом организовать и спланировать свои действия. Джейк Макфарланд (Jake McFarland) был дизайнером внутренней части этой книги

и справился со своей работой, на мой взгляд, просто блестяще. Кроме того, Джейк проявлял поистине безграничное терпение, когда я донимал его постоянными вопросами по программному обеспечению InDesign, за что я чрезвычайно ему благодарен. Райна Хробак (Raina Chrobak) оказывала мне огромную помощь на протяжении всего времени реализации данного проекта, особенно когда мы вышли на финишную прямую — самый ответственный и напряженный период в реализации любого проекта.

Джована Сан-Николас Ширли (Jovana San-Nicolas Shirley) оказалась, безо всяко-го преувеличения, фантастическим редактором проекта, связанного с выпуском этой книги. Она добилась, чтобы этот проект продвигался с минимально возможным количеством помех, координируя действия всех его участников (особенно в последние месяцы). Я высоко ценю ее оперативные ответы на мои сообщения, отправлявшиеся по электронной почте практически в любое время суток. Сан Ди Филлипс (San Dee Phillips) безупречно справилась с окончательным редактированием моей книги. Я благодарен ей за скрупулезную вычитку текста и исправление остававшихся в нем мелких ошибок и недочетов, т.е. за работу, которую принято называть окончательной отделкой и шлифовкой текста.

Хотел бы также поблагодарить дизайнера обложки этой книги Алана Клементса (Alan Clements). Обложка получилась просто замечательная! Можно ли судить о книге (любой книге) по ее обложке? Я полагаю, что можно. Мое мнение основывается на отзывах ряда людей, которым понравилась эта книга. Лайза Стумпф (Lisa Stumpf) составила превосходный указатель к этой книге. Саму Лайзу следовало бы, на мой взгляд, отнести к разряду чрезвычайно дотошных и внимательных людей. Керин Джилл (Karen Gill) в последний раз вычитала текст, в результате чего ей удалось выявить кое-какие незначительные несоответствия и мелкие проблемы. Ким Скотт (Kim Scott) из Bumpy Design взяла на себя труд по разработке дизайна последней страницы. Я высоко ценю ее вклад, который помог нам уложиться в крайний срок, указанный издательством.

Я хотел бы также поблагодарить сотрудников издательства *Pearson* Криса Гузиковски (Chris Guzikowski) и Керин Геттман (Karen Gettman) за предоставленную мне возможность редактировать книги серии *Signature Series* издательства Addison-Wesley. Я хорошо помню, как в 1985 году я сидел за столом Кена Каплана (Ken Kaplan) в Бен-Ломонде (шт. Калифорния) и читал книгу *C Primer Plus*. Ее написал Стефан Прата (Stephen Prata), но она была частью серии, автором которой являлся Митчелл Уэйти (Mitchell Waite). В то время я не знал, что входит в задачи редактора серии, но название этой должности звучало, на мой взгляд, солидно и даже “крутого”. Сейчас я выполняю обязанности редактора серии и необычайно польщен оказанным мне доверием.

Хотелось бы также поблагодарить Лизу Адкинс (Lyssa Adkins), Лайзу Криспин (Lisa Crispin), Джанет Грэгори (Janet Gregory), Клинтона Кейта (Clinton Keith), Романа Пихлера (Roman Pichler) и Кенни Рубина (Kenny Rubin). Каждый из них написал (или пишет) книгу, входящую в эту серию. Нам пришлось изрядно подискутировать по поводу стиля изложения материала, а также по многим другим вопросам, касающимся написания книг этой серии. В ходе этих дискуссий мы достигали наилучшего понимания своих задач и оттачивали свое мастерство, что, несомненно, способствовало повышению качества и данной книги.

Особая благодарность — всем моим клиентам, а также тем, кто когда-либо посещал проводимые мною занятия. Я не могу похвастаться умением генерировать

оригинальные и великие идеи, сидя в кабинетной тиши. Всему, что я сейчас знаю, я научился в процессе работы с коллективами, наблюдая за тем, что удается и что не удается, а также общаясь со своими слушателями. Если бы не вы, объем этой книги составлял бы не более четырех страниц. Вот почему я так благодарен всем вам.

Хочу поблагодарить также Кена Швабера (Ken Schwaber), Джеффа Сазерленда (Jeff Sutherland), Майка Бидли (Mike Beedle), Джеффа Маккенна (Jeff McKenna), Мартина Девоса (Martine Devos) и всех остальных, кто причастен к рождению Scrum. Если бы не их статьи о Scrum, если бы не их выступления о Scrum на конференциях, Scrum не стал бы тем, чем он является сегодня. Также спасибо всем преподавателям и наставникам, входящим в сообщество Scrum, которые сделали так много не только для его популяризации, но и для того, чтобы он не оставался просто некой моделью, как он зачастую воспринимается в настоящее время. Мои беседы с многими из вас оказали на меня гораздо большее влияние, чем вы можете себе представить.

Нет таких слов, которыми я мог бы выразить признательность членам своей семьи за все жертвы, принесенные ими во имя того, чтобы предоставить мне как можно больше времени для написания этой книги. Невозможно даже мечтать о более преданной и любящей жене, чем моя Лора. Наши дочери, Саванна и Делани, остаются для меня такими же идеальными и бесценными принцессами, какими были всегда. Я дорожу буквально каждым мгновением общения с ними. Завершив написание этой книги, я обещаю уделять им гораздо больше своего времени и восполнить таким образом недостаток общения, вызванный необходимостью уложиться в срок, указанный издательством. Теперь настал мой черед показать членам моей семьи, как я всех их люблю!

Об авторе

Майк Кон является основателем компании Mountain Goat Software, занимающейся обучением и предоставляющей консалтинговые услуги в области Scrum и гибкой методологии разработки программного обеспечения. Майк специализируется на оказании помощи компаниям во внедрении Scrum и повышении их гибкости, позволяющей этим компаниям стать необычайно эффективными проектными организациями. Помимо данной книги, Майк Кон написал *User Stories Applied for Agile Software Development* и *Agile Estimating and Planning*, а также книги по программированию на Java и C++.

Обладая более чем 25-летним опытом, Майк Кон в свое время работал главным технологом в компаниях разного масштаба, начиная с компаний-новичков и заканчивая компаниями, входящими в перечень “Fortune 40”. Кроме того, он писал статьи для таких периодических изданий, как *Better Software*, *IEEE Computer*, *Cutter IT Journal*, *Software Test and Quality Engineering*, *Agile Times* и *C/C++ Users Journal*. Майк часто выступает с докладами на отраслевых конференциях и является членом-основателем Agile Alliance и Scrum Alliance. Кроме того, он является дипломированным преподавателем Scrum. В мае 2003 года он совместно с Кеном Швабером (Ken Schwaber) провел первый курс обучения по программе Certified ScrumMaster.

Более подробные сведения о Майке Коне приведены на сайте www.mountaingoatsoftware.com. К нему также можно обратиться через Twitter ([@mikewcohn](https://twitter.com/mikewcohn)) или по электронной почте (mike@mountaingoatsoftware.com).

Введение

Эта книга не для тех, кто совершенно не знаком со Scrum или гибкой методологией разработки программного обеспечения. Для новичков существуют другие книги, учебные курсы и даже веб-сайты. Если вы совсем не знаете Scrum, начните с другого источника². Данная книга и не для борцов за “чистоту идеи”. Они могут найти немало блогов, на которых отстаивается единственный, “истинный” способ использования гибкой методологии разработки программного обеспечения, или Scrum. Эта книга для программистов, для тех, кто уже начал использовать Scrum, но столкнулся с проблемами, или для тех, кто еще не начал использовать Scrum, но намерен это сделать. Им нет нужды снова читать о том, как составить график выполнения оставшихся работ или какие три вопроса задает каждый человек на ежедневном совещании разработчиков. Им нужны советы по более сложным вопросам, например как внедрить Scrum, как заставить людей уже в самом начале реализации проекта приступить к крупномасштабному проектированию, как разработать программное обеспечение, которое осталось бы работоспособным к концу каждого спринта, чем занимаются менеджеры и т.п. Если вас интересуют именно такие вопросы, значит, эта книга для вас.

Чтобы ответить на эти, а также на другие подобные вопросы, я воспользовался собственным опытом применения методологии Scrum. Особенно это относится к последним четырем годам моей работы со Scrum. На протяжении этих четырех лет, проведя очередной день с одним из моих клиентов, я возвращался в свой номер в гостинице и подробно фиксировал в блокноте проблемы, с которыми столкнулся этот клиент, вопросы, которые он мне задавал, и мои советы ему. Затем я отслеживал ситуацию в ходе своих последующих контактов с каждым из клиентов. Я хотел знать наверняка, какие из моих советов принесли реальную пользу в решении проблем, волновавших клиентов.

Накапливая перечень этих вопросов, проблем и моих советов клиентам, я нашупывал некие “глобальные” темы. Одни препятствия были совершенно уникальными для кого-то из клиентов или какой-либо одной группы разработчиков. Другие носили более универсальный, общий характер, повторяясь у многих групп разработчиков и организаций. Основу этой книги составляют именно универсальные проблемы и мои советы по их решению. Свои советы я выделял в тексте двумя способами. Во-первых, большинство глав включает врезки, озаглавленные “Попробуйте прямо сейчас”. В этих врезках приведены советы, которые я давал чаще других или которые были наиболее эффективными в тех или иных конкретных ситуациях. Во-вторых, большинство

² Неплохой отправной точкой может служить www.mountaingoatsoftware.com/scrum.

глав включает врезки “Возражение”. В этих врезках я попытался воспроизвести типичный разговор, в ходе которого мой собеседник не соглашался с моей точкой зрения на рассматриваемую проблему. Знакомясь с этими возражениями, попытайтесь услышать голос кого-либо из своих сотрудников. Подозреваю, что вам уже доводилось выслушивать нечто подобное. Знакомясь с содержимым этих врезок, вы увидите, как я отвечал на такие возражения.

Какими качествами должны обладать мои читатели

Помимо высказанного мною ранее предположения о том, что вы знакомы с основами Scrum, а сейчас хотите либо внедрить эту методологию в своей организации, либо углубить свои познания в ней, я исхожу из того, что вы обладаете определенным влиянием в своей организации. Это не означает, что моя книга предназначена исключительно для директоров, вице-президентов и глав корпораций. Упомянутое мною влияние должно обуславливаться — независимо от занимаемой вами должности — особенностями вашей личности, вашим авторитетом в организации и доверием к вам со стороны ваших сотрудников. Разумеется, наличие высокого поста способствует росту такой влиятельности. Но, как будет показано, типом влияния, необходимого для успешного внедрения и применения Scrum, чаще всего обладают неформальные лидеры, которыми обычно являются специалисты, пользующиеся высоким авторитетом среди своих коллег.

Как организована эта книга

Когда я начинал работу над этой книгой четыре года назад, я выбрал для нее такое рабочее название: “С чего начать и как стать квалифицированным разработчиком” (именно в этом я и хотел помочь своим читателям). Подбирая материал для книги, я осознал, что одно неотделимо от другого: начав осваивать Scrum, вы обязательно станете квалифицированным разработчиком программного обеспечения. К тому же нет каких-то особых методов, необходимых для начала освоения Scrum, и принципиально иных методов, позволяющих разработчику программного обеспечения отшлифовать свое мастерство в области применения этой методологии.

В части I, “Приступаем к освоению Scrum”, речь идет о том, как приступить к освоению Scrum. В ней приводятся советы, как лучше изучать Scrum: постепенно, шаг за шагом, или “одним махом” — решительно и одномоментно. Вы узнаете, как помочь людям перейти от осознания необходимости в каком-то новом процессе, к желанию перемен, а затем и к выработке у себя способности осуществить такие перемены. В ней рассказывается также о том, как выбирать исходные проекты и команды исполнителей. Базовыми механизмами, описанными в этой части, вы будете пользоваться не только для начала освоения Scrum, но и для совершенствования своего мастерства в области применения этой методологии. К числу таких механизмов относятся, в частности, сообщества в поддержку усовершенствования организации и журналы совершенствования, речь о которых идет в главе 4, “Движение в направлении гибкости”.

В части II, “Люди”, основное внимание уделяется людям и изменениям, через которые должен пройти каждый сотрудник в ходе внедрения Scrum. Из главы 6, “Преодоление сопротивления”, вы узнаете о сопротивлении переменам, которое могут оказывать отдельные лица. В этой главе приводятся причины, вызывающие такое сопротивление, и советы относительно того, как помочь человеку осознать пагубность такого сопротивления и отказаться от него. В главах 7, “Новые роли”, и 8, “Изменившиеся роли”, описываются новые роли, которые должны присутствовать в любом Scrum-проекте, и трансформации, которым должны подвергнуться роли традиционные, такие как программист, тестер, руководитель проекта и т.п. В главе 9, “Технические приемы”, описываются определенные технические приемы (такие, как непрерывная интеграция, парное программирование, проектирование на основе результатов тестирования и т.п.), которыми следует пользоваться на практике (или с которыми по крайней мере желательно поэкспериментировать) и которые могут серьезно повлиять на то, как люди подходят к выполнению своей повседневной работы.

В части III, “Команды”, мы перейдем от отдельных людей к коллективам. Сначала мы рассмотрим, как структурировать такие коллективы, чтобы достичь максимального эффекта от использования Scrum. В главе 11, “Организация коллективного труда”, рассматривается природа коллективной работы над Scrum-проектом. Из главы 12, “Управление самоорганизующимся коллективом”, вы узнаете, что значит руководить самоорганизующимся коллективом, работающим над каким-либо Scrum-проектом. В этой главе приведены конкретные советы о том, что могут сделать руководитель Scrum-проекта, функциональные менеджеры и другие руководители, чтобы помочь коллективу самоорганизоваться для достижения успеха. Главы с 13, “Журнал запросов на выполнение работ”, по 15, “Планирование”, завершают часть обсуждением спринтов, планирования и качества.

В части IV, “Организация”, мы поднимемся на еще более высокий уровень — от коллектива к организации. В главе 17, “Изменение масштаба Scrum”, под более широким углом зрения рассматривается, что необходимо для того, чтобы путем увеличения масштаба Scrum обеспечить возможность применения этой методологии к крупным проектам, над реализацией которых трудится несколько коллективов. В главе 18, “Рассредоточенные коллективы”, речь пойдет о дополнительных сложностях, связанных с работой рассредоточенных коллективов. Из главы 19, “Со существование с другими подходами”, вы узнаете, как применять Scrum, если часть проекта использует последовательный процесс или предъявляются требования их совместимости или управляемости. Часть завершается главой 20, “Кадры, техническое обеспечение и отдел управления проектами”, посвященной особым соображениям, касающимся влияния Scrum на кадры организации, ее техническое обеспечение и группы отдела управления проектами.

Часть V, “Что дальше”, состоит из двух глав. В главе 21, “Как далеко вы продвинулись в деле внедрения Scrum”, описаны подходы к измерению степени продвижения вашей организации в достижении подлинной гибкости. Глава 22, “Впереди еще много работы”, завершает книгу напоминанием о том, что поддержание подлинной гибкости требует постоянного совершенствования. Не важно, какой степени совершенства вы достигли к данному моменту. Чтобы быть действительно гибким, нужно совершенствоваться всегда.

Замечания по поводу некоторых терминов

Как обычно, писать о методике Scrum сложнее, чем рассуждать о ней. К тому же любое предложение можно неправильно истолковать или вырвать из контекста. Чтобы избежать подобных проблем, я пытался быть как можно более точным в использовании определенных терминов. Например, слово *разработчик* (*developer*) я употребляю для обозначения любого из лиц, которые принимают участие в разработке проекта. Это относится к программистам, тестерам, аналитикам, проектировщикам пользовательского интерфейса (*user experience designer*), администраторам баз данных и др.

Понятие *команда*, или *коллектив* (*team*) порождает свои проблемы. Оно, конечно же, включает разработчиков, но включает ли оно руководителя Scrum-проекта и владельца продукта? Естественно, это зависит от контекста. Когда я хотел добиться предельной ясности, я использовал термин *весь коллектив* (*whole team*), имея в виду всех: разработчиков, руководителя Scrum-проекта и владельца продукта. Однако бездумное применение термина *весь коллектив* наверняка ухудшило бы удобочитаемость этой книги. Поэтому вам будет встречаться также термин *коллектив*, но, как правило, в тех местах, где из самого контекста понятно, о какой именно группе людей идет речь.

Когда речь шла о коллективах, пользующихся методологией Scrum или agile (гибкая методология разработки), мне также требовался термин для обозначения коллективов, которые *не* пользуются указанными методологиями. В разных местах я употреблял термины *последовательная* (*sequential*), *традиционная* (*traditional*) и *негибкая* (*non-agile*) методология. Каждый из них заключает в себе несколько отличный от других смысл и используется соответствующим образом.

Как пользоваться этой книгой

Многие книги в наше время включают раздел с подобным названием. Правда, в них обычно рассказывается, как читать соответствующую книгу. Наилучший способ читать данную книгу — пользоваться ею. Ее недостаточно просто читать. Встретив раздел “Попробуйте прямо сейчас”, постарайтесь испытать соответствующий метод “прямо сейчас”. Или по крайней мере зафиксируйте его в своем блокноте и попробуйте применить его на своем очередном ретроспективном совещании или совещании по планированию, если я рекомендую это.

Вовсе необязательно читать эту книгу от первой главы до последней в указанном порядке. Возможно, какие-то главы вам вообще не понадобится читать. Если ваша организация, стремящаяся внедрить у себя Scrum, не испытывает особых проблем с планированием или не задействует при выполнении проектов рассредоточенные коллективы, вам нет нужды читать соответствующие главы. Тем не менее я рекомендую в любом случае прочесть хотя бы первые четыре главы, причем именно в такой последовательности, в какой они приведены в этой книге. В них заложен фундамент всего последующего материала.

В главе 4, “Движение в направлении гибкости”, вы познакомитесь с идеей сообществ в поддержку усовершенствования организаций и журналов совершенствования. Сообщество в поддержку усовершенствования организаций (*improvement community*) — это группа единомышленников, стремящихся к усовершенствованию

в соответствующей области. Одно из таких сообществ может, например, сформироваться, когда трое специалистов, занимающихся журналом запросов на выполнение работ, принимают решение накапливать передовой опыт и рекомендации, представляющие большую ценность для всех коллективов. Другое сообщество может включать сотни людей, заинтересованных в совершенствовании методов тестирования приложений, используемых вашей организацией. Журнал совершенствования — это расположенный в порядке убывания приоритетов перечень дел, которыми сообщество в поддержку усовершенствования организации намеревается помочь своей организации добиться более высоких результатов.

Я очень надеюсь, что сообщества в поддержку усовершенствования организации — в том числе сообщество в поддержку перехода к Scrum (Enterprise Transition Community), которое вдохновляет и направляет действия, связанные с трансформацией предприятия — воспользуются этой книгой для заполнения своих журналов совершенствования. Вообще говоря, многие из заголовков разделов верхнего уровня преднамеренно выбраны мною таким образом, чтобы их можно было без каких-либо изменений переносить в журналы совершенствования.

Будучи опытным инструктором и консультантом по Scrum, я успел поработать с несколькими сотнями коллективов и организаций. В результате я пришел к выводу, что успех Scrum возможен практически в каждой организации. Разумеется, одним из организаций это дается тяжелее, чем другим. Одним из них мешает слишком жесткая корпоративная культура. Другим приходится сталкиваться с сопротивлением хорошо окопавшихся, “неподатливых” работников, опасающихся, что будут затронуты их личные интересы. Организации, которым в этом отношении повезло больше, чем остальным, располагают лидерами и рядовыми исполнителями, готовыми всемерно поддерживать перемены. Однако общей для всех этих организаций является потребность в прагматичных и надежных советах. Преследуя именно такую цель, я и написал эту книгу.

Часть I

Приступаем к освоению Scrum

Готовность к переменам является преимуществом, даже если начало таких перемен будет означать погружение какой-то части компании на некоторое время в полный хаос.

— Джек Уэлч (Jack Welch)

Глава 1

Гибкая методология разработки: нелегко, но перспективно

Многие из организаций, занимающихся разработкой программного обеспечения, пытаются сделать свои процессы разработки более гибкими. И можно ли что-то возразить против этого? Успешные команды, применяющие в своей практике гибкую методологию разработки, создают высококачественное программное обеспечение, которое максимально удовлетворяет потребности пользователей, причем быстрее и с меньшими издержками, чем команды, применяющие традиционные методы разработки. В конце концов, есть ли кто-то, кто не хотел бы стать более гибким, тем более если это приводит к желаемым результатам? Организации, которые относятся к использованию гибкой методологии разработки (такой, как Scrum) со всей серьезностью, получают огромные преимущества перед своими конкурентами.

Эти преимущества выражаются в более высокой производительности и снижении издержек. Организации, применяющие гибкую методологию разработки, выводят свои новые продукты на рынок гораздо быстрее, и при этом степень удовлетворенности клиентов такими продуктами гораздо больше. Таким организациям удается намного эффективнее контролировать процесс разработки, что обуславливает большую предсказуемость результата. Для таких организаций далеко позади остались времена, когда им не удавалось в достаточной степени управлять работой над своими проектами, что неминуемо приводило как минимум к срыву сроков их завершения.

Одной из компаний, которые воспользовались преимуществами внедрения Scrum, является компания Salesforce.com, занимающаяся разработкой программного обеспечения. История этой компании, основанной в 1999 году в Сан-Франциско, является ярким примером историй успеха в эпоху расцвета интернет-компаний. Salesforce.com, доход которой в 2006 году превышал 450 миллионов долларов, а число сотрудников составляло две тысячи человек, обратила внимание на то, что частота выпуска ее продуктов сократилась с четырех выпусков в год до одного. Клиенты получали все меньше продуктов, а ожидать выхода новой версии им приходилось все дольше.

Короче говоря, нужно было срочно что-то менять. Компания решила перейти к использованию Scrum. В течение первого года, когда происходило внедрение Scrum, Salesforce.com предложила на 94% больше новых возможностей, чем прежде, обеспечив на 38% больше новых возможностей программного обеспечения в расчете на одного разработчика. В последующие два года доходы Salesforce.com повысились более чем в два раза и составили более миллиарда долларов. Учитывая такие результаты, неудивительно, что Scrum завоевала популярность у столь большого числа организаций (хотя и не все организации, которые пытались внедрить у себя Scrum, довели это дело до логического завершения).

Я написал “пытались”, поскольку переход к использованию Scrum и других гибких методологий разработки — не такое уж простое дело (во всяком случае это гораздо сложнее, чем кажется многим компаниям). Изменения, которые требуется осуществить для получения максимальных преимуществ от использования гибких методологий разработки, довольно обширны. Эти изменения предъявляют высокие требования не только к разработчикам, но и ко всем остальным сотрудникам организации. Изменить мышление и психологию людей — это далеко не то же самое, что изменить методы работы. Цель этой книги — показать не только то, как перейти к использованию Scrum, но и как достичь успеха в долгосрочной перспективе.

Я был свидетелем нескольких неудачных попыток внедрения Scrum. Между тем эти неудачи вполне можно было предотвратить. Первый из случаев относится к компании, которая потратила свыше миллиона долларов на переход к использованию Scrum. Руководители этой компании привлекли сторонних инструкторов и наставников, приняли на работу в “Отдел гибкой методологии разработки” пятерых специалистов (этот отдел был создан специально для того, чтобы в него могли обращаться за консультациями члены вновь создаваемых Scrum-групп). Неудача этой компании была обусловлена ложными представлениями руководства о том, что внедрение Scrum касается только команд разработчиков. Руководители, которые инициировали этот переход, полагали, что достаточно лишь обеспечить обучение и поддержку разработчиков, и совершиенно не задумывались о том, что внедрение Scrum может затрагивать торговых представителей компаний, группу маркетинга и даже финансовый отдел. Без изменений в этих подразделениях внедрение Scrum было заранее обречено на провал.

Неудачное внедрение Scrum в другой компании было обусловлено совершенно иными причинами. Методика Scrum сразу же приглянулась новоназначенному руководителю проекта Джозефу, поскольку она полностью отвечала стилю управления, присущему Джозефу. Джозеф сразу же убедил свою группу испытать Scrum на их новом проекте (еще за месяц до того члены этой группы были коллегами Джозефа, хорошо знали его и полностью ему доверяли). Предстоящий проект был многообещающим как для команды разработчиков, так и для самого Джозефа, который в случае успеха мог претендовать на должность руководителя гораздо более крупного проекта. Джозеф познакомил со Scrum всех участников проекта и большинство из них горело желанием испытать на практике этот новый подход. Несмотря на то что сами участники проекта были готовы использовать Scrum, некоторые из руководителей, которым подчинялась команда разработчиков, занервничали, полагая, что Scrum может таить в себе угрозу их служебной карьере. Джозефу не повезло. Руководители — в частности, менеджеры по обеспечению качества и по разработке баз данных — объединились

и убедили вице-президента по инжинирингу в том, что Scrum не годится для столь сложных и важных проектов, как те, которые реализуются в их компании.

В этом отношении Кэролайн повезло несколько больше. Под руководством Кэролайн, занимающей пост вице-президента по разработкам в крупной компании, специализирующейся на управлении данными, трудилось свыше 200 разработчиков. Убедившись в преимуществах использования Scrum на примере одного из проектов, Кэролайн с энтузиазмом начала пропагандировать внедрение Scrum в своем подразделении. Всем сотрудникам была предоставлена возможность научиться использованию Scrum и получать необходимые консультации. Уже через несколько месяцев почти все группы разработчиков создавали работоспособные программы в конце каждого двухнедельного спринта. Это был впечатляющий прогресс. Однако когда спустя год я посетил эту компанию, ее сотрудники не продемонстрировали сколько-нибудь заметного прогресса. Да, группы разработчиков создавали высококачественные программы несколько быстрее, чем до внедрения Scrum, но достижения этой компании оказались гораздо скромнее, чем могли бы быть. Просто Кэролайн забыла, что необходимым условием успешного применения Scrum является постоянное совершенствование.

Такие примеры настораживают, не так ли? Общим для всех этих примеров неудачного применения Scrum является то, что внедрение Scrum обусловливалось самыми добрыми намерениями, однако добрые намерения — хотя и необходимое, но все же далеко не достаточное условие успеха. Тем не менее это вовсе не повод опускать руки. Переход к использованию Scrum может даться нелегко, но ничего невозможного здесь нет, если только применить правильный подход. В этой главе мы рассмотрим, чем обусловливается трудность перехода к гибкой методологии разработки (в любом ее варианте, включая Scrum). Мы подробно проанализируем некоторые из проблем, ставших причиной неудач в упомянутых мною компаниях. Что гораздо важнее, мы рассмотрим причины, по которым преимущества применения гибкой методологии разработки стоят усилий, требующихся для внедрения такой методологии.

Чем обусловлены трудности перехода к использованию Scrum

Любые перемены даются нелегко. Я сам был свидетелем глубоких переживаний работников по поводу даже такого, казалось бы, малозначительного события, как изменение плана медицинского страхования в их компании. Более серьезные перемены могут оказаться куда болезненнее. Но у перехода к использованию Scrum есть свои особенности, которые делают эту трансформацию даже более сложной, чем большинство других преобразований. Эти особенности таковы.

- Успешное изменение не осуществляется исключительно по принципу “сверху вниз” или “снизу вверх”
- Конечное состояние такого перехода непредсказуемо
- Scrum проникает буквально во все поры организации
- Scrum — очень своеобразная методология
- Перемены происходят быстрее, чем прежде
- Передовые методы таят в себе опасность

Успешное изменение не осуществляется исключительно по принципу “сверху вниз” или “снизу вверх”

Успешные организационные преобразования не могут происходить исключительно по принципу “сверху вниз” или “снизу вверх”. В случае изменений “сверху вниз” авторитетный лидер делится с сотрудниками своим видением будущего и организация осуществляет трансформацию согласно этому видению. Представьте себе, что такой харизматичный, уважаемый и могущественный лидер, как Стив Джобс (Steve Jobs), говорит сотрудникам своей компании Apple, что их организация не должна ограничиваться лишь разработкой компьютерного “железа” и программного обеспечения, а должна обеспечить себе доминирующее положение в сфере цифровой музыки. Его репутация и стиль управления могли бы задать для компании Apple новое направление, но одного только этого было бы все же недостаточно, чтобы совершить столь грандиозный поворот. Эксперт в вопросах управления переменами Джон Коттер (John Kotter) согласен с таким утверждением.

Никакая отдельно взятая личность, даже глава компании, привыкший к тому, чтобы ему беспрекословно повиновались, не в состоянии выработать верное представление о будущем, донести это представление до большого числа людей, устранить важнейшие препятствия, обеспечить успех на этом пути в краткосрочной перспективе, руководить десятками проектов, связанных с реализацией намеченных им перемен, и глубоко внедрить новые подходы в культуру соответствующей организации (1996, 51–52).

В отличие от изменений, осуществляемых по принципу “сверху вниз”, в случае изменений “снизу вверх” некая группа людей или отдельные сотрудники принимают решение о необходимости определенных перемен и приступают к их воплощению. Одни команды реализуют намеченные перемены, руководствуясь принципом “попросим прощения позже”. Другие бравируют тем, что действующие правила им ни почем. Третьи пытаются как можно дольше находиться вне поля зрения “корпоративного радара”.

Самые успешные перемены — и особенно переход к использованию гибкой методологии разработки, подобной Scrum — должны включать элементы изменений, осуществляемых по принципу “сверху вниз”, *а также* элементы изменений “снизу вверх”. В своей книге *Fearless change: Patterns for introducing new ideas* Мэри Линн Маннс (Mary Lynn Manns) и Линда Райзинг (Linda Rising) соглашаются с этим утверждением: “Мы полагаем, что изменения лучше всего инициировать снизу — с поддержкой в определенные моменты со стороны руководства как на локальном, так и на более высоком уровне” (2004, 7). Организация, пытающаяся перейти к использованию Scrum без поддержки со стороны руководства, неминуемо столкнется с сопротивлением, которое невозможно преодолеть снизу. Это обычно происходит, как только новый Scrum-процесс начинает оказывать влияние на выполнение работы людьми и подразделениями, находящимися за рамками первоначальной группы энтузиастов. В ответ руководители среднего звена защищают свои подразделения, ополчаясь против перемен, порождаемых использованием Scrum. Для устранения подобных помех и препятствий потребуется поддержка сверху.

Аналогично без поддержки снизу переход к использованию Scrum будет похож на следующую бессмысленную ситуацию: оказавшись в Мехико, вы приходите в

ресторанчик, расположенный на тротуаре под тентом, и, пытаясь хоть немного освежиться, усаживаетесь у вентилятора, подвешенного под потолком. Но напрасно вы надеетесь на спасительную прохладу: вместо этого вас с потолка обдает горячим воздухом. Когда переход к использованию Scrum осуществляется без поддержки снизу, люди не очень-то склонны делать то, чего от них требуют. Понимание и поддержка снизу нужны постольку, поскольку именно рядовым исполнителям проектов (т.е. членам команд разработчиков) придется разбираться в конкретных вопросах и тонкостях использования Scrum и добиваться оптимального эффекта от такого использования.

Таким образом, ключом к успешному внедрению Scrum является сочетание элементов изменений, осуществляемых по принципу “сверху вниз”, с элементами изменений, осуществляемых по принципу “снизу вверх”.

Конечное состояние перехода к Scrum непредсказуемо

Возможно, вы уже прочитали одну из книг по экстремальному программированию и решили, что этот метод подходит для вашей компании. Возможно, вы посещали курсы обучения для тех, кто желает получить квалификацию ScrumMaster, и считаете, что Scrum — это круто. Возможно, вы читали книгу, посвященную какой-либо другой гибкой методологии разработки, и считаете ее более подходящей для своей организации.

Скорее всего, вы заблуждаетесь.

Ни одна из этих методологий — в таком виде, в каком они описаны их творцами — не является идеальной для вашей организации. Любая из них может быть хорошей отправной точкой, но вам придется приспособить ее к уникальным особенностям вашей организации, ваших разработчиков и вашей отрасли. Алистер Кокбурн (Alistair Cockburn) приходит к следующему выводу: “Адаптация какого-либо процесса к конкретным обстоятельствам определенной группы его пользователей является, по-видимому, критическим фактором успеха и необходимым условием внедрения этого процесса. Именно этот творческий акт — акт адаптации — обуславливает привязанность к данному процессу группы пользователей, выполнивших такую адаптацию”.¹

У вас может быть четкое представление о том, что для вас лично означает “использование Scrum”, и вы можете убедить других в правильности такого представления. Но из этого представления еще не следует, в чем будет заключаться в конечном счете использование Scrum вашей организацией. Более того, было бы даже неправильным употреблять выражение “в конечном счете” применительно к внедрению и использованию Scrum, поскольку для процесса, который предполагает постоянное совершенствование, вообще не может существовать какого-то конечного состояния.

Это создает проблему для организации, желающей перейти к использованию Scrum посредством традиционного метода изменений, который базируется на анализе просчетов прогнозирования и последующем исправлении выявленных просчетов. Если мы не можем предсказать, каким будет конечное состояние перехода к использованию Scrum, то мы не можем выявить все расхождения между этим конечным состоянием и текущим состоянием. Иначе говоря, метод изменений, базирующийся на анализе просчетов прогнозирования, в данном случае непригоден. Самое большее, что мы можем сделать, — это выявлять расхождения между текущим состоянием и неким “улучшенным”, промежуточным состоянием.

¹ Это, а также остальные соображения высказывались в ходе личного общения между их автором и мною.

Однако после выявления этих меньших расхождений у нас по-прежнему остается нерешенной проблема их устранения (т.е. исправления выявленных просчетов). Трудно (а зачастую и невозможно) точно предсказать, как будут реагировать люди на множество небольших изменений, которые потребуются в ходе внедрения гибкой методологии разработки. Эксперт по коллективной работе Кристофер Эвери (Christopher Avery) рассматривает организаций как живые системы.

Мы не можем направлять живую систему. Мы можем лишь как-то воздействовать на нее и анализировать ее реакцию на эти воздействия... Мы не знаем всех сил, формирующих организацию, которую мы хотим изменить, поэтому все, что мы можем сделать, — это тем или иным способом провоцировать данную систему путем экспериментов с силой, которая, как нам кажется, может оказывать какое-то влияние, а затем смотреть, каким будет результат нашего воздействия (2005, 22–23).

Итак, переход к использованию Scrum не может представлять собой процесс, который “формулирует и определяет весь процесс изменений, требующийся для устранения расхождения между текущим и конечным состояниями, и создает тактические планы”, как говорится в одной из книг, посвященных традиционному методу изменений (Carr, Hard, and Trahant, 1996, 144–5). Создание такого плана потребовало бы преодоления двух непреодолимых барьеров: во-первых, точного знания, где же мы хотим оказаться в конечном счете, и, во-вторых, точного знания шагов, которые приведут нас к этому конечному состоянию. Поскольку мы не можем преодолеть эти барьеры, самое лучшее, что мы можем сделать, — воспользоваться подходом “воздействовать и анализировать реакцию” (Avery, 2005, 23), т.е. осуществлять какое-то действие, наблюдать, приближает ли оно нас к промежуточному, “улучшенному” состоянию, и, если приближает, то продолжать в том же духе, т.е. предпринимать такое же действие. Такие итерации, осуществляемые организацией, вовсе не носят хаотический, произвольный характер. Они тщательно выбираются на основе имеющегося опыта, запаса знаний и интуиции, которые и должны обеспечить успешный переход к использованию Scrum.

См. также В главе 4, “Движение в направлении гибкости”, описан рекомендованный мною процесс внедрения Scrum.

Scrum проникает буквально во все поры организации

Когда какое-либо изменение носит изолированный характер и не влияет на все, что делает тот или иной человек, внедрить его в организации зачастую оказывается легче. Рассмотрим случай организации, которая использует какой-либо негибкий процесс и решает внедрить у себя обязательный отчет по операционной готовности, причем такой отчет должен составляться еще до того, как то или иное приложение будет размещено на веб-серверах компании. Это может служить примером относительно изолированного изменения. Конечно, кому-то из разработчиков эта новая процедура не понравится; они будут ворчать (возможно, даже вслух). Однако это изменение отнюдь не носит всепроникающего характера. Даже если это изменение кому-то не нравится, оно не касается всей остальной их работы.

А сейчас рассмотрим переход разработчика к использованию Scrum. Разработчику каждый раз придется выполнять сравнительно небольшие порции работы таким образом, чтобы получить какой-то завершенный фрагмент к окончанию очередного спринта. К каждому такому фрагменту разработчику наверняка придется писать автоматизированные тесты. Возможно, ему придется даже чередовать короткие периоды тестирования и кодирования, осуществляя на практике разработку программ на основе составления тестов. Возможно, ему придется делать все это, отказавшись от прослушивания любимых музыкальных записей на своем MP3-плеере, поскольку ему придется работать в паре с другим программистом (так называемое парное программирование). В этом случае речь уже идет о фундаментальных изменениях. Они не являются чем-то таким, чему можно посвящать не более двух-трех часов в день или в неделю (как, например, проверке кода). Такого рода фундаментальные изменения даются нелегко, поскольку разработчику приходится иметь с ними дело на протяжении всего своего рабочего дня. И сопротивление таким переменам будет большим, поскольку и влияние их большее.

Scrum носит всепроникающий характер и в другом отношении. Последствия применения гибкой методологии разработки распространяются далеко за пределы отдела разработки программного обеспечения. Внедрение отчета по операционной готовности почти наверняка не повлияет на финансовый отдел, отдел сбыта и другие отделы. Однако внедрение Scrum почти наверняка повлияет на эти отделы. Финансовому отделу придется привести во взаимное соответствие политику компании в отношении капитализации или расходов и способ выполнения Scrum-проектов. Отделу сбыта, возможно, придется пересмотреть способ доведения до клиентов своих обязательств относительно сроков исполнения и масштаба работ, а также способ структурирования договоров. Чем больше групп людей ощущает на себе последствия перехода к использованию Scrum, тем больше вероятность сопротивления такому переходу и непонимания людьми необходимости такого перехода. Это обуславливает возникновение дополнительных трудностей при переходе к использованию Scrum по сравнению с другими изменениями.

См. также Влияние Scrum на другие подразделения (например, на финансовый отдел, отдел операций, отдел кадров и другие) обсуждается в главе 20, “Кадры, техническое обеспечение и отдел управления проектами”.

Scrum — очень своеобразная методология

Изменения, обусловленные переходом к использованию Scrum, не только радикально меняют методы работы разработчиков, но и идут вразрез со всем тем, чему эти люди учились в прошлом. Многие тестеры, например, усвоили, что их работа сводится к проверке соответствия нового программного обеспечения определенным спецификациям. Программистов учили тому, что нужно глубоко проанализировать любую проблему и найти ее радикальное решение еще до того, как начнется процесс кодирования. В случае Scrum-проектов тестерам и программистам придется отказаться от этих привычек. Тестерам придется понять, что в ходе тестирования нужно проверить, насколько новое программное обеспечение соответствует потребностям пользователей. Программистам придется понять, что, прежде чем начнется процесс

кодирования, далеко не всегда нужно (а иногда даже нежелательно) выходить на окончательное проектное решение. Эбби Фихтнер (Abby Fichtner), которая делится своими соображениями в собственном блоге Hacker Chick, сказала мне, что, по ее мнению, такая ломка сложившихся стереотипов может оказаться для программистов очень мучительной.

Привыкнуть к концепции стихийного проектирования нелегко, поскольку у программиста может сложиться впечатление, что от него ожидают каких-то хакерских действий! А если вы гордитесь тем, что являетесь очень хорошим разработчиком и ваши разработки всегда отличались хорошей продуманностью, то у вас может сложиться впечатление, что кто-то пытается перевернуть все ваши представления с ног на голову и говорит: “Нет, все те качества, которые делали вас великим разработчиком, теперь делают вас плохим разработчиком”. Вам начинает казаться, что мир, в котором вы чувствовали себя так комфортно, рушится.

См. также Концепции стихийного проектирования и разработки программного обеспечения на основе составления тестов обсуждаются в главе 9, “Технические приемы”.

Поскольку переход к использованию Scrum предполагает радикальное изменение стиля и методов работы людей, которое идет вразрез с полученным ими образованием и их прошлым опытом, люди не очень-то охотно соглашаются на такие перемены. Рассмотрим, например, случай Терри, опытного иуважаемого в своей компании программиста. Терри прошел переподготовку на курсах разработки программного обеспечения на основе составления тестов и убедился в преимуществах такого метода. Вернувшись к работе, Терри, полный энтузиазма, рассчитывал, что больше не будет выполнять масштабные предварительные проекты, а станет исходить из того, что проект должен вырисовываться стихийно, сам собой, в процессе разработки программ на основе составления тестов. Однако все оказалось гораздо сложнее. Он отправил мне по электронной почте письмо с описанием своего печального опыта.

Заставить других программистов хотя бы попытаться перейти к разработке программ на основе составления тестов оказалось гораздо труднее, чем я ожидал. Я попытался представить такой переход как способ избежать привычных для нас длительных фаз предварительного проектирования, но мои попытки потерпели неудачу. Через пару месяцев мне все же удалось заставить других разработчиков начинать работу с написания тестов (и то лишь потому, что эта идея была хороша сама по себе). Они по-прежнему не могли отказаться от длительной фазы предварительного проектирования. Мне потребовался еще год для того, чтобы добиться существенного прогресса в деле сокращения продолжительности этой фазы (правда, мы могли бы сделать ее еще короче).

Перемены происходят быстрее, чем прежде

Еще в 1970 году Элвин Тоффлер (Alvin Toffler) придумал термин *шок от будущего* (future shock), означающий дезориентированность, которую ощущают люди, сталкиваясь со “слишком значительными переменами на слишком коротком отрезке

времени” (1970, 4). Человеческая — и, следовательно, организационная — способность к изменениям весьма ограничена (попросите людей изменить очень многое в своей жизни на очень коротком отрезке времени — и они “пойдут вразнос”: тяжелый стресс и дезориентированность, вызванные шоком от будущего, сделали свое дело).

Работники многих организаций годами испытывают сильный стресс в ожидании шока от будущего. От работников требуют все большего, в то же время сокращая их численность. Аутсорсинг и использование рассредоточенных команд приобретают все большую популярность среди руководства компаний. Этим переменам предшествовал перевод приложений в модель “клиент/сервер”, затем — в веб, а позже — в веб-службы. Добавьте к этому все ускоряющийся темп перемен в самой технологии — новые языки, новые инструменты, новые платформы — и шок от будущего вам гарантирован. Поэтому не должно вызывать удивления то обстоятельство, что переход к использованию Scrum вполне может оказаться именно той переменой, которая загоняет людей в шок от будущего. Всепроникающий характер внедрения Scrum и вызываемые им фундаментальные перемены в методах работы и взаимодействия людей связаны с повышенным риском запуска шока от будущего в действие.

Передовые методы таят в себе опасность

Характер большинства организационных перемен таков, что, как только кому-либо удается выявить правильный или наилучший способ что-то сделать, этот способ начинает рассматриваться как “передовой метод” и получает широкое распространение. Для некоторых типов работы аккумулирование и использование передовых методов является большим подспорьем в осуществлении перемен. Организация, которая продает какой-либо продукт новой категории клиентов, может, например, выявить передовые методы опровержения возражений со стороны потенциальных клиентов. Однако при осуществлении перехода к использованию Scrum аккумулирование передовых методов может таить в себе опасность.

Подобно сиренам, скрывающимся в скалах и завлекающим моряков своим пением, передовые методы соблазняют нас, предлагая расслабиться и прекратить дальнейшее совершенствование, которое является жизненно важным для Scrum. Тайichi Оно (Taiichi Ohno), создатель Toyota Production System, писал, что “существует такое понятие, как стандартная работа, но стандарты также подлежат постоянному совершенствованию. Если же вы представляете себе стандарт как лучшее, на что вы способны, то ваша песенка спета”. Продолжая свою мысль, Тайichi Оно утверждает, что если мы рассматриваем нечто как “наилучший способ, то у нас пропадает какая бы то ни было мотивация для *кайдзен* [непрерывное постепенное совершенствование]” (1982).

Несмотря на то что члены одной команды должны всегда делиться друг с другом эффективными методами работы, которые им удалось выработать или выявить, им не следует поддаваться соблазну рассматривать эти методы как наилучшие. Пример “наилучшего” метода, который в конечном счете дискредитировал себя, дает нам компания, которая решила, что все ежедневные совещания разработчиков должны начинаться не позднее 10:00. Мне это показалось ничем не оправданным диктатом. Я не знаю наверняка, какой цели служил этот диктат, но многие работники восприняли введение этого правила, как еще одно подтверждение того, что “Scrum сводится исключительно к микроуправлению”.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Поразмышляйте над своим текущим переходом к использованию Scrum. Вы лишь приступили к этому переходу, находитесь в середине пути или вам кажется, что вы приближаетесь к его завершению? На каком бы этапе вы ни находились, постарайтесь выявить основное препятствие, которое, как вам кажется, мешает вам выйти на следующий уровень успеха.

Почему переход к использованию Scrum стоит затрачиваемых на него усилий

Несмотря на перечисленные выше факторы, существенно затрудняющие переход компании к использованию Scrum, все те, кого этот переход в той или иной степени коснулся, не жалеют потраченных на это сил. Одна из причин такой удовлетворенности заключается в том, что при использовании гибкой методологии разработки, подобной Scrum, сокращается время вывода нового продукта на рынок. Это сокращение объясняется более высокой производительностью команд, использующих гибкую методологию разработки, что, в свою очередь, оборачивается повышением качества, характерным для проектов, выполняющихся посредством гибкой методологии разработки. Поскольку в этом случае время работников высвобождается для более качественного выполнения ими заданий и поскольку они видят, что их работа быстрее попадает в руки пользователей, степень удовлетворенности людей выполненной ими работой повышается. Когда люди удовлетворены своей работой, они проявляют большую заинтересованность в ней, что способствует росту их производительности. Таким образом, налицо эффективный цикл непрерывного совершенствования.

Оставшаяся часть данной главы посвящена углубленному анализу этих утверждений. Я постараюсь представить убедительные доказательства каждого из них. Некоторые из этих доказательств почерпнуты из моего собственного опыта, опыта моих клиентов и коллег или опыта, заимствованного из журналов или конференций. К тому же правильность этих утверждений доказывается данными из следующих источников.

- Строгое сравнение 26 проектов, выполненных посредством гибкой методологии разработки, с “эталонной” базой данных 7500 проектов, выполненных посредством традиционного метода разработки. Это исследование было проведено Майклом Махом (Michael Mah), управляющим партнером QSM Associates (QSMA), который на протяжении более чем 15 лет собирал данные о производительности и качестве и другие показатели, характеризующие выполнение проектов. Исследованные Махом проекты, выполненные посредством гибкой методологии разработки, были весьма разнообразны по своему масштабу: количество исполнителей этих проектов составляло от 60 до 1000 человек (Mah, 2008).
- Всевозможные научные и исследовательские статьи, в том числе обобщающее исследование доктора Дэвида Рико (David Rico), который составил обзор опубликованных работ по 51 проекту, выполненному посредством гибкой методологии разработки (2008).

- Онлайн-опрос более чем трех тысяч человек, выполненный поставщиком инструментов гибкой методологии разработки, VersionOne (2008), и еще один опрос 642 человек, выполненный Dr. Dobb's Journal (Ambler, 2008a), популярным журналом для разработчиков. Оба эти опроса были проведены в 2008 году. Отраслевые опросы, подобные этим, не могут, конечно, рассматриваться как истина в последней инстанции. Люди, проводящие подобные опросы, наверняка склонны к такой их интерпретации, которая свидетельствовала бы в пользу гибкой методологии разработки. Результаты этих опросов представлены здесь, поскольку они скорее репрезентативны, чем убедительны. На данные этих опросов мы будем ссылаться в последующих разделах как VersionOne и DDJ.
-

См. также Данные из этой главы представлены в виде презентаций Microsoft PowerPoint и Apple Keynote на веб-сайте www.succeedingwithagile.com.

В следующих разделах мы рассмотрим причины целесообразности перехода к использованию гибкой методологии разработки, такой как Scrum.

- Повышение производительности и снижение издержек
- Повышение заинтересованности работников в выполняемой ими работе и более высокая удовлетворенность ею
- Сокращение времени вывода новых продуктов на рынок
- Повышение качества
- Повышение удовлетворенности всех, кто так или иначе связан с разработкой новых продуктов
- Неудовлетворенность прежними методами работы

Повышение производительности и снижение издержек

К сожалению, в настоящее время отсутствует какой-либо общепринятый, универсальный показатель производительности. Мартин Фаулер (Martin Fowler) утверждает даже, что измерить производительность разработчиков вообще невозможно (Martin Fowler, 2003). И хотя я согласен с Фаулером, мне кажется, что можно измерять некоторые параметры, являющиеся заменителями производительности. В качестве такого “заменителя” одни группы используют количество строк кода. Другие используют в его качестве количество разработанных функциональных точек или, попросту говоря, количество реализованных возможностей, не обращая внимания на то обстоятельство, что они сильно разнятся между собой по сложности (и размеру). Возникают ли какие-то проблемы с такими заменителями? Несомненно. Но мне кажется, что полезность таких показателей производительности представляется вполне оправданной, если предположить, что разработчики не занимаются подтасовкой данных, манипулируя количеством строк кода или функциональных точек путем дублирования кода, игнорирования возможности многократного использования одних и тех же фрагментов кода в разных местах программы и т.п. Во многих случаях, особенно когда речь идет о крупных массивах данных, как в случае исследования QSMA, такое предположение мне кажется вполне допустимым.

QSMA вычисляет индекс производительности для проектов, имеющихся в ее базе данных. Этот индекс учитывает затраченные усилия, график выполнения работ, техническую сложность и многое другое и представляет собой попытку сделать более обоснованным сравнение производительности разных команд. Сравнивая проекты, выполнявшиеся посредством гибкой методологии разработки, и проекты, выполнявшиеся традиционным методом, Mah приходит к выводу, что производительность проектов, выполнявшихся посредством гибкой методологии разработки, на 16% превышает производительность проектов, выполнявшихся традиционным методом. Такая разница, несомненно, является статистически значимой. На рис. 1.1 представлены в виде точек проекты, выполнявшиеся посредством гибкой методологии разработки, в сравнении со средней производительностью и одним среднеквадратическим отклонением от нее в базе данных QSMA. Как видно из этого рисунка, большинство точек располагается над линией отраслевого среднего, причем небольшое количество проектов отстоит от линии отраслевого среднего на расстояние, превышающее одно среднеквадратическое отклонение.

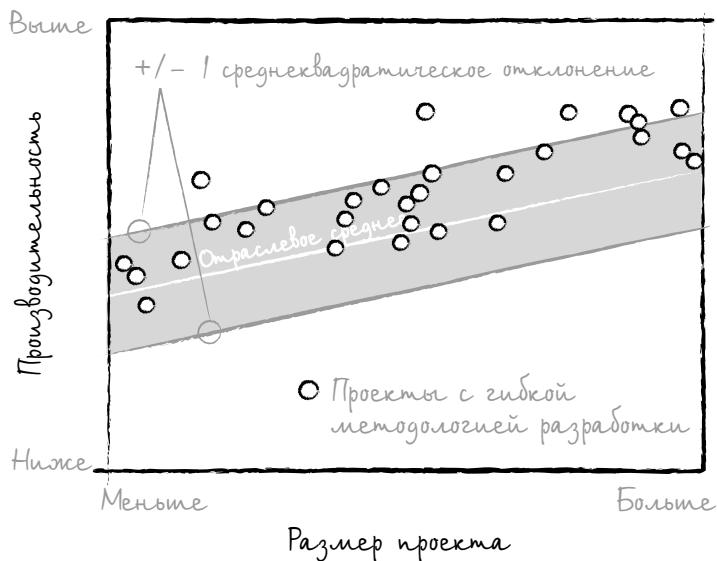


Рис. 1.1. Группы, выполняющие проекты с применением гибкой методологии разработки, демонстрируют гораздо более высокую производительность, чем в среднем по отрасли. Источник: Mah 2008

Результаты QSMA подтверждаются опросами VersionOne и DDJ. Восемьдесят два процента участников опроса DDJ выразили мнение, что в случае использования гибкой методологии разработки типа Scrum производительность была несколько выше или намного выше, чем прежде. Лишь 5% участников этого опроса выразили мнение, что производительность в этом случае оказалась несколько ниже или намного ниже. Семьдесят три процента респондентов опроса VersionOne полагали, что использование гибкой методологии разработки существенно улучшило (23%) или просто улучшило (50%) производительность.

Здравый смысл подсказывает, что повышение производительности ведет к снижению издержек. Опросы VersionOne и DDJ подтверждают эту мысль, что следует из табл. 1.1.²

Таблица 1.1. Значительное число респондентов, участвовавших в опросе, указывают, что применение гибкой методологии разработки позволило сократить затраты на разработку

Затраты на разработку	DDJ, %	VersionOne, %
Сократились	32	30
Значительно сократились	5	8

Данные обзора, составленного Дэвидом Рико и касающегося групп, которые применяли в практике проектирования гибкую методологию разработки, были опубликованы в 2008 году. Они представлены в табл. 1.2. Дэвид Рико пришел к выводу, что среднее повышение производительности составило 88%, а средняя экономия издержек — 26%. Эти данные со всей очевидностью свидетельствуют о том, что группы, которые применяли в практике проектирования гибкую методологию разработки, добивались роста производительности, который приводил к сокращению издержек по выполнявшимся ими проектам.

Какими бы обнадеживающими ни казались эти данные, они не дают полного представления о реальном положении дел. Значительным преимуществом использования гибкой методологии разработки — преимуществом, не получившим отражения здесь — является то, что группы, применяющие в своей практике проектирования гибкую методологию разработки, в меньшей степени склонны реализовывать функциональность, оказывающуюся в конечном счете бесполезной. Типичная критика в адрес процесса последовательной разработки заключается в том, что ко времени поставки готового программного обеспечения его пользователям, эти пользователи уже не нуждаются в функциональности, реализованной разработчиками. В результате регулярной обратной связи, периодического проведения спринтов и возможности изменения приоритетов каждого спрингта Scrum-группа с большей вероятностью будет работать над реализацией именно той функциональности, которая действительно понадобится пользователям. Если бы мы включили это преимущество в наше измерение производительности, то мы увидели бы еще более впечатляющие результаты.

Таблица 1.2. Влияние гибкой методологии разработки на производительность и издержки. Источник: Rico 2008

Категория	Минимальное указанное улучшение, %	Среднее улучшение, %	Максимальное указанное улучшение, %
Производительность	14	88	384
Затраты	10	26	70

² Авторы опроса VersionOne просили своих респондентов ответить с помощью шкалы, которая включала следующие градации: значительно улучшилась, улучшилась, ничего не изменилось, ухудшилась, значительно ухудшилась. Авторы опроса DDJ воспользовались шкалой со следующими градациями: гораздо выше, несколько выше, ничего не изменилось, несколько ниже, гораздо ниже. С целью повышения удобочитаемости во всех таблицах этой главы используются обозначения, применявшиеся в опросе VersionOne.

Повышение заинтересованности работников в выполняемой ими работе и более высокая удовлетворенность ею

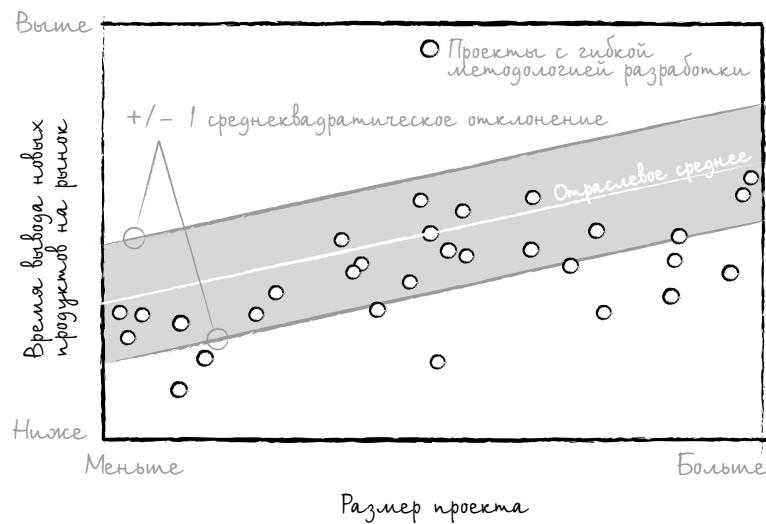
Одним из факторов, способствующих повышению производительности и снижению издержек при выполнении проектов посредством гибкой методологии разработки, возможно, является то, что исполнители испытывают большее удовлетворение от своей работы. Спустя пятнадцать месяцев после внедрения у себя Scrum компания Salesforce.com опросила своих сотрудников и пришла к выводу, что 86% из них получают удовольствие, работая в этой компании. До внедрения Scrum лишь 40% сотрудников Salesforce.com получали от этого удовольствие. Кроме того, 92% сказали, что они порекомендовали бы другим компаниям внедрить у себя Scrum. Подобные результаты являются типичными: многие из моих клиентов проводили подобные опросы среди своих сотрудников и всегда получали примерно такие же результаты. В ходе своего общеотраслевого опроса VersionOne пришла к выводу, что 74% опрошенных указывали на улучшение (44%) или значительное улучшение (30%) моральной атмосферы в их коллективах.

Одной из возможных причин повышения удовлетворенности работников своей работой являются стабильные темпы выполнения работ, обеспечиваемые гибкой методологией разработки. Крис Манн (Chris Mann) и Франк Маурер (Frank Mauger) из Университета Калгари исследовали объем сверхурочного времени, отработанного одной из команд за год до внедрения гибкой методологии разработки и на протяжении первого года после ее внедрения (2005). Исследователи пришли к выводу, что до внедрения гибкой методологии разработки объем сверхурочных работ в этой команде составлял в среднем 19%. После внедрения объем сверхурочных работ сократился в среднем до 7%. Кроме того, несмотря на то что необходимость в выполнении сверхурочных работ время от времени возникала даже после внедрения гибкой методологии разработки, удавалось снизить изменчивость объемов сверхурочных работ, измеряемую величинами среднеквадратического отклонения до и после внедрения гибкой методологии разработки. Йоханнес Бродуолл (Johannes Brodwall), архитектор гибкого программного обеспечения, говорит: “Необходимость в выполнении сверхурочных работ существенно снизилась после того, как мы перешли к использованию гибкой методологии разработки. Особенно заметили эту перемену тестеры. Раньше они постоянно страдали от перегрузок на работе”.

Снижение объемов сверхурочных работ в результате внедрения гибкой методологии разработки является, по-видимому, лишь одним фактором, способствующим повышению степени удовлетворенности людей своей работой. Другими преимуществами являются получение большего контроля над вашей повседневной работой, возможность быстрее увидеть использование плодов вашего труда, работа в более тесном контакте с членами своей команды, создание продуктов, в большей степени удовлетворяющих потребностям и ожиданиям ваших клиентов и пользователей, и т.п. Работники, в большей степени удовлетворенные своей работой и своими работодателями, как правило, больше заинтересованы в выполняемой ими работе. А большая заинтересованность работников в выполняемой ими работе оборачивается для соответствующей организации многочисленными дополнительными преимуществами.

Сокращение времени вывода новых продуктов на рынок

Команды, применяющие в своей работе гибкую методологию разработки, как правило, выпускают новые продукты быстрее, чем команды, пользующиеся традиционным методом разработки. Согласно опросу, проведенному VersionOne, 64% участников опроса указывают на сокращение (41%) или значительное сокращение (23%) времени вывода новых продуктов на рынок. Исследование QSMA, в ходе которого проводилось сравнение 26 проектов, выполненных посредством гибкой методологии разработки, с “эталонной” базой данных 7500 проектов, выполненных посредством традиционного метода разработки, привело, в частности, к следующему результату: в случае проектов, выполнявшихся с применением гибкой методологии разработки, время вывода новых продуктов на рынок оказалось на 37% меньшим, чем в случае проектов, выполнявшихся с применением традиционного метода разработки, что отражено на рис. 1.2.



*Рис. 1.2. Проекты, выполняемые посредством гибкой методологии разработки, обеспечивают меньшее время вывода новых продуктов на рынок, чем в среднем по отрасли.
Источник: Mah 2008*

Такое сокращение времени вывода новых продуктов на рынок в случае проектов, выполняющихся с применением гибкой методологии разработки, обусловлено двумя причинами. Во-первых, более высокая производительность команд, применяющих в своей работе гибкую методологию разработки, позволяет им быстрее реализовывать функциональность, необходимую клиентам. Во-вторых, команды, применяющие в своей работе гибкую методологию разработки, с большей вероятностью производят свою продукцию инкрементным способом (т.е. способом постепенных приращений). Когда заказчики видят, что исполнители могут создавать важные возможности к завершению каждого спринта, они зачастую приходят к выводу, что им нет нужды ждать, пока будет реализована вся функциональность.

Компания Salesforce.com почувствовала это преимущество сразу же после быстрого внедрения у себя Scrum (Greene and Fry, 2008). На рис. 1.3 представлено совокупное количество функциональных возможностей, созданных для клиентов в 2006 году (до внедрения Scrum) и в 2007 году (после начала использования Scrum примерно в начале года). На этом рисунке представлено простое соотношение: примерное количество функциональных возможностей, созданных для клиентов, и время их создания.

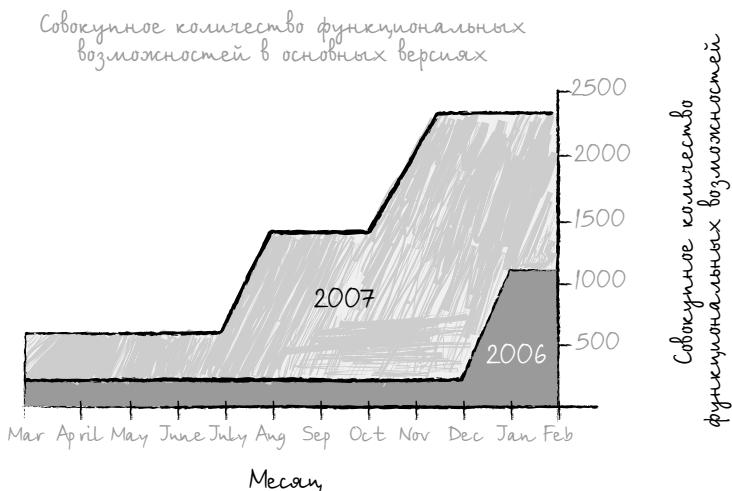


Рис. 1.3. Совокупное количество функциональных возможностей, созданных компанией Salesforce.com в 2006 году (до внедрения Scrum) и в 2007 году (после начала использования Scrum)

Повышение качества

Если спросить у разработчиков, применяющих Scrum, что обеспечивает большую их производительность по сравнению со временами, когда они пользовались традиционной методологией разработки, большинство ответит вам, что по меньшей мере часть их успеха обусловлена тем, что они стабильно выполняют высококачественную работу. Без тормозящих работу группы ошибок они могут быстро и уверенно продвигаться вперед. Качество улучшается за счет того, что стабильный темп работы команды предотвращает небрежность при написании кодов. Оно повышается и за счет таких прогрессивных методов разработки, как парное программирование, рефакторинг и безусловный упор на раннее и автоматизированное тестирование.

Исследование, проведенное Дэвидом Рико, подтверждает вывод о том, что группы, применяющие в своей работе гибкую методологию разработки, создают более высококачественные продукты. В предпринятом Дэвидом Рико обзоре опубликованных работ по 51 проекту, выполненному посредством гибкой методологии разработки, было установлено, что минимальное улучшение качества составило 10%, а среднее улучшение качества составило 63%. Результаты исследования, проведенного Рико, подтверждают мой собственный опыт работы с клиентами, который позволял мне оценивать качество выполненных работ и публиковать результаты этих оценок.

Например, ePlanServices составляет пенсионные планы для компаний среднего размера. Эта услуга предоставляется главным образом посредством одного мощного веб-приложения. В течение первых девяти месяцев после начала перехода к Scrum частота появления ошибок на каждую тысячу строк кода сократилась у них на 70%.

Результаты опроса, проведенного VersionOne, также подтверждают вывод о повышении качества при использовании гибкой методологии разработки, такой как Scrum. Шестьдесят восемь процентов участников этого опроса ответили, что использование гибкой методологии разработки улучшило (44%) или значительно улучшило (24%) качество программного обеспечения. Кроме того, у 84% респондентов сложилось впечатление, что использование гибкой методологии разработки позволило сократить количество дефектов не менее чем на 25%. Опрос, проведенный DDJ, дал аналогичные результаты: 48% респондентов ответили, что качество “несколько повысилось”, а 29% респондентов ответили, что качество “повысилось существенно”.

Повышение удовлетворенности всех, кто так или иначе связан с разработкой новых продуктов

Учитывая все перечисленные выше преимущества использования гибкой методологии разработки, неудивительно, что они ведут к повышению удовлетворенности всех сторон, заинтересованных в разработке новых продуктов. Результаты опроса, проведенного DDJ, показали, что 78% участников опроса полагают, что использование гибкой методологии разработки привело к несколько более высокой (47%) или гораздо более высокой (31%) удовлетворенности всех сторон, заинтересованных в разработке новых продуктов.

Одна из причин более высокой удовлетворенности гибкой методологией разработки всех сторон, заинтересованных в разработке новых продуктов, заключается в том, что эта методология позволяет легко изменять приоритеты, чем приходится регулярно заниматься современным динамичным и конкурентоспособным организациям. Со-гласно данным опроса, проведенного VersionOne, 92% участников высказали мнение, что применение гибкой методологии разработки повышало способность управления процессом изменения приоритетов. Помимо повышения способности маневрирования приоритетами, лица, заинтересованные в реализации проектов, выполняемых с применением гибкой методологии разработки, получают представление о последствиях соответствующих изменений. В частности, на это указывает один из пайщиков PetroSleuth, небольшой проектной компании, принадлежащей к нефтегазодобывающей промышленности.

Scrum-процесс повысил степень нашей вовлеченности в ежедневные обзоры и обсуждения. Это, в свою очередь, повысило нашу информированность о предстоящих изменениях и степень нашей ответственности за возможные последствия таких изменений (Mann and Maurer, 2005, 77).

Опрос, проведенный VersionOne, позволил проанализировать дополнительные факторы, повышающие удовлетворенность сторон, заинтересованных в разработке новых продуктов. Из табл. 1.3 видно, что большое число участников опроса указало на применение гибкой методологии разработки как на причину улучшения координации целей групп технологов и групп экономистов, снижения риска проекта, улучшения способности управлять изменением приоритетов и улучшения обозримости проекта.

Стив Фишер (Steve Fisher), старший вице-президент Salesforce.com и участник многих команд этой компании, выполняющих проекты с применением гибкой методологии разработки, утверждает, что внедрение Scrum “привело к улучшению обозримости проектов, их прозрачности, а также к невероятному росту производительности — одним словом, к полному и безоговорочному успеху” (Greene, 2008).

Таблица 1.3. Некоторые из факторов, обусловливающих повышение удовлетворенности в случае применения гибкой методологии разработки

	Улучшилось, %	Значительно улучшилось, %
Повышение способности управлять изменением приоритетов	41	51
Улучшение обозримости проектов	42	41
Улучшение координации целей IT и экономических целей	39	27
Снижение риска проекта	48	17

Неудовлетворенность прежними методами работы

Решающей причиной, которая должна заставить вас перейти к использованию Scrum, может быть ваша неудовлетворенность текущим процессом разработки программного обеспечения. Когда процесс, который устраивал вас в прошлом, утрачивает свою эффективность, вы, как правило, стремитесь каким-то образом его усовершенствовать. Именно так поступили в компании Yahoo!, где вице-президент по производству Пит Димер (Pete Deemer) был одним из первых, кто признал необходимость перемен.

Поначалу Yahoo! стала использовать Scrum просто от отчаяния — метод водопада перестал работать. В течение года компания пыталась “усовершенствовать” метод водопада путем более тщательного планирования и анализа, более подробного документирования, более широкого согласования и т.п. Однако все эти “усовершенствования” не улучшили, а напротив, лишь усугубляли ситуацию. Команды, которые перешли к использованию Scrum, практически сразу же увидели преимущества этой новой методологии по сравнению с методом водопада.

Аналогичную историю рассказывает Клинтон Кейт (Clinton Keith), бывший вице-президент по технологиям в компании High Moon Studios, специализирующейся на разработке консольных видеоигр.

Нам, удачливым руководителям проектов в хорошо финансирующейся начинающей компании, казалось, что к нашим новым амбициозным проектам можно было бы применить “более мощный водопад”. Однако на этом пути мы добились лишь обратного эффекта: мы начали терять контроль над своими проектами. Наши предположения оказались неправильными. Это заставило нас переосмыслить используемые нами методы управления проектами.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Выявите преимущества, полученные вами к настоящему времени в результате использования Scrum.
- Если вы еще не собрали данных о качестве, атмосфере в коллективе, удовлетворенности тех, кто заинтересован в выполнении вашего проекта, и т.п., выберите несколько ключевых факторов и проведите измерения для получения некой "точки отсчета", которую вы будете впоследствии использовать для сравнения.
- Если вы уже собрали данные, которые будут служить вам "точкой отсчета", и пользуетесь методологией Scrum по меньшей мере три-шесть месяцев, выполните повторные измерения и оцените достигнутый вами прогресс. Постройте собственные диаграммы "Почему Scrum заслуживает внедрения", с которыми вы сможете ознакомить другие команды, когда они начнут переходить к использованию Scrum, или команды, внедрившие у себя Scrum, но испытывающие проблемы с привыканием к этой необычной методологии.

Что дальше

Переход к гибкой методологии разработки, как правило, дается нелегко. Во всяком случае, он происходит гораздо труднее, чем большинство других организационных перемен, с которыми мне приходилось иметь дело самому или которые я наблюдал со стороны. Я начал данную главу с перечисления ряда причин, обуславливающих эти трудности, среди которых — необходимость осуществлять изменения одновременно по принципу "сверху вниз" и по принципу "снизу вверх", невозможность точно знать, каким будет конечное состояние, радикальные и всепроникающие изменения, вызываемые переходом к использованию Scrum, сложность добавления новых изменений поверх всего, что уже происходит, а также необходимость избегать превращения Scrum в перечень "наилучших методов".

Поскольку вы еще не перестали читать эту книгу, я смею предположить, что этот перечень проблем не отбил у вас желания внедрить у себя Scrum. Это обнадеживает, поскольку организация, которая преодолеет указанные трудности, получит ряд колossalных преимуществ, в том числе более производительные команды, сокращение издержек, большую удовлетворенность людей своей работой, сокращение времени вывода новых продуктов на рынок, улучшение качества и повышение степени удовлетворенности всех сторон, заинтересованных в успешной реализации проекта.

В следующей главе мы рассмотрим подробнее, что требуется для того, чтобы перевести вас, вашу команду и вашу организацию из стадии, на которой вы уже осознали необходимость перемен и поняли, что их оптимальным вариантом является внедрение Scrum, на стадию, на которой вы начнете демонстрировать реальный прогресс и постоянные улучшения.

Дополнительная литература

Ambler, Scott. 2008. Agile adoption rate survey, February. <http://www.ambysoft.com/surveys/agileFebruary2008.html>.

В этой статье излагаются результаты опроса, проведенного в феврале 2008 года. Эти результаты могут служить хорошим дополнением к результатам, представленным выше.

Greene, Steve, and Chris, Fry. 2008. Year of living dangerously: How Salesforce.com delivered extraordinary results through a “big bang” enterprise agile revolution. Session presented at Scrum Gathering, Stockholm. <http://www.slideshare.net/sgreene/scrum-gathering-2008-stockholm-salesforcecom-presentation>.

Грини и Фрай управляли процессом внедрения Scrum в Salesforce.com. В указанном материале они рассказывают о том, как происходило внедрение, чему они научились в процессе внедрения и что они сделали бы по-другому, если бы им представилась такая возможность.

Mah, Michael. 2008. How agile projects measure up, and what this means to you. *Cutter Consortium Agile Product & Project Management Executive Report* 9(9).

В этой статье Mah сравнивает 26 проектов, выполненных посредством гибкой методологии разработки, с “эталонной” базой данных, включающей более 7500 проектов, выполненных посредством традиционного метода разработки.

Rico, David F. 2008. What is the ROI of agile vs. traditional methods? An analysis of extreme programming, test-driven development, pair programming, and Scrum (using real options). С персонального веб-сайта Дэвида Рико можно загрузить электронную таблицу <http://davidfrico.com/agile-benefits.xls>.

Обширный обзор имеющейся в наличии литературы по проектам, выполненным посредством гибкой методологии разработки, в котором представлены ключевые улучшения, касающиеся производительности, затрат, качества, графика выполнения работ, удовлетворенности клиентов и прибыли на инвестированный капитал и выраженные в процентах.

VersionOne. 2008. The state of agile development: Third annual survey. Рассыпается в виде файла, загружаемого в формате PDF из библиотеки Library of White Papers на веб-сайте VersionOne по адресу http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf.

Каждый год компания VersionOne, занимающаяся разработкой инструментов гибкой методологии разработки, проводит крупнейший международный опрос, касающийся текущего положения с внедрением гибкой методологии разработки. Этот опрос позволяет получить самое широкое представление об использовании гибкой методологии разработки.

Глава 2

ADAPТация к Scrum

Лори Шубринг (Lori Schubring) была одной из первых, кто осознал необходимость перемен. Лори, менеджер по разработке приложений в крупной промышленной компании, поняла, что используемый ими процесс разработки стал “до такой степени формализованным, что мы начали терять способность к поддержанию гибкости. Дело дошло до того, что нам уже не удавалось достаточно быстро пересматривать требования к проектам” (2006, 27). Осознав необходимость перемен, Лори решила принять участие в бесплатном семинаре продолжительностью в половину рабочего дня. Этот семинар был посвящен ознакомлению со Scrum. Лори показалось, что методология Scrum позволяет эффективнее разрабатывать программное обеспечение и, следовательно, способна помочь ее организации. В результате у Лори возникло желание внедрить эту методологию в своей организации. Чтобы уяснить, как реализовать на практике процесс внедрения, Лори решила прослушать курс ScrumMaster, поучаствовать в конференции, посвященной гибкой методологии разработки, и посетить какую-либо из компаний, которые уже внедрили у себя Scrum. Затем Лори рассказала о Scrum своему начальнику и подчиненным и убедила их в преимуществах этой методологии. Наконец Лори перенесла опыт применения Scrum своей команды на всю свою организацию, чтобы деятельность разных подразделений способствовала, а не препятствовала использованию этой методологии.

История Лори заключает в себе пять типичных действий, необходимых для успешного внедрения и длительного применения Scrum.

- **Осознание (Awareness).** Осознание того, что текущий процесс не обеспечивает приемлемых результатов.
- **Желание (Desire).** Желание применять Scrum как средство решения текущих проблем.
- **Способность (Ability).** Способность достичь успеха с помощью Scrum.

- **Продвижение (Promotion).** Пропаганда Scrum путем распространения опыта успешного применения этой методологии, что позволяет не только ознакомить с этими успехами других, но и самим не забыть об этом.
- **Распространение (Transfer).** Распространение Scrum, а также последствий, которые влечет за собой применение Scrum, на другие подразделения компании.

Эти пять типичных действий обозначаются удобной для запоминания аббревиатурой ADAPT (Awareness, Desire, Ability, Promotion, Transfer).¹ Перечисленные действия отражены также для большей наглядности на рис. 2.1. На этом рисунке осознание, желание и способность перекрываются, в то время как продвижение и распространение повторяются и происходят на протяжении всего процесса перехода. После того как переход к Scrum завершится, этот цикл будет продолжаться по мере постоянного совершенствования.

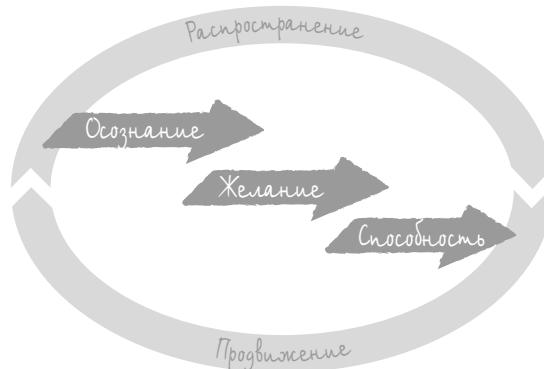


Рис. 2.1. Пять действий, связанных с переходом к использованию Scrum

Организацию, которая успешно применяет Scrum, можно представить как участвующую в этих действиях на нескольких уровнях.

- **Организационно.** В этих действиях принимает участие вся организация в целом. Какой бы ни была осознанность отдельно взятого человека или группы людей, должна накопиться некая критическая масса людей с одинаковой осознанностью. Лишь после этого организация сможет коллективно двигаться вперед. Представляя себе модель ADAPT на этом уровне, можно говорить о компании с *организационным* желанием применять Scrum. Или можно сказать, что организации в настоящее время не хватает способности применять Scrum.
- **На индивидуальном уровне.** Поскольку организации состоят из индивидуумов, важно понимать, что эти индивидуумы преодолевают фазу перехода

¹ Указанные пять действий базируются на ADKAR (Hiatt, 2006), общей модели изменений, которая включает следующие этапы: Awareness (осознание), Desire (желание), Knowledge (знание), Ability (способность) и Reinforcement (закрепление). Вообще говоря, разделение Knowledge (знание) и Ability (способность) кажется мне неоправданным. В такой сфере, как разработка программного обеспечения, знание без способности лишено какого бы то ни было смысла. Кроме того, этап Reinforcement (закрепление) в модели ADKAR заменен в ADAPT двумя этапами, Promotion (пропаганда) и Transfer (распространение), что подчеркивает важность этих действий для успешного перехода к использованию Scrum.

организации в целом (к использованию Scrum) с разными скоростями. Например, возможно, вы уже готовы к использованию Scrum, т.е. уже получили некоторые навыки и новые представления о том, как следует разрабатывать программное обеспечение. Однако в то же время до некоторых ваших коллег лишь начинает доходить мысль о том, что прежний подход уже не годится.

- **На уровне отдельных команд.** Переход отдельных людей к использованию Scrum может либо ускоряться, либо тормозиться коллективами, в которых они работают. Команды, как правило, продвигаются через цикл ADAPT более или менее одновременно. Результаты исследований показывают, что люди обычно склонны набирать лишний вес, если их друзья, мягко говоря, не страдают дистрофией (Thaler and Sunstein, 2009). Точно так люди обычно демонстрируют желание использовать Scrum, если команда, в которой они работают, также охотно использует Scrum.
- **На уровне отдельных приемов работы.** Модель ADAPT может применяться к каждому новому техническому приему, который приобретается в рамках внедрения Scrum. Рассмотрим, например, рост доверия к автоматизированному тестированию исходного кода, которое является типичным для команд, использующих Scrum. Команда и ее члены должны сначала осознать, что используемый ими подход к тестированию утратил свою эффективность. Затем у них должно появиться желание автоматизировать как можно большее число тестов — причем сделать это на как можно более ранней стадии данного процесса. Для этого комуто из членов команды, возможно, придется освоить какие-то новые технические приемы. Пропаганда успехов команды в деле автоматизированного тестирования подвигнет другие команды разработчиков к использованию их опыта. Наконец, распространение на другие команды опыта широкого использования автоматизированного тестирования, а также последствий, которые влечет за собой широкое применение автоматизированного тестирования, приведет к тому, что силы, внешние по отношению к данной команде, не станут препятствовать дальнейшему использованию данной командой этого нового технического приема.

Одним из ваших первых действий — независимо от того, используете ли вы Scrum или только приступаете к внедрению этой методологии — должно быть определение, в каких именно точках цикла ADAPT пребывают в данный момент отдельные ваши работники, команды и организация в целом. Может оказаться, что вы приобретаете способность разрабатывать программное обеспечение на основе составления тестов в команде, *продвигающей* свои успехи внутри подразделения, которое *желает* внедрить у себя Scrum. Однако организация в целом может *осознавать* лишь общую необходимость перемен. В этой главе мы обсудим не только пять действий ADAPT, но и инструменты, которые понадобятся для выработки осознания, возникновения желания и способности, продвижения и распространения по всем уровням организации.

Осознание

Перемены начинаются с осознания неприемлемости дальнейшего сохранения статус-кво. Однако иногда осознать неэффективность метода, которым вы привыкли пользоваться, очень нелегко. Самым ярким примером этого является случай, к которому я имею непосредственное отношение. В середине 1990-х годов я занимал пост

директора по перспективным разработкам в компании, которая разрабатывала программное обеспечение для здравоохранения. Основатель компании пришел к выводу, что единственный продукт компании — продукт, который в свое время принес ей огромный успех — будет продаваться от силы еще год. Обусловлено это было, по его мнению, фундаментальными переменами, происходившими в то время в системе здравоохранения Соединенных Штатов Америки. Нашей компании необходимо было разработать новый продукт, который должен был учесть эти фундаментальные перемены и принести новый успех. На общем собрании компании основатель представил слайд со схемой, показанной на рис. 2.2.

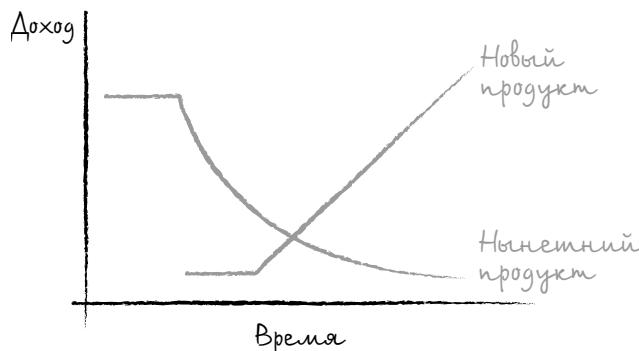


Рис. 2.2. “Долина смерти” демонстрирует снижение доходов от текущего продукта, наступающее задолго до начала выпуска нового продукта

В то время как большинство сотрудников компании намеревалось и в дальнейшем пожинать богатые плоды, которые принес первый продукт, основатель компании осознал, что мы вступаем в “Долину смерти” (по его собственному выражению). Когда компания оказывается в “Долине смерти”, доходы от текущего продукта быстро снижаются, причем это снижение намного опережает увеличение доходов от нового продукта, к разработке которого в то время мы даже не приступили.

Лишь немногие из нас способны проявить такую дальновидность, как основатель этой компании. Между моментом, когда впервые возникает необходимость перемен, и моментом, когда мы осознаем эту необходимость, почти всегда существует определенный зазор. Он может оказаться особенно большим, если компания чувствует себя достаточно уверенно. Ниже перечислен ряд других типичных причин, замедляющих осознание людьми необходимости перемен.

- **Неспособность увидеть картину в целом.** Необходимость внедрения Scrum может быть обусловлена сочетанием факторов, которые далеко не каждый способен увидеть. Потребность в изменениях может быть очевидна лишь для тех, кто обратил внимание на снижение продаж новым клиентам, до кого дошли слухи о появлении серьезного конкурента в нише, занимаемой данной компанией, и кто предвидит необходимость наращивания производства без увеличения числа работников.
- **Отказ видеть то, что происходит буквально у вас под носом.** Даже когда потребность в изменениях очевидна, мы подчас закрываем на это глаза. Мы

предпочитаем думать, что речь идет лишь о временных проблемах, и нередко опасаемся того, что могут повлечь за собой перемены. Точка зрения “давайте не будем трогать это до тех пор, пока оно окончательно не сломается” имеет очень мало общего с точкой зрения “давайте совершенствовать это до тех пор, пока не добьемся идеала (а идеала мы не добьемся никогда)”, характерной для гибкой методологии разработки.

- **Путаница в таких понятиях, как “движение” и “прогресс”.** Каждый день мы наблюдаем бурную деятельность. Проводятся всевозможные совещания, циркулируют отчеты о ходе выполнения работ, составляются какие-то документы, проверяются коды. Короче говоря, наблюдается путаница в таких понятиях, как “движение” и “прогресс”. Когда происходит много событий, иногда трудно допустить, что вся эта кипучая деятельность ни на йоту не приближает нас к желаемому результату.
- **Склонность прислушиваться к собственной пропаганде.** Информационный бюллетень компании полон восторженных статей, предсказывающих ее безоблачное будущее. В вестибюле штаб-квартиры компании выставлены награды за прошлые достижения. О чем говорят в коридорах работники компании? О том, что дела идут как нельзя лучше! Тем не менее клиенты спрашивают: “Каковы ваши *нынешние* достижения? Что нового вы сделали для меня *сегодня*?“ Предаваясь самовосхвалениям, мы неминуемо скатываемся в состояние самоуспокоенности и самодовольства. Подлинные успехи, разумеется, нужно праздновать, но никогда нельзя забывать о нелегком труде, без которого не бывает успеха.

Инструменты, способствующие осознанию

Отдельные члены команды осознают необходимость перемен не одновременно. Те, кто пришли к этому пониманию быстрее остальных, получают возможность помочь другим осознать необходимость перемен. В этом разделе мы рассмотрим инструменты, которыми можно воспользоваться, чтобы помочь людям осознать необходимость перемен.

Доведите до сведения других факт существования проблемы. Компания BioWare является одним из ведущих разработчиков видеоигр в мире. В этой компании трудятся свыше 400 человек, а ее продуктами являются такие известные сюжетные игры, как *Mass Effect*, *Jade Empire*, *Dragon Age*, *Knights of the Old Republic*, *Neverwinter Nights* и *Baldur's Gate*. Несмотря на то что продукты BioWare пользовались успехом, проекты, связанные с их разработкой, оказались не очень-то эффективными. Эти проекты страдали обычными недугами (проблемы со значительными объемами сверхурочных работ и взаимодействием разработчиков, несоответствие разработок ожиданиям пользователей и т.п.).

Вследствие того что BioWare создала немало продуктов, пользующихся успехом у потребителей, почти никому из лиц, связанных с деятельностью этой компании, не приходило в голову, что сами по себе эти проекты могут быть более успешными. К счастью, поиски более эффективного способа разработки видеоигр, предпринятые продюсером Трентом Остером (Trent Oster), вывели его на Scrum. Тренту Остеру удалось принять на работу нескольких руководителей проектов, уже имевших к тому времени опыт использования Scrum. Но это ядро первопроходцев Scrum не могло добиться существенного прогресса до тех пор, пока не начало помогать другим осознать

необходимость перемен. Это удалось им лишь после того, как они сформулировали цель, которая должна быть общей для всех проектов, выполняемых BioWare.

Разрабатывать высококачественные видеоигры с минимальными затратами так же увлекательно, как и играть в них.

Эта цель оказалась замечательной по двум причинам. Во-первых, ее сложно обосновать. Трудно вообразить разработчика, который утверждал бы, что работать долгими ночами над проектами так же увлекательно, как и играть в игры, появляющиеся в результате выполнения этих проектов. Во-вторых, в этой цели не предлагается какое-то решение. Представим, что было бы, если бы первопроходцы Scrum в BioWare выбрали, например, такую цель: “Разрабатывать высококачественные видеоигры с применением гибкой методологии разработки”. Вряд ли это убедило бы кого-нибудь в необходимости перемен (конечно, за исключением тех, кто и без того уже был на стороне Scrum). Уильям Бриджес (William Bridges), автор книги *Managing Transitions*, подчеркивает важность формулирования проблемы и доведения факта ее существования до всех заинтересованных лиц, а не предложения конкретного способа ее решения (2003, 16).

Используйте показатели. Являясь одним из элементов стратегии коммуникации в целом, показатели заостряют внимание всех заинтересованных сторон на основных причинах, обуславливающих необходимость перемен. Я могу привести примеры компаний, которые использовали такие простые показатели, как текучесть кадров, результаты опросов сотрудников, касающиеся их удовлетворенности своей работой, величина дохода в расчете на одного работника и другие, свидетельствующие о том, что перемены действительно необходимы.

Знакомьте своих сотрудников с новыми людьми и их опытом. Предлагайте своим сотрудникам посещать конференции или учебные курсы, где они смогут узнать о новых методах и технических приемах. Отправляйте их на отраслевые выставки, где они смогут ознакомиться с продуктами, выпускаемыми вашими конкурентами. Организуйте встречи членов своих команд с клиентами, в ходе которых они смогут узнать из первых рук, какая функциональность необходима пользователям ваших продуктов и в какие временные рамки должны уложиться разработчики этих продуктов. Хорошей долгосрочной стратегией знакомства ваших сотрудников с новыми людьми, их опытом и новыми идеями является обеспечение разнообразия сотрудников, принимаемых на работу. Иными словами, желательно принимать на работу людей, обладающих разным опытом, поскольку это не только способствует привнесению в организацию новых идей, но и помогает ей лучше их воспринимать в будущем.

Выполните пилотный проект. Успешный пилотный проект продемонстрирует людям реальную возможность улучшить ситуацию с проектированием. Когда те, кто еще не осознали необходимости перемен, увидят успешный результат проекта, выполненного новым способом, им придется либо проигнорировать этот результат, либо удостовериться в целесообразности предлагаемых перемен.

См. также В главе 3, “Модели внедрения Scrum”, сопоставляются преимущества начала внедрения Scrum с реализацией пилотного проекта и одновременного перехода всех сотрудников к использованию Scrum. В главе 5, “Ваши первые проекты”, показано, как выбрать такой первоначальный пилотный проект.

Сосредоточьте внимание на самых важных причинах, обуславливающих необходимость перемен. Если ваша организация не является исключением из общего правила, вы наверняка можете составить длинный перечень причин снижения эффективности текущего процесса разработки: продукты не отвечают ожиданиям пользователей, разработка новых продуктов отнимает слишком много времени, низкое качество, неудовлетворительная моральная атмосфера в команде разработчиков, чрезмерные объемы работ, выполняемых сверхурочно, непредсказуемость календарных планов, высокие издержки разработки и т.п. Чтобы помочь людям осознать необходимость перемен, зачастую полезно заменить этот длинный перечень гораздо более коротким. Какие две или три причины вызывают самые большие проблемы? Этих причин должно быть вполне достаточно, чтобы оправдать необходимость перехода к Scrum. Сократив полный перечень причин до важнейших, мы сосредоточиваем внимание людей на самых веских доводах в пользу перехода к использованию Scrum.

Один из моих клиентов решил перейти к использованию Scrum потому, что его продукты утратили статус “лучшие в своей категории”. Клиенты продолжали пользоваться ими, но скорее по инерции и в силу лояльности к данному производителю, формировавшейся годами. Чтобы сосредоточить внимание на этой проблеме, я попросил их убрать из вестибюля штаб-квартиры своей компании все плакаты, призы и отраслевые награды за исключением тех, которые относились к прошлому году их деятельности. Убрав из вестибюля старые призы “Лучший продукт года”, мы обратили внимание сотрудников этой компании на тот факт, что ее клиенты постоянно задавали вопрос: “А что вы сделали для нас за последнее время?” После того как мы убрали подальше все эти знаки прежних заслуг компании, в вестибюле все равно оставалось изрядное количество наград. Но контраст с тем, к чему привыкли сотрудники компании, оказался впечатляющим и помог им осознать, что лучшие дни этой компании уже не вернуть, если не осуществить необходимые перемены.

Желание

Помимо осознания необходимости перемен, у людей должно возникнуть желание этих перемен. Я осознаю, что должен есть больше овощей, но у меня еще не возникло желание внести это изменение в свой рацион. До тех пор, пока мое осознание не превратится в желание, мой рацион не изменится. Инструктор и консультант по Scrum Мишель Слайгер (Michele Sliger) рассказывает о компании, переходу которой к использованию Scrum препятствовало именно такое отсутствие желания. Спустя пару недель после завершения учебного курса, в ходе которого сотрудники этой компании осваивали использование Scrum, Слайгер позвонил ее руководству и поинтересовалася, как идет внедрение Scrum.

Руководство этой компании проводило такую политику, что ее сотрудники заранее решили для себя, что гибкая методология разработки не приживется в их компании. Это единственная из известных мне групп, которая не поленилась узнать как можно больше о гибкой методологии разработки (не только из книг, но и от опытного консультанта по этой методологии), тщательно проанализировала свою культуру и политику, после чего сказала твердое “нет”. Можно ли назвать их подход практическим, реалистическим или они просто чего-то испугались и испытывали пессимизм в отношении внедрения у себя гибкой методологии?

разработки? Не знаю. Но никакая другая компания из тех, с которыми мне приходилось работать, не говорила ничего подобного. Возможно, таких компаний должно быть больше. Я с уважением отношусь к тому, что вместо того чтобы заниматься делом, к которому они не испытывают никакого энтузиазма, эти люди приняли твердое решение о неготовности к применению Scrum (какой бы ни была причина этого решения).

Поскольку они решились на расходы, связанные с привлечением к работе в компании инструктора по гибкой методологии разработки, по меньшей мере часть сотрудников этой компании должна была отдавать себе отчет в необходимости перемен. Но из истории, которую поведал нам Мишель Слайгер, мы можем заключить, что желание осуществлять дальнейшие перемены в данном случае оказалось недостаточным большим.

Переход от осознания того, что используемый в настоящее время процесс разработки уже не приносит требуемого результата, к желанию использовать какой-то другой процесс разработки для многих людей может оказаться очень трудным. В конце концов, раньше нас учили — как в учебных заведениях, так и на практике — использовать последовательный подход к разработке программного обеспечения. К тому же, несмотря на то что нас могут не устраивать отдельные элементы наших проектов, в результате упорного труда у нас сформировалась хорошая команда и появился хороший руководитель. Внедрение Scrum может перевернуть все с ног на голову. Наконец, может быть, сейчас не самый подходящий момент для перехода к Scrum.

Двадцать лет тому назад один мой приятель порекомендовал мне прочитать какой-нибудь из романов Джона Д. Макдональда (John D. MacDonald) из серии “Тревис Макги”. В тот же день я купил книгу *The Girl in the Plain Brown Wrapper* и приступил к ее чтению. Эта книга настолько мне не понравилась, что я бросил ее, не дочитав и до половины. Примерно через год я наткнулся на нее на своей книжной полке и решил сделать еще одну попытку. На этот раз она понравилась мне настолько, что я прочитал все двадцать книг из серии “Тревис Макги”. Очевидно, когда я впервые читал книгу *The Girl in the Plain Brown Wrapper*, у меня была другая психология, другие представления о жизни. Те же факторы могут вступить в игру и тогда, когда разработчики выслушивают доводы в пользу использования Scrum. Может быть, вы начали убеждать их в целесообразности перехода к Scrum в не самое подходящее для этого время? В таком случае остается надеяться лишь на то, что те же доводы, представленные в той же форме, но в другое время, вызовут у людей желание перейти к использованию Scrum.

Инструменты, способствующие возникновению желания перейти к Scrum

Выработать у людей желание перейти к использованию Scrum зачастую труднее, чем помочь им осознать необходимость изменения существующего порядка вещей. К счастью, существует немало инструментов, способствующих переходу от осознания необходимости перемен к желанию таких перемен.

Убедите людей в существовании более эффективного способа выполнения их работы. При выработке у людей осознания необходимости перемен необходимо сосредоточиться на ключевых проблемах, с которыми сталкивается организация или команда,

намеренная внедрить у себя Scrum. После того как мы перейдем от выработки осознания к выработке у людей желания перемен, необходимо сосредоточиться на том, как Scrum может помочь в решении этих проблем. Объединение этих двух фактов (что используемый в настоящее время подход уже не приносит желаемых результатов и что Scrum позволит решить эту проблему) может привести к “выключению” некоторых из работников и сделать их невосприимчивыми ни к одному из них. Однако чем больше работников осознают необходимость перемен, тем быстрее будет идти процесс. Лори Шублинг, о которой упоминалось в начале этой главы, пишет о заразительной природе желания.

Я была убеждена, что гибкая методология разработки способна нам помочь. Я составила соответствующий план, заручилась поддержкой директора и стала таким “внутрикорпоративным проповедником”. Поскольку я была искренне убеждена в высокой эффективности Scrum, игнорировать мои доводы людям было не так-то просто. Если кто-либо возражал мне, я давала ему аргументированный и исчерпывающий ответ. Мое желание сразу же подкупало одних и заставляло задуматься тех, кто все еще сомневался. Ряд ключевых специалистов проявили интерес к Scrum. Это очень помогло остальным членам нашей команды убедиться в огромных возможностях, которые таит в себе Scrum.

Создайте у людей ощущение необходимости перемен. Один из способов превратить осознание в желание — “завести” людей. Создавая ощущение необходимости перемен у одних, мы даем понять другим, что статус-кво не может сохраняться до бесконечности. Помните осознание мною необходимости кушать больше овощей? Допустим, мой врач позвонит мне завтра и скажет, что через полгода я умру, если не начну кушать брокколи, аспарагусы, цветную капусту и т.п. Наверняка я отреагирую на это предупреждение тем, что постараюсь выработать у себя любовь к овощам.

Сформируйте импульс. Вместо того чтобы сосредоточивать свое внимание на оппонентах Scrum, тратьте свое время и силы на помощь тем, кто уже является энтузиастом Scrum. Вместо того чтобы рассказывать оппонентам, что можно и чего нельзя сделать, рассказывайте это тем, кто разделяет ваши убеждения. Ваша цель — создать непрекращающийся импульс, движущую силу, когда каждый новый успех приводит к следующему. Когда Стив Грини (Steve Greene) и Крис Фрай (Chris Fry) из Salesforce.com проанализировали успешный переход их компаний к использованию Scrum, они начали советовать другим “сосредоточиться на достижении несколькими командами превосходных результатов” (2008). Вместо того чтобы “размазывать” поддержку тонким слоем по всем командам, пытайтесь представить эти первые успехи как очевидный результат внедрения Scrum. В этом случае другие тоже захотят добиться таких же успехов.

Заставьте команду воспользоваться Scrum для “пробной поездки”. Вместо того чтобы заниматься отвлеченными рассуждениями о Scrum среди членов своей команды, предоставьте им возможность как можно быстрее получить практический опыт использования Scrum. После этого они смогут обсуждать преимущества и недостатки Scrum вполне предметно. Полезно, например, организовать трехмесячное пробное использование Scrum. Это предоставит вашей команде возможность продвинуться дальше пары первых спринтов, которые наверняка вызовут у ваших сотрудников значительный дискомфорт. В конце такого трехмесячного пробного использования Scrum

проводите тщательный “разбор полетов” вместе со всеми участниками эксперимента и примите решение, как вам быть дальше. Это решение не должно формулироваться по принципу “или-или” (или мы будем продолжать использовать Scrum, или не будем). Если эксперимент с использованием Scrum привел к неоднозначным результатам или если мнения участников этого эксперимента разделились, третьим вариантом может быть продление эксперимента еще на несколько месяцев. Возможно даже, команда решит, что она еще не готова к использованию какого-то конкретного метода, и временно “положит его на полку”, но при этом продолжит работу со Scrum.

Приведите стимулы во взаимное соответствие (или по крайней мере устраните анти-стимулы). В организациях может существовать много программ стимулирования сотрудников (финансовых и прочих), которые могут препятствовать использованию Scrum. Во многих организациях действуют программы премирования отдельных работников за их значительный вклад в результаты работы своей команды или подразделения. Несмотря на то что подобная программа может казаться довольно эффективной, она работает против коллективного менталитета (“все мы — одна команда”), который должен присутствовать у членов команды, использующей Scrum. Программы премирования, которые вознаграждают тестеров в зависимости от количества обнаруженных ими дефектов (которые фиксируются в системе отслеживания дефектов), имеют примерно такой же разрушительный эффект.

Одна из организаций, с которыми мне довелось работать, откорректировала свою форму ежегодной отчетности, убрав из нее такие ориентированные на отдельных работников критерии, как знание работы, управление временем и умение сбалансировать несколько приоритетов. Она заменила их критериями, ориентированными на команду, такими как “помогает другим выполнять их работу”, “делится своими знаниями с другими”, “готов выполнять задания, не входящие в перечень его должностных обязанностей” и “добивается командных целей, связанных с созданием качественного продукта и с качеством”.

В другой компании я убедил владельца продукта и функциональных менеджеров пообещать команде какой-нибудь уникальный не денежный бонус, если конечный продукт с заранее утвержденным перечнем возможностей будет выпущен в точно намеченный срок. Несмотря на веру владельца продукта и менеджеров в то, что команда и в дальнейшем будет выполнять свою работу с высоким качеством, я попросил членов команды предложить руководству использовать какой-либо показатель качества. Я не хотел, чтобы они получили свой бонус, пожертвовав качеством. Команда предложила измерять качество количеством дефектов, выявленных в течение 30 дней после выпуска продукта. Они поставили перед собой цель добиться, чтобы количество выявленных дефектов оказалось меньшим, чем у двух предшествующих выпусков аналогичного размера. Спустя четыре месяца эта команда обеспечила к запланированной дате даже несколько большую функциональность, чем было обещано. Когда спустя месяц было измерено качество, члены команды получили свой бонус — четырехнедельный спринт, в течение которого они становились владельцами собственного продукта и могли работать по своему усмотрению — над чем пожелают. Они воспользовались этой возможностью, чтобы выполнить определенный рефакторинг, который давно хотели сделать. Один из тестеров решил поэкспериментировать с новым инструментом тестирования. Двое разработчиков добавили в один из разделов созданного ими приложения интерфейс сценариев. Это был очень удачный вид

бонуса, причем руководству удалось избежать проблем, которые обычно сопровождаются выплаты денежных премий.

Сконцентрируйтесь на том, чтобы развеять у людей опасения. То, чего мы опасаемся, зачастую влияет на наше поведение. Исходя из отрицательного прошлого опыта, владелец продукта может опасаться, что организация-разработчик выйдет из-под контроля и станет создавать только то, что сама пожелает. Это может заставить владельца продукта предпочесть процесс разработки с этапом сбора подобных предварительных требований, ведь такой процесс не позволит разработчикам создавать только то, что они сами пожелают.

С другой стороны, руководство может опасаться серьезного отставания от календарного плана. Это может заставить руководителей предпочесть такой процесс разработки, который позволяет как можно раньше получить точные оценки сроков готовности продукта. Руководители почти всегда знают, что не получат готовый продукт к обещанному сроку. Но, как они рассуждают, взяв с команды обещание изготовить продукт к более раннему сроку и оказывая постоянное давление на разработчиков, они получат продукт раньше, чем получили бы без таких ухищрений, и смогут избежать значительного отставания от календарного плана.

См. также Многие опасения являются результатом заблуждений, связанных с применением метода водопада, а также всевозможных фобий, связанных с гибкой методологией разработки. Эти опасения обсуждаются в главе 6, “Преодоление сопротивления”. Многие другие опасения рассматриваются во врезках “Возражение”, встречающихся в этой книге.

Архитектор может отдавать предпочтение разработке подробного предварительного проекта системы, поскольку ему это хорошо удается. Он опасается, что если убрать из проекта фазу подробного проектирования, то его авторитет в команде будет подорван и он не будет казаться умнее своих коллег. Общаясь с людьми, желание которых может подавляться теми или иными опасениями, пытайтесь развеять эти опасения, доказывая их необоснованность.

Помогите людям расстаться со своим прошлым. Люди не пожелают для себя никакого нового будущего до тех пор, пока не перестанут цепляться за прошлое. Каждый переход заключает в себе возможность тех или иных потерь, а любые потери вызывают у людей сожаление. Предоставьте людям возможность “немножко погоревать”. Выслушайте эти сожаления без лишних возражений. Эти утраты носят глубоко личный и субъективный характер. Вы никогда не убедите людей в том, что они слишком остро реагируют на эти утраты или что эти утраты “не так уж важны”. Не нужно даже пытаться убедить их в этом.

Не пытайтесь дискредитировать прошлое. Описывая предстоящий переход к “светлому будущему” в лице гибкой методологии разработки, не пытайтесь каким-либо образом принизить или дискредитировать прошлое. Какой бы процесс разработки вы ни использовали раньше, он помог вашей организации добиться определенных успехов. Он заслуживает, чтобы его оценили по достоинству. Он не может состоять из одних лишь недостатков. Уильям Бриджес (William Bridges), автор книги *Managing Transitions* (“Управление переходами”), описывает последствия формирования поддержки для новых инициатив за счет прошлых усилий.

Многие руководители, излучая энтузиазм в отношении будущего (которое, несомненно, окажется лучше прошлого), высмеивают прежние способы работы или высказываются о них в лучшем случае снисходительно. Поступая так, они лишь консолидируют усилия противников перехода, поскольку люди, как правило, отождествляют себя с этими прежними способами работы и принимают на свой счет любые нападки руководителей на них (2003, 34).

Вовлекайте людей в действия, связанные с переходом. Вербуйте на этой стадии как можно больше сторонников перехода. Идеальным сторонником является так называемый “лидер мнения”, пользующийся авторитетом среди тех, кого вы пытаетесь привлечь на свою сторону. Заразительный энтузиазм нескольких таких “лидеров мнения” может быстро распространиться на других сотрудников организации. Бенуа Хоул (Benoit Houle), руководитель Scrum-проектов в BioWare, убедился в этом на собственном опыте.

Мне очень повезло в том отношении, что я наладил продуктивные деловые отношения с одним из старших и очень авторитетных программистов в команде. Он был руководителем нашей первой, “пилотной” Scrum-команды на протяжении нескольких спринтов. Он испытывал огромный энтузиазм в отношении перехода к использованию Scrum и накупил кучу книг о Scrum и экстремальном программировании. Он сыграл чрезвычайно важную роль как руководитель Scrum-проекта, а его неподдельный энтузиазм заразил всех сотрудников.

См. также В главе 4, “Движение в направлении гибкости”, описано использование сообществ в поддержку перехода к применению Scrum как способа вовлечения людей в действия, связанные с переходом к Scrum.

Не обходите своим вниманием и скептиков. Спросите у сотрудников, что они хотели бы увидеть, попробовать или узнать, чтобы убедиться в целесообразности перехода к использованию Scrum, и изыщите способы удовлетворить этот их интерес.

Способность

Одного лишь осознания необходимости и желания перейти к использованию Scrum не хватит, если ваша команда не выработает у себя способность к работе согласно гибкой методологии разработки. В главе 1, “Гибкая методология разработки: нелегко, но перспективно мы кратко упоминали о том, что важным условием успешного применения Scrum является не только приобретение членами вашей команды новых навыков и умений, но и “удаление из памяти” своих старых навыков и умений. Scrum-команды сталкиваются со следующими серьезными проблемами.

- **Освоение новых технических навыков.** Разработчики, впервые применяющие Scrum в своей деятельности, как правило, начинают понимать, что, попрежнему оставаясь хорошими специалистами в своей области, они еще не достигли необходимого уровня квалификации в гибкой методологии разработки. Им приходится приобретать новые навыки и умения, которые им не были нужны прежде (что, возможно, и оправдывало их игнорирование). Например,

программистам придется научиться постепенно выстраивать проект системы. Тестерам нередко приходится учиться тестирувать систему, не полагаясь на документацию. Тем и другим наверняка придется освоить новые способы автоматизации тестов.

- **Приобретение навыков командного мышления и командной работы.** Многие из нас в результате многолетнего опыта привыкли работать в своем “отсеке” наедине с собой, надев наушники. Общение с коллегами сводилось к минимуму. “Вы разрабатываете свой кусок программы, я разрабатываю свой. Нам придется общаться, если будут выявлены какие-то проблемы в процессе объединения наших фрагментов”. Scrum-командам приходится отыскать мыслить в категориях *моих задач* и *ваших задач* и привыкать мыслить в категориях *наших задач*. Такой подход стимулирует более тесное сотрудничество между членами команды. Кроме того, подобный подход к работе формирует у людей чувство коллективной ответственности, что может оказаться несколько необычным для многих членов команды.
- **Приобретение навыков создания работоспособного программного обеспечения в жестких временных рамках.** Короткие, сфокусированные, ограниченные во времени спринты, характерные для Scrum, могут представлять немалую проблему для большинства команд, не привыкших работать подобным образом. Scrum-команды стремятся избегать неоправданных “перепасовок” от члена команды, специализирующегося на чем-то одном, члену команды, специализирующемуся на чем-то другом. Необходимость создавать работоспособное программное обеспечение к концу каждого спринта заставляет членов команды изыскивать способы устранения неоправданных передач из рук в руки и работать в более тесном взаимодействии друг с другом.

Инструменты, позволяющие выработать способность

В большинстве организаций выработка способности к применению гибкой методологии разработки (и последующее совершенствование этой способности) отнимает больше времени, чем осознание или выработка желания. К счастью, существует немало эффективных инструментов, позволяющих выработать у людей способность к применению гибкой методологии разработки.

Наставничество и обучение. Scrum существенно отличается от традиционного способа разработки программного обеспечения в том отношении, что для освоения Scrum обычно требуется обучение в сочетании с наставничеством в ходе выполнения конкретной работы. Лори Шубринг, которая возглавляла успешный процесс внедрения Scrum, говорит: “Наша способность добиваться успеха в деле применения гибкой методологии разработки начала формироваться в процессе обучения. По моему мнению, это было главным. Если бы мы не поняли чего-либо, мы, наверное, не смогли бы принять это с распростертыми объятиями”. Элизабет Вудвард (Elizabeth Woodward), один из лидеров процесса внедрения гибкой методологии разработки в компании IBM, согласна с этим мнением.

Мы начали переход к использованию гибкой методологии разработки, поставив перед собой цель организовать и провести в течение первого квартала под руководством инструкторов двухдневный курс обучения искусству формализованной гибкой методологии разработки (Disciplined Agile Development) в каждом из крупных

отделений нашей фирмы, разбросанных по всему миру. В течение первых трех кварталов мы провели обучение свыше 4400 разработчиков программного обеспечения по всему миру. Нам было очень важно создать для всех разработчиков одну и ту же базу знаний, выработать у них одинаковое представление о гибкой методологии разработки и сформировать у них ощущение настоятельности этих перемен. Оказалось, что у многих разработчиков сложилось неправильное представление о гибкой методологии разработки, которое нужно было скорректировать, чтобы люди с большей готовностью переходили к использованию этой методологии.

Определенное первоначальное обучение, ориентированное на выработку у людей готовности попробовать работать со Scrum и уяснить основные принципы этой методологии, по-видимому, необходимо для большинства компаний. Это первоначальное обучение сопровождается обучением или наставничеством в ходе выполнения конкретной работы (например, компания может воспользоваться услугами эксперта по разработке программного обеспечения на основе составления тестов; такой эксперт мог бы консультировать членов команды в процессе написания ими кодов).

Вскоре после начала внедрения Scrum компания Salesforce.com попросила меня организовать у них курс обучения для более чем 30 руководителей Scrum-проектов, в том числе несколько человек, которые не собирались выступать в этой роли в ближайшее время. Спустя два месяца руководители Salesforce.com попросили меня провести двухдневный сеанс обучения для 35 владельцев продукта. В этот же период меня попросили консультировать членов команд, работавших над конкретными проектами. Вспоминая это время, могу сказать, что, хотя обучение и наставничество использовались в этом случае уже на довольно ранних этапах внедрения Scrum, Крис Фрай и Стив Грини впоследствии сожалели о том, что не догадались “начать обучение владельцев продукта еще раньше и сделать курс обучения для них более интенсивным” и “еще раньше прибегнуть к услугам стороннего наставника”. Вот какой совет они дают компаниям, переходящим к использованию Scrum: “Пользуйтесь помощью профессионалов” (2008).

Внушите людям мысль об их ответственности. Наряду с обучением и наставничеством работникам нужно внушить мысль о том, что они несут ответственность за применение на практике новых навыков и умений, за приобретение которых организация заплатила немалые деньги.

Организуйте взаимный обмен информацией. В ходе выработки способности к применению гибкой методологии разработки людей будет захлестывать новая информация и они будут сталкиваться с множеством проблем. Предоставьте им возможность делиться этой информацией и сообщать другим о своих проблемах. Это можно делать путем “перекрестного опыта” команд. Предлагайте членам своей команды время от времени посещать ежедневные совещания или обзоры разработчиков какой-либо другой команды. Для распространения информации можно воспользоваться возможностями корпоративной сети, локальных википедий, сообществ практических пользователей Scrum и групп новостей. Еще одним способом взаимного обмена информацией является обращение к тем, кто уже освоил какой-либо новый навык, с просьбой провести краткий сеанс обучения этому навыку всех желающих. Если ваша группа достаточно велика, можно пойти еще дальше и провести однодневную миниконференцию, посвященную гибкой методологии разработки. Именно так поступили в калифорнийской штаб-квартире компании Yahoo!. Дж. Ф. Ансон (J. F. Unson), в то время наставник по Scrum в Yahoo!, так описывает этот подход.

В Yahoo! проводились однодневные внутрикорпоративные конференции, участие в которых могли принять все желающие (и даже предложить собственные темы для обсуждения). Мы провели ряд полезных сессий, посвященных, в частности, внедрению Scrum, распределенной гибкой методологии разработки и т.п. Подобный подход действительно помогает увеличить число сторонников гибкой методологии разработки и способствует тому, что люди начинают предлагать собственные решения. Разумеется, нам помогло и то обстоятельство, что наша компания уже накопила критическую массу сторонников гибкой методологии разработки, что обеспечило достаточно большое количество участников таких конференций (2008).

См. также Сообщества практических пользователей Scrum будут рассматриваться в главе 17, “Изменение масштаба Scrum”.

IBM применяет аналогичный подход, проводя ежегодно по две четырехдневных встречи, участие в которых принимают технические руководители и менеджеры со всего мира, а также местный технический персонал. Элизабет Вудвард описывает, как ее компания проводит по всему миру “мини-конференции” меньшего масштаба для сотрудников IBM, переходящих к использованию гибкой методологии разработки.

Каждая из таких встреч сосредоточена на гибкой методологии разработки, включая презентации, учебу, отчеты о полученном практическом опыте и заседания рабочих групп по разным темам, касающимся гибкой методологии разработки. Указанные заседания рабочих групп были особенно продуктивными, поскольку на них нам удавалось решать важные проблемы, в частности проблему использования Scrum в распределенной среде. Эти проблемы решались в ходе открытой дискуссии, в которой принимала участие большая группа квалифицированных специалистов, каждый из которых обладал уникальным опытом работы.

Ставьте разумные цели. Команды, перед которыми поставили непродуманную цель, наподобие “немедленно приступить к использованию гибкой методологии разработки”, нередко застаивают в растерянности, не зная, с чего начать. Успешный переход к использованию Scrum нужно разделить на этапы. Вместо того чтобы предлагать команде “приступить к разработке программного обеспечения на основе составления тестов”, руководителю Scrum-проекта следует предложить команде разработать с помощью этой методологии какую-либо функциональность к концу очередного спринта. Аналогично организации должны балансировать свое стремление к быстрому прогрессу с риском, который влечет за собой такая поспешность. Побуждая своих подчиненных ставить перед собой разумные, реалистичные цели, вы помогаете им избежать нерешительности, которая может возникнуть у них перед началом тех или иных радикальных перемен.

Просто делайте это. Не останавливайтесь в ожидании получить ответы на все вопросы, прежде чем приступить к намеченным преобразованиям. Наилучший способ выработать у себя способность сделать что-либо — это приступить к реальным действиям в данном направлении. Вот что советуют в связи с этим Грини и Фрай: “Экспериментируйте, проявляйте настойчивость и будьте готовы к возможным ошибкам” (2008).

Продвижение

Продвижение преследует три цели. Первая заключается в том, чтобы заложить фундамент для следующего прохождения цикла ADAPТ. Продвигая достигнутые на данный момент успехи, вы создаете хорошие предпосылки по выработке осознания для следующего раунда усовершенствований. Вторая цель заключается в том, чтобы распространять положительный опыт команд, уже использующих гибкую методологию разработки, рассказывая о достигнутых ими результатах. Наконец, третья цель заключается в том, чтобы распространять информацию об использовании гибкой методологии разработки среди сторонних групп, имеющих непосредственное отношение к внедрению Scrum. Многие из этих групп (например, отдел кадров, отдел сбыта, маркетинговый отдел, плановый отдел и отдел оборудования) могут оказывать решающее влияние на успешное внедрение Scrum. На фазе перехода нужно заботиться о том, чтобы эти группы не препятствовали проектной организации в деле внедрения гибкой методологии разработки.

Пропагандируя Scrum, не превращайте свою деятельность во что-то наподобие маркетинговой кампании. Многие работники вовлекаются в реализацию бесчисленных инициатив, направленных на осуществление перемен. Бесконечный парад таких инициатив вызывает у людей чувство пресыщения и усталости. Сотрудники многих организаций поняли одну простую истину: не спеши реализовывать очередную инициативу — на смену ей вскоре придет другая. Заявление о том, что “мы внедряем у себя гибкую методологию разработки”, наверняка приведет к саркастическим комментариям и скептицизму.

Чтобы противодействовать такому скептицизму, как минимум следует избегать присвоения громких названий мерам, направленным на внедрение гибкой методологии разработки. Команды, реализовавшие инициативу “Качество 2000”, которая сопровождалась инициативой “Лучше, быстрее, дешевле”, которая, в свою очередь, сопровождалась инициативой “Главное — клиенты！”, вряд ли серьезно отнесутся к кампании “Scrum — наша гордость！” Эксперт по организационному развитию Гленн Аллен-Мейер (Glenn Allen-Meyer) говорит, что организации присваивают названия и создают бренды для своих инициатив, связанных с осуществлением тех или иных перемен, просто потому, что это именно тот тип маркетинга, которым они привыкли заниматься.

Когда люди, приходя к себе на работу, высушивают всевозможные призывы по поводу перемен, придуманные специалистами по маркетингу, они понимают, что у них есть два выхода: либо принять эти призывы как руководство к действию, либо уволиться. Когда люди не видят реальной пользы от предлагаемых им перемен, но понимают, что с этим нужно смириться, если не хочешь потерять работу, разница между их подлинным отношением к переменам и необходимостью покориться неизбежности этих перемен создает у них чувство отчужденности по отношению к выполняемой ими работе (2000c, 24).

Заставить коллег поверить в целесообразность перехода к Scrum, а не просто согласиться с этим как с неизбежным злом (возможно, даже в тайной надежде, что вся эта затея благополучно провалится) — вот чего мы желаем достичь посредством успешной пропаганды. Одна из рекомендаций Аллена-Мейера заключается в том, чтобы избегать присвоения названий процессу перемен (2000a). Мой собственный опыт

непосредственного руководства процессом внедрения Scrum, участия в этом процессе или наблюдения за ним со стороны подтверждает правильность такого вывода.

Одно из объяснений этого феномена заключается в том, что человеку, как правило, труднее сопротивляться тому, что не имеет названия. Томас, руководитель команды в очень крупной компании, занимающейся разработкой коммерческого программного обеспечения, столкнулся с этим явлением на практике. Прочитав о Scrum в первых книгах и статьях, посвященных этой методологии, Томас решил, что она подойдет для его проекта, в котором принимали участие 40 человек. Без какого-либо предварительного обучения или общения с людьми, имеющими опыт использования Scrum, Томас начал внедрять Scrum в своей команде. Люди с готовностью согласились попробовать на практике эту новую для них методологию. Команда открыто рассказывала о том, что использует ее, поскольку не видела никаких причин скрывать этот факт. К сожалению, они неправильно поняли несколько ключевых элементов Scrum и потерпели неудачу.

Когда я познакомился с Томасом, он все еще интересовался Scrum и продолжал читать книги о Scrum. Из-за того что его проект потерпел неудачу, Томас решил принять участие в одной из конференций, посвященных Scrum, и прослушать двухдневный учебный курс по Scrum. Спустя восемнадцать месяцев после того, как первая попытка использования Scrum завершилась неудачей, Томас почувствовал себя готовым предпринять еще одну попытку. Команда Томаса также была готова к этому. Члены команды Томаса чувствовали, что использование Scrum таит в себе немало преимуществ, и это послужило для них решающим доводом в пользу повторения попытки. К сожалению, специфическая терминология Scrum — *ScrumMaster, sprint, журнал запросов на выполнение работ, ежедневное совещание разработчиков* и даже сам по себе термин *Scrum* — уже успела сформировать у сотрудников этой компании негативные ассоциации. Томас не решался сообщить своему начальнику о том, что они намерены еще раз попытаться использовать Scrum. Поэтому он сказал, что они намерены использовать некую “гибкую методологию разработки”. Применение Томасом и его командой их собственной версии “гибкой методологии разработки”, которая представляла собой ни что иное, как Scrum, но без дискредитировавшей себя терминологии, оказалось успешным.

Инструменты продвижения Scrum

Придя к заключению, что применение той или иной эффективной стратегии присвоения названий является инструментом, совершенно *неподходящим* для продвижения перемен, переключим свое внимание на инструменты, которыми *можно* пользоваться.

Пропагандируйте “истории успеха”. Как и в обычной жизни, коммуникации играют ключевую роль в той части цикла ADAPT, которая называется пропагандой. Особенно важно оповестить как можно более широкий круг лиц об успехах “первопроходцев Scrum” в данной организации. Исследование, проведенное McKinsey & Company, показало, что в успешных переменах акцент делался на побуждении работников строить свою деятельность на успехах, а не на их принуждении к устраниению проблем (2008). Пропагандистские меры помогают переключить внимание и энергию работников с проблем, выявленных ими на этапе осознания, и сфокусировать их внимание и энергию на успехах, которых им удалось достичь.

См. также Свободно распространяемая презентация введения в Scrum имеется по адресу www.mountaingoatsoftware.com/scrum-a-presentation. Содержание этой презентации можно редактировать.

Эффективным способом пропаганды успеха является проведение презентаций отчетов о внутрикорпоративном опыте команд, которые успешно внедрили у себя Scrum. Нет более эффективного способа пропаганды тех или иных дел, чем непосредственное знакомство с опытом людей, которые уже добились каких-то успехов в этих делах. Такие отчеты о внутрикорпоративном опыте можно сочетать с презентацией общего “Введение в Scrum”, чтобы те, кто еще не знакомы со Scrum, могли не только уяснить, что собой представляет Scrum, но и ознакомиться с историей какой-либо из команд, которые уже пользуются этой методологией. Если команды уже начали сбор показателей, соответствующие данные также можно включить в презентации. Ранние показатели могут быть всего лишь результатами опроса, показывающими, какой процент людей удовлетворен использованием Scrum, какой процент людей считает, что использование Scrum повысило их производительность, и какой процент людей считает, что использование Scrum привело к повышению качества. Впоследствии вы можете добавить более строгие показатели.

См. также Некоторые из показателей представлены в главе 21, “Как далеко вы продвинулись в деле внедрения Scrum”.

К счастью, наилучший способ пропаганды перехода к Scrum не предполагает каких-либо усилий с вашей стороны. Вот что говорит по этому поводу Бенуа Хоул, руководитель Scrum-проектов в BioWare: “Как и в вирусном маркетинге, самым единственным инструментом в нашем случае была молва, т.е. устное слово. Участники проекта, выполнявшегося с помощью гибкой методологии разработки, вовсю хваливали эту новую методологию (отношение к проекту каждого из его участников, как к своему кровному делу, большая предсказуемость, меньшие потери времени, сокращение времени реализации проекта и т.п.). Другие слушали все это и у них появлялось желание проверить новую методологию на собственном опыте”.

Мэтт Траксоу (Matt Truxaw), руководитель разработок и поборник гибкой методологии разработки в First American CoreLogic, рассказывает об аналогичном опыте.

Я уподобил бы использование гибкой методологии разработки постепенно обра- зующемуся водовороту, который, по мере своего формирования засасывает в себя все больше и больше людей и групп. Мы начинали с небольшой заинтересованности самих разработчиков. Регулярно рассказывая о гибкой методологии разработки и пропагандируя успехи ее первопроходцев, мы вовлекали в процесс использования этой методологии все большее число разработчиков. Поощряя “утечки информации” из команд, применяющих гибкую методологию разработки, а также предоставляя консультации и наставническую помощь группе управления проектами, мы добились признания гибкой методологии разработки со стороны большинства членов команды.

Организуйте сафари под названием “Гибкая методология разработки”. Один из моих любимых способов пропаганды Scrum впервые был применен в Google. Членам

команды, которые интересуются гибкой методологией разработки, но еще не имели возможности работать над реализацией проектов с ее помощью, предлагается поучаствовать в “сафари” под названием “Гибкая методология разработки”. Когда разработчики решают принять участие в таком сафари, они на пару недель подключаются к команде, использующей гибкую методологию разработки. Таким образом, они могут на практике получить представление об этой методологии и ее преимуществах. Ценность такого опыта заключается именно в том, что он был получен на практике, а не в результате чтения книг. Мне, безусловно, нравится эта идея, поскольку она полностью подтверждает мысль, высказанную Макиавелли еще 500 лет назад, когда он написал, что “люди не способны по-настоящему поверить в новое и непривычное для себя до тех пор, пока их не убедит личный практический опыт” (2005, 22).

Привлеките внимание и интерес людей. Привлекайте внимание людей всеми доступными вам способами. Чем чаще люди будут слышать о Scrum (или, что еще лучше, убеждаться на практике в преимуществах этой методологии), тем больше вероятность того, что окончательное внедрение Scrum станет неизбежным. Обеспечивая переход своего отдела к использованию Scrum, Лори Шубринг привлекала внимание людей к этой новой методологии несколько необычным способом.

Мы провели “День открытых дверей”, во время которого все желающие могли прийти в наш отдел и увидеть, как мы на практике используем Scrum. Мы составили кроссворд на тему Scrum и вручали призы тем, кому удалось его разгадать. Мы развесили на стенах постеры, поясняющие разные аспекты Scrum, такие как Scrum Board, график выполнения оставшихся работ, журнал запросов на выполнение работ и обязанности руководителя Scrum-проекта. Мы раздавали призы, предлагали угощение и напитки. Сотрудники внутрикорпоративных информационных служб помогали нам в приготовлении блюд и украшении помещения. Это мероприятие пользовалось огромным успехом среди работников компании.

В книге *Fearless change: Patterns for introducing new ideas* Мэри Линн Маннс (Mary Lynn Manns) и Линда Райзинг (Linda Rising) указывают, что угощение всегда является хорошей идеей. Оно не только привлекает дополнительных посетителей, но и улучшает настроение всех участников (2004). Бенуа Хоул всегда заботился о закуске и напитках для участников спринт-обзоров в BioWare, стремясь таким способом улучшить посещаемость этих мероприятий, которые, по его мнению, были “великолепным способом пропаганды успехов. Для участия в этих спринт-обзорах приглашались все сотрудники BioWare”. Кроме того, Хоул успешно использовал все пространство помещений, в которых проводились такие мероприятия: стены помещения были увешаны диаграммами, которые подробно иллюстрировали функционирование спрингта, привлекая внимание всех участников.

Стены наших комнат, увешанные диаграммами размером 4×6 дюймов, схемами, иллюстрирующими организационную структуру команд, и графиками выполнения оставшихся работ, превосходно отражали прогресс, достигнутый нашей командой, и полученные ею результаты. Из-за ограниченности площади стен в наших комнатах мы решили использовать также стены коридоров, на которых мы развесили диаграммы выполнения задач спрингта и иллюстрировали прогресс и достижения нашей команды.

Распространение

После трех лет усердной работы, личного участия буквально в тысячах ежедневных совещаний разработчиков и проведения десятков однодневных курсов “Введение в Scrum” для более чем 500 членов разных команд Джино было чем гордиться. Большая часть сотрудников отдела разработки уже использовала Scrum. Джино инициировал переход своей компании к применению Scrum, когда был одним из многих руководителей разработки. Благодаря тому, что его команда добилась быстрых результатов, Джино пошел на повышение: его назначили директором новой группы в компании, получившей название “Scrum Office”. Scrum Office оказывал поддержку и услуги всем командам, нуждавшимся в помощи. Scrum Office был своего рода аналогом отдела управления проектами (Project Management Office — PMO) в компании, выполнявшей традиционную разработку программного обеспечения. Джино хорошоправлялся со своей новой ролью, и вскоре большая часть разработчиков его компании выполняла проекты с помощью (в той или иной степени) гибкой методологии разработки. Еще до того как переход к Scrum был полностью завершен, Джино выдвинули на еще более высокий и ответственный пост, на котором ему предстояло выполнять еще более сложные задачи, связанные с внедрением Scrum. В его же прежней компании внедрение Scrum закончилось провалом. Дело вовсе не в том, что Джино там уже не работал. Причина неудачи заключалась в том, что никто в этой компании не задумывался над необходимостью учитывать последствия внедрения Scrum за рамками отдела разработки программного обеспечения.

Лично я представляю себе Scrum как ракету. Ракета взлетает с помощью двигателей, но ее тянет вниз сила гравитации. Если ракете удастся подняться на достаточную высоту, она выйдет на околоземную орбиту. Но если это ей не удастся, она неизбежно свалится на землю — на то самое место, с которого стартовала. Последствия внедрения Scrum нужно так или иначе учитывать в других подразделениях организации, чтобы силы “организационной гравитации” не помешали переходу к использованию Scrum в целом и чтобы в конечном счете мы не оказались в том самом месте, с которого все начиналось.

Джино удалось добиться признания Scrum среди программистов, тестеров, руководителей проектов, разработчиков баз данных, проектировщиков пользовательского интерфейса, аналитиков и т.п. Но само по себе использование Scrum более чем 500 разработчиками никогда не приведет к переменам в отделе кадров, отделе сбыта, отделе маркетинга и в других подразделениях, где по-прежнему остаются старые системы премирования, ориентированные на отдельных работников, и ежегодная отчетность. Отдел сбыта при этом по-прежнему обещает клиентам одноразовые усовершенствования, не обсуждая предварительно такие обещания с командами разработчиков.

Команда разработчиков не может долгое время игнорировать свое окружение. Если последствия применения Scrum не будут переноситься на другие подразделения организации, то “организационная гравитация” в конечном счете сведет на нет все ваши усилия по переходу к Scrum. Это, конечно, вовсе не означает, что и другие подразделения организации должны перейти к использованию Scrum. Это лишь означает, что необходимо по меньшей мере обеспечить совместимость других подразделений организации с использованием Scrum разработчиками.

Источники “организационной гравитации”

В предыдущих разделах этой главы рассказывалось об инструментах, которые способствуют переходу организации к использованию Scrum путем постепенной ADAPTации к этой методологии. Существует, по сути, лишь один инструмент для распространения гибкой методологии разработки на другие подразделения: взаимодействие с этими подразделениями. Поэтому, вместо того чтобы описывать некий перечень инструментов, предлагаю рассмотреть подразделения или группы, обладающие, как правило, значительной “организационной гравитацией”. Это группы, которые заслуживают особого внимания на том этапе цикла ADAPT, который мы обозначили как “распространение”. Работая с этими группами, нужно поставить перед собой цель образования, а не проповеди. Необходимо, чтобы другие группы уяснили преимущества, которые получает отдел разработчиков в результате использования Scrum. Нет необходимости превращать их в надежных сторонников вашего процесса. Скорее, нужно, чтобы они поняли некоторые из принципов этой методологии и как эти принципы могут порождать трения между вашей группой и ее окружением.

См. также Новая роль владельца продукта описывается в главе 7, “Новые роли”, а изменения в роли тестера описываются в главе 8, “Изменившиеся роли”.

Ниже приведен перечень групп, которые должны в своей работе учитывать факт использования Scrum. Обратите внимание, что я не включил в этот перечень тестирование и продукт-менеджмент. Эти группы являются фундаментальными *участниками* процесса внедрения Scrum, а не группами, которые должны каким-то образом учитывать в своей работе факт использования Scrum. Участие в Scrum владельцев продуктов и тестеров является очень важным и должно быть обеспечено уже в самом начале процесса перехода.

Отдел кадров. Вероятность возникновения трений между отделом разработки, использующим в своей работе Scrum, и отделом кадров очень велика. Политика отдела кадров во многих организациях препятствует успешному внедрению Scrum. Периодический процесс переаттестации, который заставляет руководителей классифицировать работников по степени их ценности, подрывает усилия по стимулированию командной работы. Не менее вредным является и то, что при этом делается упор на индивидуальный вклад работников и игнорируется командная работа.

См. также Влияние Scrum как на отдел кадров, так и на группу технического обеспечения подробно обсуждается в главе 20, “Кадры, техническое обеспечение и отдел управления проектами”.

Техническое обеспечение. Истории о навязчивом вмешательстве со стороны службы технического обеспечения давно уже превратились в настоящие легенды (DeMarco and Lister, 1999). Многим командам запрещается развешивать на стенах индексные карты, графики выполнения оставшихся работ и другие схемы, отражающие ход работ. Лишь некоторым командам разрешается обустраивать свои отсеки по собственному усмотрению; многие пришли к выводу, что наилучший способ “обойти” подобные требования — это выполнить необходимое обустройство в выходные дни,

придерживаясь принципа, что “лучше попросить прощения, чем добиваться разрешения”. Бенуа Хоул из BioWare рассказывает более обнадеживающую историю успешного учета последствий внедрения Scrum в деятельности служб технического обеспечения.

Служба технического обеспечения выполнила перепланировку наших помещений, чтобы способствовать успешной работе команд, использующих методологию Scrum. Они предоставили нам более просторные помещения, в которых могли комфортно работать от шести до восьми человек. Наша служба технического обеспечения создала веб-приложение, в котором фиксируется местоположение каждого нашего работника. Это позволило нам легко отслеживать любые перемещения технических средств с помощью внутрикорпоративной сети. Все наши сотрудники работают за одинаковыми столами, поэтому, как правило, речь идет лишь о перемещениях компьютеров и периферийного оборудования. Все происходит быстро и безболезненно.

Маркетинговый отдел. Во многих организациях группы разработчиков настолько слабы в деле планирования дат поставки готовых продуктов, что маркетинговые группы в конце концов перестают советоваться с ними по этому вопросу и просто ставят их в известность о принятых решениях. Это случается и в организациях, где маркетинговые группы гораздо более влиятельны, чем группы разработчиков, и, следовательно, могут диктовать им свои условия (в частности, желательные даты поставки готовых продуктов). Когда же речь идет о том, чтобы скорректировать действия маркетинговой группы после внедрения Scrum, необходимо сосредоточить усилия на информировании сотрудников маркетинговой группы о прозрачности, обеспечивающей Scrum.

Большинству маркетинговых групп — ничуть не больше, чем командам разработчиков — не нравится составлять планы работ на год. Возможно, маркетинговая группа предпочла бы планировать проведение рекламной кампании на девять месяцев вперед. Но, подобно командам разработчиков, они обычно предпочитают обеспечивать себе определенную свободу действий. Вместо того чтобы уже сейчас указать точное содержание рекламной кампании, они предпочли бы сегодня просто взять на себя обязательство провести рекламную кампанию, а точное ее содержание указать ближе к дате ее проведения. Постепенное уточнение планов, характерное для любой Scrum-команды, в сочетании со строгим соблюдением сроков должно вызвать благожелательное отношение маркетинговых групп, для которых подобный подход также предпочтителен.

Финансовый отдел. Финансовый отдел нередко пересекается со Scrum-проектами в двух областях. Первой из них является прогнозирование календарных планов и бюджетов проекта. Очень важно, чтобы сотрудники финансового отдела понимали — независимо от используемого процесса разработки, — что точность оценок, на которые способна команда разработчиков, не может быть выше 5% от описания нового продукта, набросанного на коленке на клочке бумаги. Такие нереалистичные запросы могут исходить от финансового отдела, который в прошлом уже страдал от неточных оценок, сделанных командами разработчиков. Чтобы восстановить доверие между финансовым отделом и командой разработчиков, может понадобиться немало времени.

После того как несколько Scrum-команд начнут демонстрировать успехи в деле применения этой новой методологии, может оказаться полезным провести встречу с сотрудниками финансового отдела. В ходе такой встречи следует признать недостатки прошлого планирования проектов, но вместе с тем показать следующее: несмотря на то что само по себе применение Scrum еще не может гарантировать своевременную поставку новых продуктов, оно позволяет еще на ранних стадиях выявить возможные ошибки, допущенные при составлении календарного плана.

Второй областью, в которой финансовый отдел нередко пересекается со Scrum-проектами, является составление отчетов о затратах рабочего времени. Несмотря на то что Scrum не требует, чтобы команда разработчиков строго отслеживала затраты рабочего времени, команда должна быть готова к такому отслеживанию, если финансовый отдел нуждается в этой информации. Это может понадобиться, например, в компании, выполняющей разработки по контрактам (в случае почасовой оплаты клиентом услуг этой компании).

Отслеживание затрат рабочего времени может быть связано с желанием финансового отдела капитализировать затраты на реализацию соответствующего проекта. Под капитализацией проекта подразумевается распределение затрат на разработку по всему предполагаемому сроку действия соответствующего проекта, вместо того чтобы относить эти затраты лишь к месяцу, когда они в действительности имели место. В разных странах действуют разные правила капитализации, причем многие из этих правил основываются на устаревших концепциях (включая и то, что проект не может быть капитализирован до тех пор, пока не будет продемонстрирована его техническая осуществимость). Основываясь на полученном нами опыте разработки, мы рекомендовали финансовым отделам исходить из того, что техническая осуществимость определяется лишь после выполнения анализа и проектирования. Без осуществления отдельных фаз анализа и проектирования в отношении любого Scrum-проекта финансовому отделу будет нелегко определить момент, когда можно будет с уверенностью говорить о технической осуществимости проекта.

Я обсуждал эту проблему со многими финансовыми отделами и всегда убеждал их в том, что техническая осуществимость достигается после буквально двух-трех спринтов. В конце концов, если команда создала работоспособную программу, которая включает одну функциональную возможность конечного продукта, то и весь этот продукт должен быть технически осуществимым. Я могу понять контраргументы, которые готовы привести противники такого утверждения, однако эти аргументы можно применить и к рассмотрению после выполнения анализа и проектирования, но до того, как разработчики приступят к написанию кода.

Кроме указанных, существуют и другие группы, действия которых также должны быть постепенно скорректированы для учета последствий внедрения Scrum. Например, вы можете работать с отделом управления проектами, отделом сбыта, отделом информационных технологий, операционным (учетно-расчетным) отделом, отделом разработки технических средств и другими группами, которые являются источниками “организационной гравитации”. Корректировка их деятельности с целью учета последствий внедрения Scrum очень важна для обеспечения долговременного успеха ваших начинаний.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Выявите ту часть цикла ADAPT, которая в наибольшей степени соответствует вашей деятельности. Сделайте то же самое в отношении вашей команды, вашего подразделения и вашей организации. Выявите три вещи, которые вы могли бы сделать, чтобы перевести вашу команду, подразделение или организацию на следующий уровень адаптации. Выберите одно из этих трех дел (или поработайте со своей командой, чтобы сократить такой перечень, если это целесообразно) и приступите к его выполнению.
- ❑ Если вы уже начали внедрять Scrum, подумайте над тем, как пропагандировать свой опыт и достижения. Определите способы пропаганды своих первых успехов, чтобы заинтересовать этим процессом других.

Взглянем на картину в целом

Подобно самой Scrum, ADAPТация к Scrum носит итеративный характер. Она начинается с того, что кто-то в организации осознает, что используемый в настоящее время способ работы не обеспечивает получения приемлемых результатов. По мере распространения информированности у некоторых сотрудников появляется желание испытать Scrum на практике и попытаться таким способом улучшить ситуацию. Эти первопроходцы, действуя методом проб и ошибок, вырабатывают у себя способность добиваться успехов с помощью Scrum. Новый статус-кво может возникнуть, когда успеха в использовании Scrum добьется хотя бы сравнительно небольшое число команд в организации.

Когда эти первопроходцы продолжат совершенствовать свои навыки использования Scrum, они начнут пропагандировать свои успехи, иногда — неформально (например, за обедом с друзьями из какой-нибудь другой команды), в других случаях — формально (например, в ходе презентации, организованной для всех сотрудников отдела). Это помогает членам других команд проходить все этапы цикла ADAPT: от осознания к желанию, а затем — к способности, после чего эти другие команды также начинают пропагандировать свои успехи.

Все эти первые успехи — замечательная вещь, но они немного стоят, если внедрение Scrum рассматривается как явление, касающееся исключительно отдела разработчиков. Для гарантирования долговременного успеха необходимо учесть последствия применения Scrum в других подразделениях, в том числе в отделе сбыта, маркетинговом отделе, операционном (учетно-расчетном) отделе, отделе кадров и в технических службах организации. Эти группы не пользуются Scrum в своей повседневной деятельности (сотрудникам отдела сбыта ни к чему составлять графики выполнения оставшихся работ, а сотрудникам технических служб нет нужды проводить ежедневные совещания). Но если эти группы не внесут в способ своего взаимодействия с группой разработчиков небольшие, но очень важные изменения, это подорвет способность группы разработчиков успешно пользоваться гибкой методологией разработки.

В следующей главе мы рассмотрим варианты моделей, которые можно воспроизвести в процессе перехода к использованию Scrum. Мы рассмотрим варианты постепенного и всеохватывающего перехода к использованию Scrum и поразмышляем над

тем, какой должна быть интенсивность пропаганды в начале внедрения Scrum. Кроме того, мы обсудим несколько способов распространения использования Scrum за рамки вашего первоначального проекта (или проектов). Понимание процесса ADAPT, изложенное в настоящей главе, послужит основой для решений, которые вам придется принимать в следующей главе.

Дополнительная литература

Derby, Esther. 2006. A manager's guide to supporting organizational change. *Crosstalk*, January, 17–19.

В этой статье Эстер Дерби, которая в соавторстве с Дайаной Ларсен написала книгу Agile Retrospectives, излагает десять соображений относительно того, что может сделать руководитель в поддержку инициативы, связанной с осуществлением тех или иных изменений. Большая часть этих соображений касается фаз осознания и желания.

Hiatt, Jeffrey. 2006. *ADKAR: A model for change in business, government and our community*. Prosci Research.

ADKAR — аббревиатура, означающая Awareness (осознание), Desire (желание), Knowledge (знание), Ability (способность) и Reinforcement (закрепление) — представляет собой групповую модель личных и организационных изменений. Она послужила отправным пунктом при создании модели ADAPT. В этой книге излагаются превосходные (хотя и несколько общего характера) советы относительно осознания, желания и способности.

Глава 3

Модели внедрения Scrum

Существует множество путей, которыми организация может внедрять Scrum. К счастью, если проанализировать опыт компаний, которые уже внедрили у себя Scrum, можно указать ряд распространенных шаблонов, позволяющих успешно перейти к использованию Scrum. В этой главе мы рассмотрим преимущества и недостатки четырех таких моделей, а также ситуации, в которых применение той или иной модели может оказаться оправданным. Эти модели базируются на двух вопросах, на которые нужно ответить, прежде чем переходить к использованию Scrum.

- Следует ли начинать внедрение Scrum с одной-двух команд или такой переход должен с самого начала охватывать все команды?
- Следует ли объявлять о своем намерении (возможно, лишь в рамках своей компании, а возможно, и во всеуслышание) или лучше заниматься внедрением Scrum без лишнего шума?

Помимо рекомендаций, связанных с ответами на два этих вопроса, мы рассмотрим три варианта распространения Scrum после начала внедрения данной методологии. Наконец мы выясним, когда новая Scrum-команда должна начать внедрение технических приемов гибкой методологии разработки.

Массовость: за и против

Принято считать, что переход к использованию Scrum (или какой-либо другой гибкой методологии разработки) лучше всего начать с какого-нибудь пилотного проекта, в ходе его реализации приобрести определенный опыт, а уж затем вовлечь в процесс внедрения Scrum остальные команды. При использовании такого подхода в организации выбираются, как правило, одна-три команды (из пяти-девяти человек каждая), которые достигают требуемого результата. Затем в процесс использования Scrum вовлекаются все остальные команды. По мере привлечения все большего и большего числа команд они учитывают опыт, полученный командами-первоходцами.

Применяются разные варианты такого подхода — все зависит от того, сколько сотрудников организация намерена вовлечь в процесс внедрения Scrum и как быстро она планирует осуществить этот переход. Этот традиционный метод перехода к использованию Scrum может применяться по-разному в зависимости от того, насколько организация склонна к риску и насколько она уверена в необходимости и успехе такого перехода. Например, в некоторых случаях команды-первоходцы завершают свои проекты еще до того, как к реализации своих проектов приступит вторая волна команд. Другие организации предпочитают, чтобы первая и вторая волны внедрения Scrum частично перекрывались между собой (например, вторая волна команд приступает к реализации своих проектов после того, как команды-первоходцы выполняют один или два спринта).

Несмотря на свою популярность, этот традиционный метод постепенного перехода к использованию Scrum устраивает далеко не всех. Например, компания Salesforce.com предложила противоположный вариант (Fry and Greene, 2006). Я помню, как 3 октября 2006 года Крис и Стив из Salesforce.com позвонили мне, чтобы сообщить, что у них в течение одного дня 35 команд перешли к использованию Scrum. Они сказали, что нуждаются в моей помощи. Первое, что пришло мне в голову, — это то, что им требуется помочь психиатра, а не эксперта по Scrum. Тем не менее я не собирался прятаться в кусты и согласился помочь им. Я уложил в чемодан свой ноутбук и какой-то из трудов Фрейда и отправился в Сан-Франциско, в штаб-квартиру Salesforce.com. Кое-что из увиденного мною там не было для меня большой неожиданностью: многих сотрудников Salesforce.com охватила настоящая паника, настолько внезапными и радикальными показались им перемены, уготованные для них руководством компании. С другой стороны, я увидел и то, что обеспечило в конечном счете успех этого крупномасштабного и стремительного преобразования.

Salesforce.com взяла на вооружение модель одновременного вовлечения всех команд в процесс перехода к использованию Scrum. Компании Salesforce.com присуща напористая, агрессивная модель поведения, главным двигателем которой являются достижения этой компании. Для нее не характерен осторожный подход, основанный на постепенном, поэтапном движении к намеченной цели. Когда ведущих специалистов этой компании ознакомили с идеей внедрения Scrum, их не пришлось долго уговаривать. Они сразу пришли к выводу: если Scrum подходит одной команде, то она наверняка подойдет и остальным командам. Зачем же в таком случае “растягивать удовольствие”? Поэтому решено было делать все одним махом.

Как ни странно, но оба эти подхода можно объединить. Все чаще применяется следующий комбинированный подход: сначала выполняется pilotный проект, в котором принимают участие от одной до трех команд, а затем внедрение Scrum выполняется одновременно во всех остальных командах. Пилотный проект служит в этом случае типичной цели: предоставить организации возможность ознакомиться со Scrum и применить данную методологию в специфических условиях этой организации. Однако пилотный проект в данном сценарии служит и более важной цели: повышению осведомленности организации о Scrum. Если к использованию Scrum переходят одновременно не менее 200 человек, иногда очень полезно указать на команду, которой такой переход удался, и сказать: “Мы собираемся сделать то, что удалось им”.

Почему следует отдать предпочтение постепенному переходу

Постепенный переход дает несколько преимуществ.

- **Постепенный переход обходится дешевле.** Одновременный переход всех команд к использованию Scrum почти наверняка обойдется дороже, чем постепенный переход. Когда обучаться новому способу работы приходится большому числу людей одновременно, организация вынуждена прибегать к помощи сторонних консультантов, специалистов, имеющих квалификацию ScrumMaster, и инструкторов. Более медленные темпы внедрения Scrum (в случае постепенного перехода) позволяют организации растить собственных специалистов и впоследствии использовать их в качестве консультантов, оказывающих услуги командам, еще недостаточно хорошо владеющим методологией Scrum. Постепенный переход обходится дешевле еще и потому, что ошибки, которые допускают команды-первоходцы, влияют только на сравнительно небольшую часть организации. Том Гилб (Tom Gilb), один из пионеров гибкой методологии разработки, писал: “Если вы еще недостаточно хорошо знаете то, что делаете, не делайте это в больших масштабах” (1988, 11).
- **Ранний успех почти гарантирован.** Тщательно отобрав первоначальный проект и членов команды, которая будет реализовать этот проект, вы практически гарантируете успех своего первого Scrum-проекта. Я допускаю, что это покажется вам преувеличением. Если вы отдали предпочтение постепенному переходу к использованию Scrum, то цель нескольких первых проектов заключается в том, чтобы добьть знания, которые обеспечат дальнейшее успешное распространение Scrum в масштабе всей вашей организации. Полезно начать с проекта и команды, которые без особых усилий обеспечат успех, а затем воспользоваться этим положительным опытом. К тому же первый успех очень важен, поскольку он помогает убедить скептиков и колеблющихся.
- **Риск существенно ниже, чем в случае одновременного перехода.** Одновременный переход всех команд к использованию Scrum может оказаться очень рискованной затеей. Мелкие ошибки будут разрастаться по мере того, как их будет повторять большое число команд. Возможно, самый большой риск одновременного перехода к использованию Scrum заключается в том, что у вас не будет “второго шанса”. Если переход к использованию Scrum выполняет одновременно вся организация и вы совершаете ошибку, которая повышает сопротивление сотрудников осуществляемым вами переменам, а затем возвращаетесь к старому методу разработки, параллельно размышляя над тем, в чем причина допущенной вами ошибки и как от нее избавиться, то маловероятно, что члены команды предоставят вам еще одну возможность начать переход к Scrum. К тому времени сопротивление переменам укоренится в людях до такой степени, что все ваши повторные попытки внедрить Scrum наверняка закончатся провалом. И наоборот, если вы будете переходить к Scrum постепенно и обнаружите серьезную проблему в самом способе вашего перехода, то сможете совершить следующий раунд в таком же масштабе, как и текущий, вместо того чтобы расширять этот масштаб, начиная, по сути, процесс перехода заново.
- **Постепенный переход к использованию Scrum вызывает меньший стресс у его участников.** Организации XXI столетия (и их работники) постоянно

испытывают стресс. Объявление о том, что *вся* проектная организация переходит к использованию Scrum — что, несомненно, повлияет на многие аспекты повседневной деятельности ее работников, — может превратить их жизнь в настоящий кошмар. Стресс, связанный с таким переходом, снижается, если переход к использованию Scrum выполняется постепенно, поскольку первоходцы со временем становятся наставниками и консультантами. Они облегчают переход других команд, рассказывая о своих успехах, честно обсуждая возникавшие у них проблемы и способы их решения.

- **Постепенный переход к использованию Scrum можно выполнить без реорганизации.** Большинство организаций, которые одновременно переходят к использованию Scrum, в той или иной степени неминуемо подвергаются реорганизации. Это может порождать дополнительный стресс и вызывать дополнительное сопротивление со стороны некоторых сотрудников. При постепенном переходе потребность в реорганизации может возникнуть значительно позже. В идеальном случае такую реорганизацию можно осуществить уже после того, как будет накоплен ценный опыт использования Scrum.

Почему следует отдать предпочтение одновременному переходу

Одновременный переход всей организации к использованию Scrum имеет не только недостатки, но и преимущества.

- **Можно снизить сопротивление переменам.** В случае постепенного перехода к использованию Scrum всегда найдутся скептики, которые будут надеяться, что использование Scrum закончится пилотным проектом. Подобно Цезарю, который перешел Рубикон, организация, которая решительно и одномоментно переходит к использованию Scrum, демонстрирует тем самым твердость своих намерений. Работники понимают, что возврата к прошлому быть не может. Такой уровень решимости может способствовать успеху намеченных преобразований.
- **Удается избежать проблем, порождаемых параллельной работой Scrum-команд и команд, использующих традиционный метод разработки.** Если к использованию Scrum переходит не вся организация, то вы рискуете тем, что в вашей организации будут параллельно работать не только команды, использующие Scrum, но и команды, использующие традиционный метод разработки. Это означает, что время от времени будут возникать ситуации, когда Scrum-команде придется координировать свои действия с командой, использующей традиционные методы разработки. Необходимость такой координации создает определенные проблемы вследствие разных подходов Scrum-команд и команд, использующих традиционный метод разработки, к таким вещам, как планирование, крайние сроки готовности продукции и коммуникации. Подобные проблемы отсутствуют, когда к использованию Scrum переходит вся организация. Стив Грини и Крис Фрай из Salesforce.com указывают, что “основным фактором, который заставил нас настаивать на одновременном переходе всей организации к использованию Scrum, было стремление избежать «организационного диссонанса» и желание решительных действий. Все должны действовать слаженно и синхронно” (2007, 137).

См. также В главе 19, “Сосуществование с другими подходами”, приведены советы по организации совместной работы Scrum-команды и команды, использующей традиционный метод разработки.

- **Одновременный переход всей организации к использованию Scrum завершится быстрее, чем постепенный переход.** Одно из центральных положений этой книги заключается в том, что организация никогда не “завершит полностью” переход к гибкой методологии разработки, поскольку ей всегда придется в этом совершенствоваться. Однако, несомненно, наступит момент, когда, оглянувшись назад, организация, внедряющая у себя Scrum, сможет сказать, что худшее уже позади. Организация, все подразделения которой переходят к использованию Scrum одновременно, может достигнуть этого момента гораздо быстрее.

Выбор между одновременным и постепенным переходами

Как упоминалось в начале этой главы, большинство авторов, пишущих о гибкой методологии разработки, рекомендует постепенный переход организации к использованию Scrum. Именно такой вариант внедрения Scrum применяется большинством организаций. Сочетание низкого риска и высокой вероятности успеха, присущих этому подходу, снижает убедительность аргументов, выдвигаемых сторонниками противоположного подхода. Постепенному переходу следует отдавать предпочтение в случаях, когда руководители организации испытывают сомнения относительно целесообразности внедрения Scrum. Успех, пусть даже в небольших масштабах, способен убедить даже самых стойких скептиков. Постепенному переходу следует также отдавать предпочтение в случаях, когда неудача может обойтись вам слишком дорого. Если цена неудачи слишком высока для тех, кто возглавляет переход, нужно отдать предпочтение постепенному переходу, даже если такой вариант не является наилучшим для организации в целом. Постепенный переход, наверное, — не идеальный вариант, если организация отчаянно нуждается в том, чтобы как можно быстрее извлечь выгоды из использования Scrum. (Но если вы все же решили отдать предпочтение постепенному переходу, постарайтесь как можно быстрее внедрить Scrum в масштабе всей организации.) Постепенный переход является более надежным, но вместе с тем более медленным вариантом.

См. также Ниже в этой главе рассматриваются способы распространения Scrum на другие команды.

Одновременный переход всей организации к использованию Scrum применяется сравнительно редко. Если время является критическим фактором, следует отдать предпочтение одновременному переходу. Несмотря на то что одновременный переход всей организации к использованию Scrum может обойтись дороже, затраты времени могут оказаться меньшими. Если главным является экономия времени, одновременный переход — оптимальное решение. Одновременный переход может быть оправдан и в случае, если вы, подобно Salesforce.com, хотите дать понять небольшому числу критиков и заинтересованных лиц, что альтернативы использованию Scrum

нет. Ни в коем случае не следует отдавать предпочтение одновременному переходу, если в вашем распоряжении нет достаточного числа специалистов, обладающих квалификацией ScrumMaster, которые могли бы оказывать помощь каждой из команд. В краткосрочной перспективе не так уж важно, являются ли эти специалисты работниками вашей организации или приглашены со стороны. Но не следует забывать, что в конечном счете все такие специалисты должны быть работниками вашей организации. Наконец, “размер имеет значение”. Если в вашей команде работают лишь десять человек, то одновременный переход к использованию Scrum представляется вполне оправданным. Но в случае команд, насчитывающих, к примеру, более 400 человек, одновременный переход может оказаться логистически невозможным.

Какой бы из двух вариантов внедрения Scrum вы ни выбрали, помните, что этот выбор является лишь первым из множества решений, которые вам придется принимать в процессе перехода к использованию Scrum. Следующим должно стать решение, касающееся обнародования информации о переходе.

Открытый и скрытый варианты перехода

Один из путей заключается в *открытом варианте перехода к использованию гибкой методологии разработки*. В этом случае команда или организация в целом объявляет во всеуслышание о намерении внедрить у себя Scrum. В зависимости от масштаба и значимости перехода такое объявление может принимать разные формы, начиная с информирования членов других команд (например, во время обеденного перерыва) о намерении внедрить Scrum и заканчивая публикацией соответствующих пресс-релизов в общенациональных средствах массовой информации. Каким бы ни был масштаб этого информирования, суть его заключается в оповещении окружающих о том, что соответствующая команда (или команды) намеревается перейти к использованию Scrum.

Противоположным вариантом перехода к Scrum является *скрытый переход*. В случае скрытого перехода о внедрении у них Scrum знают лишь члены соответствующей команды. Так продолжается до самого завершения проекта. Именно так внедряла у себя Scrum группа разработчиков одного из моих клиентов. Во время своего первого визита к этому клиенту я поговорил с Сарой, директором отдела управления проектами в этой компании. Она рассказала мне, что они уже добились определенных успехов в деле внедрения Scrum. Переход к использованию Scrum начался у них спустя какое-то время после того, как я провел у них двухдневный курс обучения, в котором приняли участие разработчики из их головного офиса. Сара поделилась со мной хорошо продуманным планом перехода к использованию Scrum более чем 200 разработчиками их компаний.

План Сары охватывал четыре “пилотные” команды, каждая из которых была выбрана на основе определенных критериев. Одна команда была выбрана исходя из желания ее членов работать в общем помещении (ранее они работали в помещении, разделенном на небольшие отсеки). Другая команда была выбрана по той причине, что именно ей одной из первых предстояло осваивать новую технологию, в которую их компания вложила значительные средства. Еще две команды были выбраны в качестве “пилотных” в силу не менее убедительных причин. План Сары оказался просто замечательным, поскольку позволял командам максимизировать обучение с самого начала переходного этапа.

Я ушел от Сары с намерением посетить каждую из этих четырех команд, чтобы получить из первых рук информацию о том, как у них идут дела. Однако оказалось, что мне придется иметь дело не с четырьмя, а с пятью командами. Поняв, о какой из этих пяти команд не рассказала мне Сара, я решил поговорить с ее членами еще раз. Оказалось, что речь идет о той части пилотного проекта Сары, которая не была официально санкционирована. Члены этой команды, увидев, как идут дела в “официальных” командах, прониклись энтузиазмом и решили подключиться к пилотному проекту Сары по собственной инициативе. Вообще говоря, они подозревали, что им не следовало бы поступать таким образом, и поэтому разместили свою доску с заданиями и график выполнения оставшихся работ в дальнем углу одной из своих комнат, куда практически никогда не заглядывало начальство. Я наткнулся на эти “следы преступления” только потому, что был плохо знаком с расположением их комнат и долго бродил по разным закоулкам, прежде чем нашел нужных мне людей.

Это может служить ярким примером скрытого перехода к использованию Scrum. Члены этой команды использовали Scrum, но не афишировали данный факт до завершения проекта. Возможны разные степени такой скрытности: одни держат свою деятельность в строгой тайне, другие же просто не болтают о ней на каждом углу.

Причины, по которым предпочтение отдается открытому варианту перехода

Существует немало убедительных причин, заставляющих людей отдавать предпочтение открытому варианту перехода к Scrum. К ним, в частности, относятся перечисленные ниже причины.

- **Каждый знает, чем вы занимаетесь, поэтому вам не остается ничего другого, как твердо придерживаться намеченного курса.** Обычный совет каждому, кто пытается избавиться от какой-либо вредной привычки или добиться какой-то цели, заключается в том, что в таком случае следует прибегнуть к помощи друзей. Решили ли вы сесть на диету, бросить курить или начать делать по утрам зарядку, об этом не помешает сообщить друзьям. Наверняка вы почувствуете после этого некое дополнительное давление, дополнительное желание добиться успеха в своем начинании. Дело в том, что вы объявили друзьям о своем намерении и не хотите ударить в грязь лицом. К тому же ваши друзья могут вас поддержать. То же самое относится и к внедрению Scrum.
- **Открытый переход к Scrum помогает лучше видеть цель и успешнее продвигаться к ней.** Открыто заявив о своих намерениях, вы тем самым инициируете дискуссии и “движение мысли” вокруг поставленной вами цели. В этом случае члены команды могут свободно обсуждать с окружающими проблемы, встречающиеся на пути к достижению цели. Они могут открыто рассказывать коллегам о своих успехах и неудачах. Те, кого интересует этот переход (и те, кто, возможно, и сам не прочь в нем поучаствовать), будут делиться своими советами, а оппоненты будут высказывать свои сомнения. В таком открытом обсуждении будут задействованы обе группы (как сторонники, так и противники перехода к Scrum). Это не только создаст дополнительные стимулы для команды, внедряющей Scrum, но и заставит их задуматься над тем, как опровергнуть возражения скептиков.

- **Действуя открыто, вы подтверждаете свою готовность двигаться намеченным курсом.** В скрытом переходе кроется некая неуверенность людей в своей способности достичь поставленной цели. Они словно говорят: “Нам, конечно, хотелось бы добиться этой цели, но мы предпочитаем подстраховаться на случай, если нас постигнет неудача и придется вернуться к прежним методам работы”. Заявив о своем намерении во всеуслышание, мы не сможем отказаться от поставленной цели, не потеряв при этом лицо. Заявляя открыто о своем намерении, мы говорим не только о желании инициировать переход к Scrum, но и о готовности успешно завершить этот переход.
- **Вы можете рассчитывать на организационную поддержку.** Пытаясь утаить от окружающих факт использования вами Scrum, вы лишаете себя возможности получать помошь извне. В процессе внедрения Scrum вы неминуемо столкнетесь с множеством проблем. Прежде чем отказываться от помощи своих потенциальных союзников в решении этих проблем, удостоверьтесь, что преимущества скрытого перехода в вашем случае перевешивают недостатки такого варианта.
- **Поставив перед собой цель и достигнув ее, вы посыпаете остальным мощный позитивный сигнал.** Объявив, что проект успешно завершен благодаря скрытому использованию Scrum, вы существенно ослабляете убедительность этого аргумента для скептиков. Убедительность ваших аргументов была бы значительно большей, если бы ваш переход к Scrum был открытым. Сообщив заранее о своих намерениях и достигнув поставленной цели, вы производите на окружающих (в том числе и на своих оппонентов) гораздо большее впечатление, чем в случае, если говорите им об этом постфактум.

Причины, по которым предпочтение отдается скрытому варианту перехода

Скрытый вариант перехода к Scrum может свидетельствовать о некоторой неуверенности в своих силах людей, выбравших этот вариант. Однако такой способ перехода к Scrum не лишен определенных преимуществ, перечисленных ниже.

- **Появляется возможность добиться определенного прогресса, прежде чем оппоненты начнут оказывать вам сопротивление.** Публичное объявление о намерении перейти к использованию Scrum может активизировать действия ваших потенциальных оппонентов и недоброжелателей, причем шансы на успех у них будут наибольшими до тех пор, пока вы не добьетесь первых успехов. Именно поэтому их нападки на вас будут самыми ожесточенными сразу же после того, как вы объявили о своих намерениях.
- **Скрытый вариант перехода к Scrum позволяет избежать дополнительного давления.** Если вы открыто объявляете о своем намерении перейти к применению Scrum, воспользовавшись для этого такими средствами коммуникации, как информационный бюллетень компании, внутрикорпоративная сеть и другие, ваша команда начнет испытывать на себе дополнительное давление и еще больший груз ответственности за возможную неудачу как собственно внедрения Scrum, так и реализации соответствующего проекта. Для каких-то команд такое дополнительное давление может оказаться даже полезным и благотворным. Однако после того, как проект будет завершен, вы не сможете сказать с

уверенностью, что именно является главной причиной этого успеха — использование Scrum или давление, которое испытала ваша команда. Боб Шац (Bob Schatz) и Ибрагим Абдельшрафи (Ibrahim Abdelshafi), возглавлявшие переход компании Primavera к Scrum, не афишировали процесс перехода.

С самого начала мы решили, что не будем афишировать переход к Scrum. Мы не хотели подвергать своих людей дополнительному стрессу. Наоборот, мы хотели дать им время адаптироваться к переменам. К тому же, заявляя о своем намерении использовать новый процесс и о его преимуществах, вы невольно вселяете в людей нереалистичные ожидания (2005, 37, 38).

- **Никто не узнает об этом до тех пор, пока вы сами не скажете.** Действуя скрытно, вы можете подождать успешного завершения проекта и лишь после этого сообщить окружающим, что данный проект выполнялся новым способом. Если же проект завершится провалом, вы можете учесть свои ошибки, исправить их и предпринять еще одну попытку. Лишь уяснив все нюансы, которые принесли вам успех в конкретной ситуации, вы можете рассказать людям о своем успехе и о том, как вы его достигли.
- **Если никто не знает о том, что вы используете Scrum, то никто не потребует от вас “прекратить это безобразие”.** Если вы сумеете так законспирироваться, что о выполнении проекта с помощью Scrum будут знать только его непосредственные участники, то никто не потребует от вас “прекратить это безобразие”. Я встречал команды, которые отдавали предпочтение скрытому варианту внедрения Scrum на основании того, что “легче попросить прощения, чем добиться разрешения”. Я также знал вице-президентов по разработке и отделы управления проектами, которые предпочитали скрытое внедрение Scrum потому, что хотели убедиться в реальных выгодах этой новой методологии, прежде чем открыто дискутировать о достоинствах Scrum с группами своих потенциальных оппонентов.

Выбор между скрытым и открытым вариантами перехода к использованию Scrum

Я пришел к выводу, что организации, готовые открыто переходить к использованию Scrum, имеют больше шансов рассчитывать на успех, чем те, которые предпочитают скрытый переход. Открытым переходу к Scrum нужно отдавать предпочтение в случаях, когда вы уверены в этой методологии и твердо намерены внедрить ее у себя. Аналогично открытому переходу к Scrum нужно отдавать предпочтение в случаях, когда ожидается жесткое сопротивление такому переходу, но вы намерены преодолеть его как можно быстрее.

Скрытому варианту перехода к использованию Scrum нужно отдавать предпочтение в случаях, когда вы хотите поэкспериментировать со Scrum в целом или с отдельными ее составляющими. Например, вы решаете проводить у себя ежедневные совещания (желательно не называть их *ежедневными совещаниями разработчиков*) и посмотреть, к чему это приведет. Затем вы решаете опробовать идею работы в режиме строгого ограниченных по времени спринтов. Если эти эксперименты завершатся успешно, можете начать называть то, что вы делаете, *гибкой методологией разработки* или *Scrum* и расширить ее внедрение. Кроме того, скрытый вариант перехода

к использованию Scrum нужно выбирать, когда вам не остается ничего другого. Если вам не хватает решимости заявить во всеуслышание “Мы используем Scrum” или если такое заявление вызовет слишком сильное сопротивление, действуйте скрытно.

Варианты распространения опыта использования Scrum

Одно дело — приступить к использованию Scrum. Другое дело — распространить опыт использования Scrum в организации. Если с самого начала вы не выбрали вариант одновременного внедрения Scrum в масштабе всей организации, вам придется переносить опыт применения этой методологии, приобретенный “пилотными” командами, на остальные команды организации. Существует три основных шаблона, которые можно применять для распространения первоначального опыта использования Scrum на другие команды. Первые два предполагают использование опыта, полученного членами “пилотной” команды, в остальных командах путем непосредственного включения этих специалистов в состав команд, которые начнут внедрять Scrum. Третий вариант предполагает расширение круга команд, переходящих к использованию Scrum, путем привлечения собственных наставников (т.е. наставников, являющихся работниками данной организации).

Метод “разделить и засеять”

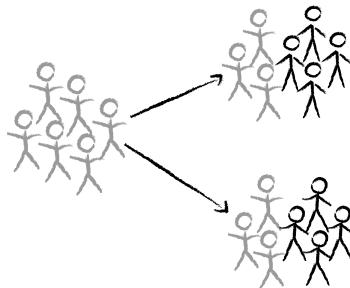
Метод “разделить и засеять” обычно используется после того, как первая пара команд внедрит у себя Scrum и выполнит по меньшей мере несколько спринтов. К этому моменту члены указанных команд начинают понимать, в чем заключается для них работа в составе Scrum-команды. Конечно, к этому времени они еще не уяснили всех тонкостей использования Scrum, но спринты, через которые они уже прошли, должны были завершаться созданием работоспособных программ. К тому же они уже должны были научиться работать вместе. Иными словами, команде еще предстоит долгий путь, чтобы в достаточной мере овладеть Scrum, но использование этой методологии уже стало для нее вполне естественным.

Именно на этой стадии недостаточного овладения Scrum мы разделяем команду на части.

При использовании метода “разделить и засеять” действующая (исходная) Scrum-команда делится на две части, каждая из которых становится ядром новой команды. К этим “центрам кристаллизации” добавляются новые люди, в результате чего получаются новые Scrum-команды. Эта модель представлена на рис. 3.1. На нем показано создание двух команд на основе одной исходной команды. Большая исходная команда может использоваться для “засева” не более четырех новых команд, особенно если кто-то из членов этой исходной команды ранее уже располагал опытом использования Scrum или имеет естественную склонность к этой методологии.

Новые члены команды могут быть либо сотрудниками организации, недавно принятymi на работу, либо старыми сотрудниками, приступающими к выполнению своего первого Scrum-проекта. Идея метода “разделить и засеять” заключается в том, что вновь сформированным командам (так сказать, Scrum-командам второго поколения) будет легче освоить механизм и технические приемы Scrum, поскольку им будут

оказывать помощь опытные члены команды. Этим новым командам предоставляется возможность выполнить несколько спринтов и оставаться в неизменном составе до тех пор, пока их новые члены не приобретут достаточный опыт использования Scrum. Затем цикл повторяется: действующие команды разделяются на меньшие команды, которые пополняются новыми членами. Этот цикл повторяется до тех пор, пока все, кого это касается, не приобретут необходимые навыки использования Scrum.



*Рис. 3.1. Метод “разделить и засеять”
в применении к двум командам*

В процессе такого внедрения Scrum вовсе необязательно, чтобы все вновь сформированные команды выполняли одинаковое количество спринтов: разделение каждой из команд можно выполнять по мере ее реальной готовности.

Метод “нарастить и разделить”

Метод “нарастить и разделить” является разновидностью метода “разделить и засеять”. Он предполагает добавление новых членов до тех пор, пока команда не окажется достаточно большой, чтобы ее можно было без проблем разделить на две, как показано на рис. 3.2. Сразу же после разделения каждая из новых команд окажется, скорее всего, на минимальном уровне желательной численности (от пяти до девяти человек). После того как новым командам будет предоставлена возможность выполнить один спринт в таком сокращенном составе, в них вливаются новые члены до тех пор, пока каждая команда не окажется достаточно большой, чтобы ее также можно было без проблем разделить на две. Этот цикл повторяется до тех пор, пока к использованию Scrum не перейдут все сотрудники организации, кого это касается.

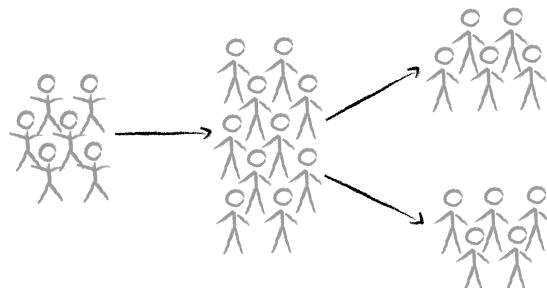


Рис. 3.2. Метод “нарастить и разделить” используетя для создания двух команд

Внутреннее наставничество

Примером третьего метода распространения Scrum, *внутреннего наставничества*, может служить внедрение Scrum в Philips Research. Эта компания приступила к внедрению Scrum и столкнулась с серьезной проблемой. Как и во многих организациях, в Philips Research были команды, которые успешно применяли Scrum, но были и команды, которым эта новая методология разработки давалась с большим трудом. Указанную проблему удалось решить с помощью внутреннего наставничества Кристу Врайенсу (Christ Vriens), работающему в этой же компании. В каждой из команд, которые успешно применяли Scrum, Крист выявил человека, который лучше остальных разобрался в особенностях этой новой методологии, и назначил его наставником в одну из команд, которые испытывали большие проблемы с внедрением Scrum.

Наставникам были вменены определенные обязанности. Например, они должны были посещать совещания, посвященные планированию, а также обзорные и ретроспективные совещания. Кроме того, они должны были каждую неделю посещать одно из ежедневных совещаний разработчиков, а также выделять еженедельно по два часа своего рабочего времени для оказания всей необходимой помощи своей “подшefной” команде. Впрочем, указанные обязанности не избавляли наставников от необходимости трудиться в полную силу в своей “родной” команде (правда, время, затраченное наставником на свою “подшefную” команду, засчитывалось ему как полноценное рабочее время, которое он был вынужден отрывать от “родной” команды).

Преимущества метода “разделить и засеять”

Преимущества метода “разделить и засеять” кроются в его природе, способствующей быстрому распространению опыта использования Scrum.

- **Нарашивать команды можно быстрее, чем в случае использования большинства других методов.** Каждая новая команда должна в идеальном случае включать по меньшей мере два члена предыдущей команды. Это означает, что, вероятно, уже через два-три спринта команду из восьми человек вполне можно будет разделить на четыре группы по два человека и использовать эти группы для “засева” второй совокупности команд. Если бы каждая из этих четырех команд насчитывала по восемь человек, то было бы уже 32 члена Scrum-команд. После выполнения нескольких спринтов эти 32 человека можно было бы использовать для “засева” еще 16 команд. При наличии восьми человек в каждой из этих команд мы получили бы после выполнения пяти или шести спринтов свыше ста (в совокупности) специалистов, располагающих опытом использования Scrum.
- **В каждой команде есть люди, располагающие опытом использования Scrum и способные помогать остальным членам команды.** Лишь в первых командах, переходящих к использованию Scrum, не окажется ни одного человека, который имел бы опыт применения Scrum. Деятельность всех последующих команд будет существенно облегчаться наличием в их составе по меньшей мере двух (а может быть, даже трех или четырех) человек, которые имеют опыт применения Scrum в виде как минимум пары спринтов. Это поможет снизить дискомфорт, испытываемый некоторыми людьми при освоении новых, совершенно непривычных методов работы.

Преимущества метода “нарастить и разделить”

Метод “нарастить и разделить” позволяет распространить опыт использования Scrum несколько медленнее, чем метод “разделить и засеять”, но обладает некоторыми важными достоинствами.

- **Нет необходимости разрушать существующие коллективы.** Основной проблемой при использовании метода “разделить и засеять” является то, что команды, которые едва научились использовать Scrum, расформировываются, а на базе их “осколков” создаются новые команды. Расформирование работоспособного коллектива — мера, к которой всегда нужно подходить с большой осторожностью. Нарастивая команду, прежде чем расформировать ее, мы избавляемся от указанного недостатка, поскольку в этом случае команда сохраняется вплоть до того момента, когда она оказывается достаточно большой, чтобы образовать две полные команды (у каждой из которых есть опыт использования гибкой методологии разработки).
- **У членов команды возникает ощущение относительной стабильности своего состава от спринта к спринту.** При использовании метода “разделить и засеять” команды постоянно подвергаются разделению и переформированию. Из-за этого у людей не успевает сформироваться ощущение товарищества и сплоченности коллектива. Поскольку метод “нарастить и разделить” предполагает разделение команды лишь после того, как она окажется достаточно большой, члены команды могут довольно долго работать вместе и поэтому у них гораздо реже возникает ощущение распада коллектива.

Преимущества внутреннего наставничества

Вообще говоря, метод внутреннего наставничества — мой любимый метод. И это неудивительно, поскольку у него есть серьезные преимущества перед другими методами.

- **Высокоэффективные команды не приходится расформировывать.** Важным недостатком методов “разделить и засеять” и “нарастить и разделить” является то, что успешно работающие команды приходится расформировывать и использовать образовавшиеся части в качестве фундамента для новых команд. При использовании метода внутреннего наставничества команды остаются в непреклоненности (с небольшим уточнением: в помощь команде назначается наставник “со стороны”).
- **Наставников для новых команд можно подбирать строго индивидуально.** При использовании, например, метода “разделить и засеять” мы имеем дело с чем-то наподобие коллективного наставничества: новая команда “засевается” несколькими членами, имеющими опыт использования Scrum. Кто-то из этих опытных членов будет хорош в роли наставника, кто-то — не очень. При использовании метода внутреннего наставничества для каждой новой команды можно подобрать самого подходящего наставника.
- **Наставников можно перебрасывать из команды в команду.** Спустя какое-то время команда и ее наставник становятся неинтересны друг другу. Новый наставник, посмотрев на ситуацию свежим взглядом, мог бы предложить способы ее улучшения. Переходя из команды в команду, внутренние наставники могут играть роль пчел, выполняющих перекрестное опыление команд новыми идеями.

Выберите свой подход

Выбор какой-либо из описанных моделей распространения Scrum обуславливается двумя главными факторами: насколько быстро нужно распространить Scrum среди команд и есть ли в вашем распоряжении квалифицированные внутренние наставники, которые могли бы помогать новым командам. Ответы на эти вопросы помогут выбрать модель, в наибольшей степени подходящую вашей организации.

Вообще говоря, когда нужно, чтобы организация как можно быстрее освоила Scrum, пользуйтесь методом “разделить и засеять”. Его “быстродействие” можно повысить двумя способами. Во-первых, команды можно разделять несколько раньше, чем обычно рекомендуется. Во-вторых, команды можно разделять на большее число новых команд, чем обычно рекомендуется (возможно, на четыре, а не на две), даже если это означает, что какие-то из новых команд получат от старых не самых идеальных наставников.

Однако при использовании метода “разделить и засеять” нужно проявлять осторожность, если соответствующая технология и специфика организации не поддерживают перемещения людей из одной команды в другую. Непостоянство состава команды, вообще говоря, отрицательно оказывается на ее производительности. Однако снижение производительности можно компенсировать за счет быстрого распространения Scrum среди участников крупного проекта или среди сотрудников организации. Однако в некоторых случаях нецелесообразно перемещать людей из одной команды в другую. Например, “засевание” .NET-команды программистами Java только потому, что у последних уже имеется опыт использования Scrum на протяжении трех спринтов, представляется мне не самой лучшей идеей.

Метод “нарастить и разделить” является, возможно, самым естественным подходом к распространению Scrum, поскольку он показывает, что, вероятнее всего, произошло бы, если бы процесс распространения Scrum был пущен на самотек, т.е. если бы команды не получали никакой помощи извне. В большинстве организаций люди переходят из одного проекта в другой, способствуя тем самым распространению полезного опыта. Метод “нарастить и разделить” — это просто более целенаправленный подход, чем стихийное распространение Scrum, для которого потребовалось бы гораздо больше времени.

Применять метод “нарастить и разделить” целесообразно в случаях, когда нас не толкает к использованию метода “разделить и засеять” необходимость быстрого распространения Scrum. Поскольку процесс наращивания и разделения команды носит менее агрессивный (и менее рискованный) характер, чем разделение и засевание команды, метод “нарастить и разделить” часто используется в сходных ситуациях, но отличающихся несколько меньшей срочностью. Применение метода “нарастить и разделить” целесообразно и в тех случаях, когда численность команды наращивается безотносительно к тем или иным методам распространения Scrum. В полном соответствии со своим названием метод “нарастить и разделить” показывает наилучшие результаты, когда численность команд наращивается.

Внутреннее наставничество можно использовать в качестве стратегии распространения Scrum либо как единственный используемый метод, либо в сочетании с другими методами для усиления эффекта. Этот метод показывает наилучшие результаты при определенных условиях.

- **Соответствующая группа настолько велика, что невозможно надеяться лишь на стихийное распространение полезного опыта.** Одно из преимуществ внутреннего наставничества заключается в том, что наставники могут переходить из одной команды в другую, распространяя в процессе таких переходов полезный опыт. Если ваша организация настолько мала, что распространение полезного опыта не представляет проблемы, то метод внутреннего наставничества для вас может оказаться неинтересным.
- **Разделение команд нецелесообразно для ваших проектов.** Если вас смущает какой-либо из недостатков, присущих разделению команд, то метод внутреннего наставничества может оказаться для вас подходящей альтернативой.
- **Имеется достаточное число внутренних наставников или можно воспользоваться помощью со стороны.** Идеальный наставник — это тот, кто обладает фундаментальным знанием Scrum и, возможно, не один год применял гибкую методологию разработки, даже не подозревая об этом. Таких людей не очень-то легко выявить загодя; это необязательно должны быть самые опытные члены команды. Если вы не располагаете достаточным числом внутренних наставников, попытайтесь вначале воспользоваться каким-либо другим методом. После того как достаточное число команд выполнит несколько спринтов, можно начать подкреплять какой-либо из методов “засева” путем привлечения внутренних наставников. Кроме того, можно компенсировать недостаточное количество внутренних наставников за счет прикрепления каждого из них к двум-трем командам. Если позволяет бюджет, можно также воспользоваться услугами сторонних консультантов, отказавшись от их услуг лишь после того, как в вашей организации появится достаточное число собственных наставников.

Внедрение новых технических приемов

Последнее решение, которое нужно принять агентам изменений, руководителям Scrum-проектов и членам новых Scrum-команд, заключается в том, насколько быстро соответствующая команда должна перейти к использованию новых технических приемов. Часть специалистов полагает, что все должно начинаться с технических приемов. Если команда использует правильные технические приемы (простое проектное решение, автоматизированное тестирование, парное программирование, рефакторинг и т.п.), то гибкая методология разработки окажется естественным результатом.

Альтернативная точка зрения заключается в том, что команду следует предоставить самой себе на более продолжительный срок, дав ей возможность самостоятельно выявить технические приемы, наиболее действенные в конкретной ситуации. Руководители Scrum-проектов, менеджеры и наставники могут время от времени побуждать команду к проверке на практике тех или иных технических приемов. Например, руководитель Scrum-проекта может спросить у команды: “А что было бы, если бы у нас были более автоматизированные тесты?” Но в общем случае сначала команде следует предоставить больше времени, чтобы ей не приходилось брать на вооружение (или хотя бы испытывать) новые технические приемы под внешним принуждением.

В этом разделе мы не только рассмотрим причины, заставляющие нас как можно раньше приступить к опробованию новых технических приемов, но и покажем, в каких случаях отсрочка с началом опробования таких приемов может оказаться благом.

Причины раннего опробования новых приемов

Можно назвать три веских довода в пользу раннего начала освоения новых технических приемов.

- **Становятся возможными очень быстрые улучшения.** Многие из технических приемов могут принести команде и организации достаточно быстрые результаты. Например, парное программирование может способствовать перекрестному обучению программистов, представляющих большее число функциональных областей системы. Внедрение процесса непрерывного наращивания может свести практически к нулю препятствия на пути к интеграции. Для других технических приемов — таких, как разработка программного обеспечения на основе составления тестов — характерна большая крутизна кривой обучения, но даже у них продолжительность освоения измеряется днями и неделями, а не месяцами и годами.
- **Если команда уже на начальной стадии не начнет испытывать новые технические приемы, то она может никогда не начать их применять.** Слишком многие Scrum-команды внедряют эту методологию лишь по минимуму, на том и останавливаются. Они считают, что улучшений, достигнутых ими в процессе освоения нового итеративного и инкрементального стиля работы, вполне достаточно. Отказываясь от освоения (или хотя бы от испытания) новых или усовершенствованных технических приемов, такие команды фактически отвергают большую часть улучшений, которых они могли бы добиться. Мне кажется, что такие команды, научившись работать итеративно, так и не стали по-настоящему гибкими. Габриэль Бенефильд (Gabrielle Benefield) утверждает, что столкнулась с подобной проблемой в Yahoo!, когда занимала в этой компании должность директора по гибкой методологии разработки новых продуктов.

Наиболее очевидные симптомы нарушения нормальной деятельности отдела разработки новых продуктов в Yahoo! обнаружились на уровне проекта и команды, а не на уровне технических приемов или инструментов. В результате компания поначалу сосредоточилась на внедрении Scrum. Активно обсуждался вопрос о том, следует ли параллельно с этим внедрять методы гибкой разработки. Оглядываясь назад, можно сказать, что компания существенно выиграла бы от такого параллельного внедрения (2008, 461).

- **Это может решить самые насущные проблемы проекта.** Знакомство команды с техническими приемами гибкой методологии разработки может решить целый ряд типичных проблем проекта, в том числе проблемы плохого качества, чрезмерно усложненные проектные решения, длинные циклы поставки и т.д. Однако есть проблемы иного рода, которые не решаются с помощью указанных приемов. Например, для проекта, у которого отсутствует владелец продукта, будет характерно замедленное или неправильное принятие решений. Эту проблему нельзя решить исключительно путем освоения новых технических приемов. То же самое можно сказать о проекте с несколькими владельцами продукта, имеющими конкурирующие между собой повестки дня, или о проекте, который разрабатывают люди, испытывающие сильную личную неприязнь друг к другу. Если самыми насущными проблемами вашего проекта являются проблемы,

решаемые с помощью одного или нескольких общепринятых приемов гибкой методологии разработки, попытайтесь сосредоточиться на использовании этих приемов в самом начале перехода к применению этой методологии.

Причины позднего опробования новых приемов

Наряду с существованием убедительных доводов в пользу как можно более раннего освоения командой новых технических приемов существуют не менее убедительные доводы, побуждающие не торопиться с их освоением.

- **Может существовать жесткое сопротивление использованию некоторых приемов.** Внедрение определенных технических приемов может создать одну из самых больших трудностей, с которыми вам придется столкнуться в процессе перехода. Многие люди упорно противятся всему новому, в частности таким методам, как простые проектные решения, парное программирование и разработка программного обеспечения на основе составления тестов. Несмотря на то что у вас могут быть достаточно веские поводы в пользу того, чтобы с самого начала подталкивать команду к опробованию новых методов, следует сопоставить эту необходимость с риском повышения сопротивления.
- **Не исключено, что члены команды начинают испытывать перегрузки.** Во многих организациях само по себе изучение основ работы в Scrum-команде может оказаться очень непростым делом. Дополнительный стресс, обусловленный необходимостью параллельного освоения новых технических приемов, может оказаться для некоторых команд чрезмерным и заставить их отказаться от каких-либо действий в этом направлении. Если же предоставить команде достаточно времени, то стресс, вызванный необходимостью поставлять работоспособное программное обеспечение в строго ограниченных рамках Scrum-спринтов, может привести эту команду к пониманию, что ей необходимо испытать новые технические приемы.

Последнее соображение

Эта глава посвящена двум вопросам, на которые приходится отвечать каждой организации, которая намерена внедрить у себя Scrum. Следует ли начинать внедрение Scrum с одной-двух команд или такой переход должен с самого начала охватывать все команды? Следует ли объявлять во всеуслышание о намерении внедрить Scrum или лучше заниматься внедрением Scrum без лишнего шума? Ответы на эти вопросы вовсе необязательно должны быть однозначными. Что касается и первого, и второго вопроса, то в большинстве организаций возможны промежуточные варианты. То же самое относится и к моделям распространения Scrum: любую из них можно использовать либо самостоятельно, либо в сочетании с другими моделями — все зависит от конкретных обстоятельств. К примеру, вы можете сначала отдать предпочтение методу “разделить и засеять”, но по прошествии времени (и после появления достаточно-го числа команд) этот процесс можно замедлить и предоставить командам возможность разрастись, прежде чем разделять их, и параллельно с этим ускорить обучение за счет использования внутреннего наставничества. Кроме того, какую бы модель вы ни выбрали, тем, кто возглавляет переход к использованию Scrum (а также тем, кто

участвует в нем), следует решить, каким должен быть объем изменений на каждом отрезке времени для команды или команд, внедряющих у себя Scrum. Если объем изменений окажется слишком большим, команды могут быть дезориентированы; если же изменений окажется слишком мало, вы рискуете тем, что бесконечная череда мелких изменений переутомит людей.

Джошуа Кериевски (Joshua Kerievsky), старший консультант в Cutter Consortium, является сторонником осуществления всех перемен “одним махом”. Он категорически против того, что сам называет “продвижением вперед мелкими перебежками”, поскольку, по его мнению, такой переход

- оказывается более мучительным (процесс изменений чрезмерно затягивается);
- не позволяет решить коренные проблемы;
- редко позволяет выполнить переход с самого начала и до конца;
- добивается изменений слишком медленно, чтобы обеспечить выигрыш;
- как правило, осуществляется без помощи экспертов, что приводит к дорогостоящим ошибкам, которых легко можно было бы избежать (2005).

Несмотря на то что Кериевски обращает внимание на действительно важные проблемы, он в конечном счете рассматривает переход к использованию гибкой методологии разработки как единовременное действие, у которого есть начало и конец. Однако переход к использованию гибкой методологии разработки, такой как Scrum, представляет собой процесс постоянного совершенствования, у которого не может быть заранее определенного конечного состояния. Вот почему неправильно говорить о выполнении перехода “с начала и до конца” или о “чрезмерном затягивании процесса изменений”. Перемены — это не то, что организация может пройти “с начала и до конца”. Перемены в наше время — это постоянное, непрекращающееся явление.

В статье, опубликованной в журнале *Agile Journal*, Лиз Барнетт (Liz Barnett) излагает точку зрения, отличную от точки зрения Кериевски.

Медленное начало — самый подходящий вариант. Для подавляющего большинства компаний, намеренных внедрить у себя гибкую методологию разработки, медленный, пошаговый переход к использованию такой методологии является наиболее прагматичным способом улучшения результатов деятельности команд, занимающихся и разработкой программного обеспечения, и повышения эффективности управления риском. Осуществляя изменения, связанные с организацией, процессом и технологией, команды могут постоянно оценивать достигнутый ими прогресс и определять, какие из последующих шагов являются для них оптимальными. Это — гибкий способ достижения гибкости (2008).

Кент Бек (Kent Beck) и Синтия Андрес (Cynthia Andres), авторы книги *Extreme Programming Explained*, согласны с таким утверждением, указывая на необходимость начинать внедрение гибкой методологии разработки с некоего подмножества технических приемов и новых способов работы и последующего постепенного (по одному за раз) совершенствования этих приемов и способов.

Удобно начинать с пошагового, постепенного изменения. Нам кажется, что, прочитав эту книгу и решив применить полученные знания на практике, не следует браться за одновременное освоение всех технических приемов и новых способов

работы и за одновременное применение всех новых принципов в непривычных обстоятельствах. Для освоения технических приемов, характерных для экстремального программирования, а также принципов, положенных в его основу, требуется определенное время. Экстремальное программирование приносит максимальный эффект, когда действуются все его возможности, но каждому, кто решил его осваивать, требуется какая-то стартовая площадка (2004, 55).

Это соображение подсказывает, о чём пойдет речь в следующей главе. После того как вы решили перейти к использованию Scrum, уяснили, какими будут последствия предстоящих перемен, и приняли решение относительно того, каким именно способом вы, вероятнее всего, будете двигаться к намеченной цели, наступает момент начала осуществления перемен, которых требует Scrum. Как правильно указывают Бек и Andres, Scrum лучше всего осваивать итеративно. Мы рассмотрим, как пользоваться концептуальным каркасом Scrum (наряду со специализированными сообществами, которые называются сообществами в поддержку перехода к Scrum) с целью внедрения и развертывания Scrum, осуществления постоянных усовершенствований и распространения идей гибкой методологии разработки во всей организации.

Дополнительная литература

Beck, Kent, and Cynthia Andres. 2005. Getting started with XP: Toe dipping, racing dives, and cannonballs. PDF-файл на веб-сайте *Three Rivers Institute*: www.threeriversinstitute.org/Toe%20Dipping.pdf.

Пользуясь метафорой вхождения в воду человека, решившего искупаться, Бек и Andres описывают три подхода к внедрению экстремального программирования. Они делят людей на тех, кто входит в воду осторожно, трогая ее вначале кончиками пальцев ноги, затем погружая ступни ног и т.д. (иными словами, они предпочитают медленное, поэтапное погружение); тех, кто бросается в воду с разбега, поднимая вокруг себя столб брызг (иными словами, они предпочитают решительное, одномоментное погружение, несмотря на неприятные ощущения, охватывающие их в первый момент); и тех, кто погружается в воду довольно быстро, но под наблюдением опытного тренера (иными словами, они предпочитают как можно быстрее осуществить значительный объем изменений, но делают это под руководством опытного наставника).

Benefield, Gabrielle. 2008. Rolling out agile in a large enterprise. В сборнике *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, 461–470. IEEE Computer Society.

В этой статье приводится подробная информация о крупномасштабном проекте по внедрению Scrum в компании Yahoo!. Приводятся подробные сведения о достижениях и просчетах, которые имели место в процессе внедрения.

Elssamadisy, Amr. 2007. *Patterns of agile practice adoption: The technical cluster*. C4Media.

Основное внимание в этой книге, которая представлена в формате PDF на сайте www.infoq.com, уделяется техническим приемам, которые следует освоить

командам, внедряющим у себя гибкую методологию разработки. Материал данной книги удачно дополняет описание методов, представленное в этой главе.

Hodgetts, Paul. 2004. Refactoring the development process: Experiences with the incremental adoption of agile practices. В сборнике *Proceedings of the Agile Development Conference*, 106–113. IEEE Computer Society.

В этой статье отражен опыт инструктора по Scrum Пола Ходжеттса (Paul Hodgetts), связанный с переходом значительного числа команд к использованию гибкой методологии разработки. Основываясь на опыте этих проектов внедрения, автор сравнивает достоинства и недостатки постепенного инкрементного внедрения гибкой методологии разработки с одновременным внедрением по принципу “все сразу”.

Striebeck, Mark. 2006. Ssh! We are adding a process... В сборнике *Proceedings of the Agile 2006 Conference*, под ред. Joseph Chao, Mike Cohn, Frank Maurer, Helen Sharp, and James Shore, 185–193. IEEE Computer Society.

Марк Страйбек описывает процесс внедрения гибкой методологии разработки в Google при создании интерфейсного приложения AdWords. Он описывает сочетание постепенного и скрытого перехода с постепенным инкрементным добавлением новых технических приемов.

Глава 4

Движение в направлении гибкости

Исторически сложилось так, что когда какой-либо организации требовалось осуществить те или иные преобразования, она выполняла соответствующую “программу перемен”. Руководством организации составлялся план преобразований с указанием сроков их начала и конца, после чего план доводился до конкретных исполнителей. Подобная схема работала удовлетворительно в эпоху, когда преобразования нужно было выполнять примерно раз в несколько лет. Кристофер Эйвери (Christopher Avery) писал: “Я полагаю, что в 60-е и 70-е годы этот подход был, наверное, более успешным, чем в 90-е годы и в наши дни, поскольку в результате глобализации конкуренции частота перемен повысилась и указанная модель осуществления перемен утратила свою эффективность” (2005, 18). Кристофер Эйвери продолжает свою мысль, утверждая, что, “поскольку в наши дни перемены происходят столь стремительно, что любые попытки осуществления «запрограммированных преобразований» уже не срабатывают, нам, возможно, придется приспособиться (в организационном смысле) к осуществлению гораздо более мелких преобразований на постоянной основе” (20).

На какой бы стадии внедрения Scrum вы ни пребывали в данный момент (то ли вы находитесь в самом начале этого процесса, то ли уже готовы к оптимизации использования Scrum), управлять переходом к применению Scrum нужно гибко. Использование итеративного процесса перехода — осуществление мелких преобразований на постоянной основе — является вполне логичным способом внедрения процесса разработки, который сам по себе является итеративным. Именно такой способ внедрения, вероятнее всего, приведет к успеху. Вот почему я считаю, что процессом внедрения Scrum лучше всего управлять с помощью самой Scrum. Итеративная природа Scrum, использование фиксированных временных рамок и акцент на коллективном труде дают мне основание утверждать, что эта методология лучше всего приспособлена к управлению таким масштабным проектом, как достижение и последующее совершенствование гибкости при разработке тех или иных проектов.

В 2004 году руководители компании Shamrock Foods пришли к выводу, что темп перемен в их отрасли необычайно высок. К тому времени Shamrock Foods, один из десяти крупнейших дистрибуторов продуктов питания в Соединенных Штатах Америки, уже более 20 лет использовала традиционный процесс стратегического планирования по принципу “сверху вниз”, затрачивая ежегодно по несколько месяцев на составление пятилетнего плана, который устаревал еще до того, как на нем успевали высохнуть чернила. Чтобы решить эту проблему, главный исполнительный директор Shamrock Foods Кент Макклелланд (Kent McClelland) отказался от безнадежно устаревшего подхода, которым его компания пользовалась уже двадцать лет, и начал применять итеративный процесс стратегического планирования на основе Scrum.

Процесс, который применяется в Shamrock Foods, создавался на базе ежеквартальных стратегических совещаний [спринтов]. Члены команды собирались на день где-нибудь вдалеке от своего офиса, чтобы сопоставить результаты, достигнутые компанией, с планами действий за предыдущий квартал. Мы просили их указать самое важное, что им удалось выяснить о стратегии компании с момента проведения предыдущей встречи и высказать свои соображения относительно того, как следовало бы интегрировать эти находки в стратегию предстоящего периода. Группа составляла новый план действий на этот предстоящий период. Помимо ежеквартальных спринтов, каждый год проводились трехдневные совещания, на которых участникам предлагалось оглянуться в прошлое и заново оценить стратегические предположения компании (McFarland, 2008, 71).

В этих спринтах принимали участие сорок пять менеджеров и работников; состав участников был подобран таким образом, чтобы было обеспечено представительство всех подразделений и функциональных областей. В начале каждого ежеквартального спрингта эта группа выбирала несколько ключевых областей, в которых, по мнению членов группы, компания должна была добиться прогресса (эти области были названы темами). Поскольку Shamrock Foods использовала Scrum не для разработки программного обеспечения, а для совершенствования организации, такие темы отражали широкие бизнес-цели данной компании. В качестве примеров можно привести повышение доходов, получаемых от тех или иных брендов Shamrock Foods, повышение качества обслуживания крупных клиентов компании, таких как Burger King, а также улучшение способности компании принимать на работу, удерживать у себя и развивать талантливых работников.

Многие инициативы, связанные с корпоративным совершенствованием, завершаются провалом, поскольку соответствующие планы оказываются неконкретными и недееспособными. Поскольку Shamrock Foods внедрила у себя Scrum, работники компании не ограничивались лишь формулированием тем для совершенствования: “Участники процесса планирования сформулировали ряд вполне конкретных и поддающихся измерению стратегических инициатив, которые были призваны реализовать каждую из стратегических тем, и определили их приоритеты. Затем они составили подробные планы действий и поставили (поддающиеся измерению) цели, которые, по их мнению, могли быть достигнуты не позднее, чем через 90 дней” (McFarland, 2008, 71).

История Shamrock Foods не только иллюстрирует широкую применимость Scrum, но и может служить наглядным примером использования Scrum для управления деятельностью, направленной на организационные улучшения. В этой главе мы

рассмотрим, как использовать Scrum для внедрения и последующего непрерывного совершенствования Scrum путем активного вовлечения в эту работу сообществ единомышленников (таких, как те 45 человек, которые направляли действия по организационным улучшениям Shamrock Foods).

Журнал совершенствования

Точно так же, как в любом Scrum-проекте ведется журнал запросов на выполнение работ, для отслеживания действий, связанных с внедрением Scrum в вашей организации, нужно вести *журнал совершенствования*. В нем фиксируются все улучшения, которых может достичь соответствующая организация в деле использования Scrum. Когда IBM начала внедрять у себя Scrum, журнал совершенствования IBM включал следующие пункты.

- Увеличить число команд, использующих Scrum
- Расширить применение автоматизации тестов
- Добиться, чтобы команды осуществляли непрерывную интеграцию
- Подумать над тем, как обеспечить наличие у каждой команды владельца продукта
- Подумать над тем, как измерять последствия внедрения Scrum
- Расширить применение автоматизированного тестирования исходного кода и разработки программного обеспечения на основе составления тестов

Журналы совершенствования, подобные представленному в табл. 4.1, являются динамическими: записи в них появляются и исчезают по мере того, как соответствующие идеи зарождаются в головах участников проекта, осуществляются на практике, кажутся ненужными и т.д. Многое из того, что мы обсуждали в главе 2, “ADAPТАция к Scrum”, обязательно найдет свое отражение в журнале совершенствования. Если вы только приступаете к внедрению Scrum, акцент в вашем журнале совершенствования будет сделан на формировании осознания и желания. Если вы уже достаточно далеко зашли в деле внедрения Scrum, то ваш журнал совершенствования может содержать больше записей, касающихся выработки способности добиваться от Scrum как можно большей эффективности, продвижения достигнутых успехов или распространения использования Scrum на другие подразделения компании. Аналогично решения, касающиеся того, какие именно модели внедрения Scrum следует использовать (сведения об этих моделях приведены в главе 3, “Модели внедрения Scrum”), также могут приводить к появлению соответствующих записей в журнале совершенствования.

В случае перехода к использованию Scrum, касающегося небольшого подразделения или какого-либо отдельно взятого проекта, можно обойтись единственным журналом совершенствования. Но когда внедрение Scrum выполняется в крупных масштабах, охватывая большое подразделение или организацию, масштаб действий, связанных с переходом к Scrum, становится столь большим, что уже не обойтись без нескольких журналов совершенствования, каждый из которых создается группой (сообществом) лиц, одержимых идеей совершенствования данной организации тем или иным способом. Может возникнуть, например, сообщество

(и соответствующий журнал совершенствования), которое пытается определить наиболее эффективный способ автоматизированного тестирования применительно к Scrum-проектам. Наряду с этим может возникнуть сообщество (и соответствующий журнал совершенствования), члены которого стремятся стать непревзойденными Scrum-мастерами, и т.п.

Таблица 4.1. Журнал совершенствования представляет собой перечень возможностей, которые предстоит разработать, работ, которые предстоит выполнить, или проблем, которые предстоит решить в соответствующей организации

Запись	Ответственный исполнитель	Примечание
Создать отдел Scrum (наподобие отдела управления проектами), который будет оказывать помощь командам		Джиму (главному инженеру) доложить об этом на очередном ежемесячном совещании по разработке. Посмотрим, проявляют ли люди интерес к этому вопросу
Разработать собственную программу обучения Scrum-мастеров		Как выявлять в своем коллективе подходящих кандидатов? Как повышать их квалификацию?
Накапливать и распространять истории успешного применения Scrum в нашей компании	SC	Интерес к этому вопросу проявила Саванна
Разработать собственную программу непрерывного обучения		Рассмотреть возможность проведения ежеквартальных открытых совещаний. Подыскать отраслевых экспертов и войти в контакт с ними с целью проведения одночасовых встреч в обеденное время
Приступить к проведению обширного автоматизированного тестирования исходного кода (даже если речь не идет о создании тестов до кодирования) и использованию FitNesse		Scrum-команда, которая добьется наибольшего прогресса в этом деле (по результатам всеобщего голосования в подразделении), может в полном составе участвовать в конференции по гибкой методологии разработки, которая состоится следующим летом
Помочь сообществу решить, какого объема предварительной проработки архитектуры будет достаточно	TG	Тод должен был начать переговоры с потенциальными добровольцами, но говорит, что не может ставить перед собой каких-либо конкретных целей до следующего квартала
Уладить спор с техническими службами по поводу перепланировки помещений на втором этаже	JS	Джиму поговорить с Ursulой из технических служб по поводу бюджета, который может потребоваться для этой цели
Составить доклад о том, зачем нам нужно внедрять Scrum; попросить Джима обсудить этот вопрос на очередном ежемесячном совещании у него	JS	Очередное совещание должно состояться 25 марта

Кроме того, когда внедрение Scrum выполняется в крупных масштабах, может возникнуть и такое явление, как головной журнал совершенствования, ведением которого занимается группа, возглавляющая переход к использованию Scrum организаций в целом. Именно о такой группе мы поговорим в следующем разделе.

Сообщество в поддержку перехода к Scrum

Небольшая группа людей, которая инициирует, поощряет и поддерживает усилия организации, направленные на внедрение и совершенствование Scrum, известна как сообщество в поддержку перехода к Scrum (Enterprise Transition Community — ETC).¹ Задачей сообщества в поддержку перехода к Scrum является формирование культуры и среды, в которых необходимые изменения будут осуществляться теми, кто заинтересован в успехе своей организации, причем такой успех повышает энтузиазм участников перехода и расширяет круг его сторонников. ETC достигает этого не путем внедрения соответствующих перемен в организации, а путем направления действий групп, которые осуществляют эти изменения, путем устранения препятствий к эффективному использованию Scrum и путем выработки у людей энтузиазма в отношении осуществляемых перемен.

Члены ETC (как правило, не больше десятка человек) представляют самый высокий организационный уровень, затрагиваемый переходом к Scrum. Если Scrum внедряется в данной организации повсеместно, ETC должно включать старших руководителей из конструкторского или проектного отдела плюс вице-президентов таких подразделений, как отдел управления производством, маркетинговый отдел, отдел сбыта, операционный (учетно-расчетный) отдел, отдел кадров и т.д. В случае внедрения Scrum на уровне подразделения ETC может включать вице-президента конструкторского отдела, а также руководителей службы технического контроля, разработки, архитектуры, проектирования взаимодействия, базы данных и т.д. Главное то, что ETC состоит из высших руководителей того уровня, на котором выполняется переход к Scrum.

Подчас Scrum внедряется в организации стихийно. Какая-то из команд опробует Scrum и успешно выполняет какой-либо проект; этот результат вызывает интерес у других команд — и Scrum начинает свое победное шествие по организации. В таких случаях ETC, как правило, формируется спонтанно кем-то из ранних приверженцев Scrum, которые просят своего начальника предоставить им время для того, чтобы помочь другим командам в освоении Scrum. На каком-то этапе возникают трудности и препятствия, устраниТЬ которые невозможно без помощи со стороны начальника. Таким образом, начальнику не остается ничего другого, как присоединиться к ETC. Если же внедрение Scrum осуществляется в масштабе всего предприятия, ETC, как правило, формируется более целенаправленно после того, как будет принято решение о широком внедрении Scrum.

В качестве примера ETC рассмотрим случай Farm Credit Services of America, кооператива по оказанию финансово-кредитных услуг, который работает с фермерами американского Среднего Запада. В рамках внедрения Scrum Farm Credit Services of America организовал сообщество в поддержку перехода к Scrum, которое получило название “команда сторонников гибкой методологии разработки” (Agile Champions Team — ACT). Членами этого сообщества являются примерно 16 человек. Продолжительность их участия в ACT колеблется от 6 до 24 месяцев и зависит от роли каждого отдельного члена в данной организации и его способности уделять работе в этом сообществе больше или меньше времени. Поскольку переход к использованию Scrum в

¹ Аббревиатура “ETC” соответствует терминологии, предложенной Кеном Швабером в книге *The Enterprise in Scrum*, хотя у Швабера говорится о “команде, ответственной за переход предприятия к Scrum” (Enterprise Transition team) (2007).

Farm Credit Services of America охватывает все информационные службы и экономические подразделения данной организации, члены ACT представляют все функциональные подразделения, которых касается внедрение Scrum. Совещания ACT Farm Credit Services of America проводятся раз в две недели. Каждое совещание длится примерно два часа. Эти совещания дополняются более продолжительными встречами, которые обычно проводятся где-нибудь на “нейтральной территории” и организуются по мере необходимости.

Команда ACT, членами которой являются как формальные, так и неформальные лидеры, часто работает над проблемами во взаимоотношениях между отделом информационных услуг и бизнесом в более широком значении этого слова. ACT удалось решить проблемы, связанные с недостаточным участием в проектах заинтересованных сторон, надлежащим использованием и смыслом сроков завершения проектов, а также с неправильным пониманием руководителями сути гибкой методологии разработки и пользы, которую эта методология может принести компании. Квин Джонс (Quinn Jones) — разработчик программного обеспечения в Farm Credit Services of America, выполнивший в ACT функцию, которую сам он называет шестимесячным “туром долга”. Квин Джонс говорит: “Одной из самых полезных вещей, которые делала команда ACT, является проведение открытых сессий, к участию в которых приглашались все желающие. Участники таких сессий могли задавать любые вопросы и делиться своими знаниями. Эти встречи помогали также выявить коренные проблемы, связанные с гибкой методологией разработки. Решением этих проблем впоследствии занималась команда ACT”.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Составьте предварительный вариант журнала совершенствования, организовав встречу продолжительностью от 30 до 60 минут. Участниками такой встречи могут быть либо члены вашей команды (несколько человек, которых, на ваш взгляд, это может заинтересовать), либо весь отдел. Обсудите методом мозгового штурма то, что вам хотелось бы улучшить. В завершение этой встречи спросите участников, достаточно ли у них энтузиазма для того, чтобы добиться одной или двух таких целей, а затем приступайте к их осуществлению.

Спринты ETC

Поскольку ETC использует Scrum, сообщество в поддержку перехода к Scrum выполняет свою работу спринтами, подобно команде разработчиков, использующих Scrum. Каждый спринт ETC начинается с совещания, посвященного планированию, и завершается обзором и ретроспективой. Эти совещания совершенно аналогичны тем, которые проводятся командами разработчиков, использующими Scrum; зачастую у них возникают те же проблемы. Томас Сефферник (Thomas Seffernick) из KeySorg, крупного финансового учреждения США, участвовал в обзоре первого спринта сообщества ETC своей организации, которое он назвал Agile Enablement Team (команда внедрения гибкой методологии разработки). Он вспоминает, как эта команда совершила ошибку, типичную для многих команд-новичков, выполняющих разработку посредством Scrum: они рассуждали о своих планах, вместо того чтобы демонстрировать достигнутые успехи.

Этот первый обзор спрингта команды внедрения гибкой методологии разработки (ETC) оказался весьма неудачным: руководители вставали и начинали рассказывать о своих планах по устранению проблем, которые они вызвались решить. Вывод совершенно очевиден: планы, конечно, — штука хорошая, но главное все же — достигнутые результаты. С тех пор мы радикально изменили содержание этих обзоров, сделав акцент на достигнутых результатах (2007, 202).

В некоторых ETC проводятся ежедневные совещания разработчиков. Мне кажется, это правильный подход. Правда, в этом случае я не столь категоричен, как в случае команд, выполняющих разработку посредством Scrum. Работы, выполняемые членами ETC, связаны между собой не столь тесно, как в случае команды, выполняющей разработку посредством Scrum. Именно поэтому проведение ежедневных совещаний в случае ETC хоть и полезно, но не критично. Аналогично члены ETC лишь в редких случаях трудятся исключительно в ETC. Основным местом работы большинства членов ETC являются их “родные” отделы, и во многих случаях им целесообразнее проводить рабочее время именно в своих отделах, а не посвящать его ETC. Например, директор по разработке, сохранивший за собой эту должность, способен устраниć больше организационных препятствий, чем директор по разработке, оставилший свою должность, чтобы целиком сосредоточиться на работе в ETC.

Продолжительность спрингта ETC определяется членами ETC. Однако мой собственный опыт показывает, что оптимальная продолжительность спрингта ETC составляет две недели. Именно такую продолжительность спрингта ETC рекомендует и Кен Швабер (Ken Schwaber) (2007, 10). Элизабет Вудворд (Elizabeth Woodward), член сообщества в поддержку перехода к Scrum, которое руководит крупномасштабным внедрением гибкой методологии разработки в IBM, излагает соображения, повлиявшие на выбор продолжительности спрингта в этой компании.

Мы использовали как двух-, так и четырехнедельные спрингты. В конечном счете мы пришли к выводу, что лучших результатов удается добиться в случае двухнедельных спрингтов. Мне кажется, причина заключается в том, что в случае двухнедельных спрингтов “продукт, полученный на выходе”, демонстрирует заметный прогресс. Мы формулируем результаты, достигнутые каждым сообществом, в кратком сообщении — в письме, рассыпаемом по электронной почте, для прочтения которого человеку достаточно примерно пятнадцати минут.

Спонсор и владелец продукта

Самые успешные переходы к Scrum инициировались или приводились в действие определенным спонсором (“опекуном”), в роли которого может выступать кто-либо из старших руководителей в организации, ответственный за осуществление такого перехода. Чрезвычайно успешный крупномасштабный переход к использованию Scrum в Salesforce.com спонсировался одним из основателей этой компании Паркером Харрисом. Занимая пост исполнительного вице-президента по технологиям, Харрис считался ярым сторонником перемен, которые должны радикально изменить способ работы каждого из сотрудников проектного отдела Salesforce.com.

Спонсор перехода должен представлять тот же уровень организации, на котором планируется этот переход. Salesforce.com требовался в качестве спонсора кто-то из высших руководителей, поскольку переход к Scrum планировался в масштабе всего

предприятия. Если вы планируете внедрить Scrum в каком-либо подразделении, подходящим спонсором может быть руководитель этого подразделения.

Спонсор является также владельцем продукта для ЕТС. Это означает, что иногда у ЕТС бывает владелец продукта с малым непосредственным опытом использования Scrum. Это нормально. Подобно всем владельцам продукта, спонсор ЕТС может исполнять эту роль, прибегая к помощи других членов ЕТС. Являясь старшим членом ЕТС, спонсор будет играть важную роль в информировании о процессе внедрения Scrum, но спонсор ЕТС необязательно должен быть единственным источником такого видения.

В ходе внедрения Scrum компания Primavera уяснила на практике важность эффективного спонсора. Вот что пишут о важности поддержки со стороны спонсора Боб Шац (Bob Schatz) и Ибрагим Абдельшрафи (Ibrahim Abdelshafi), в то время — руководители технологического отдела Primavera.

Внедрение гибкой методологии разработки, как и осуществление любых значительных перемен, требует реальной поддержки со стороны руководства компании. Приходится преодолеть немало трудностей, пока ситуация не наладится. Наличие поддержки со стороны руководства позволяет твердо следовать по намеченному пути, несмотря ни на какие трудности и неудачи (2005, 38).

Важно, чтобы спонсор, участвуя в ЕТС, демонстрировал подлинную заинтересованность в успехе перехода к Scrum. Хорошие спонсоры не инициируют переход, заявляют о своей поддержке такого перехода, а затем устраняются от практических действий по внедрению Scrum. Если спонсор не заинтересован по-настоящему во внедрении Scrum, ожидать такой заинтересованности от других также не приходится. Scrum-наставник и автор книги *Collaboration Explained* Джин Табака (Jean Tabaka) считает, что исключительно финансовая заинтересованность спонсора является одной из наиболее вероятных причин неудачного внедрения Scrum: “Внедрение гибкой методологии разработки требует неподдельной и горячей заинтересованности со стороны спонсора, его готовности к радикальным (и зачастую болезненным) организационным переменам, которые способствуют успеху команд, переходящих к использованию гибкой методологии разработки” (2007).

Несмотря на то что членов ЕТС следовало бы по справедливости охарактеризовать как лидеров процесса внедрения Scrum, их лидерство существенно отличается от лидерства в привычном понимании этого слова. Вот как описывает интересующий нас тип лидера Генри Минцберг (Henry Mintzberg) (эксперт по управлению, пользующийся широким признанием в мире) в своей статье, опубликованной в *Harvard Business Review*.

Необходимость как можно более полного учета интересов окружающих требует более умеренной формы лидерства, которую можно назвать заинтересованным и распределенным руководством. Лидер коллектива лично заинтересован в том, чтобы вовлекать и заинтересовывать других, в том, чтобы каждый из них мог проявить инициативу (2009, 141; выделение Минцберга).

Минцберг продолжает свою мысль, утверждая, что в ходе таких организационных перемен, как внедрение Scrum, “мы нуждаемся в строго дозированном лидерстве — лидерстве лишь в той мере, в какой оно совершенно необходимо, и в предоставлении людям возможности делать как можно больше на собственное усмотрение и по собственной инициативе”.

ВОЗРАЖЕНИЕ

“Спонсор нашего проекта перехода к гибкой методологии разработки утверждает, что он заинтересован в таком переходе, но он так и не смог выкроить время, чтобы поучаствовать хотя бы в одном из наших совещаний. Он предоставляет нам все, в чем мы нуждаемся, но отказывается принимать личное участие во внедрении новой методологии”.

У вас, наверное, не самый подходящий спонсор. Несмотря на то что его готовность поддерживать косвенным образом переход к гибкой методологии разработки заслуживает уважения, успешный переход к новой методологии требует непосредственного участия спонсора. Я понимаю, что вам не хотелось бы терять столь могущественного союзника, но мне кажется, что вам все же следует подыскать другого спонсора. В качестве альтернативного варианта можете попытаться уговорить его уделять этому проекту хотя бы немного своего времени. Если это вам удастся, ЕТС может подумать над тем, как лучше распорядиться этим временем. Возможно, спонсору следует участвовать в ваших совещаниях или выступать в качестве публичного сторонника перехода к гибкой методологии разработки на каких-либо других форумах.

Обязанности ЕТС

ЕТС — это рабочая группа, а не руководящий комитет. В ходе планирования спринта ЕТС поручает завершить определенный объем работы и продемонстрировать результат к окончанию соответствующего спринта. Однако еще более важным, чем материальные результаты, получения которых добивается ЕТС, является вызов заинтересованности у других. Сами по себе члены ЕТС могут не так уж много. Чтобы выполнить большую часть работы по внедрению Scrum и переходу к гибкой методологии разработки, им приходится полагаться на других работников компании. Эксперты по управлению изменениями Эдвин Олсон (Edwin Olson) и Гленда Йоянг (Glenda Eoyang) приходят к следующему выводу.

В самоорганизующейся системе лидер играет важную роль, но креативные и долговременные изменения зависят от слаженной работы многих людей на самых разных уровнях и в разных местах организации (2001, 5).

Одной из самых важных задач ЕТС является генерация энергии для внедрения Scrum. Разумеется, эти перемены приведут в восторг далеко не всех, но ЕТС обязано разбудить интерес у тех, кому придется участвовать во внедрении Scrum. Члены ЕТС достигают этой цели, демонстрируя собственный энтузиазм и принимая участие в конструктивном диалоге о происходящих переменах. Чтобы вызвать интерес у других сотрудников организации и таким образом задействовать их в креативных и долговременных изменениях, требуемых для внедрения Scrum, ЕТС обязано решать перечисленные ниже задачи.

- **Формулировать контекст.** Помимо доведения до сознания всех сотрудников организации своего видения будущего этой организации, неразрывно связанного с гибкой методологией разработки, ЕТС должно также помочь сотрудникам уяснить необходимость перемен и выработать у них желание таких перемен. Это достигается путем формулирования контекста перемен: “Зачем это нужно?”,

“Почему это нужно делать именно сейчас?” и “Почему именно Scrum?” Члены ЕТС используют свое положение в организации, свой личный авторитет, чтобы разъяснить эти вопросы остальным сотрудникам организации.

- **Стимулировать диалог.** Понимание невозможно без диалога. В процессе обсуждения достоинств и недостатков тех или иных методов и технических приемов, “историй успеха” и причин неудач, как правило, возникают ценные идеи.
- **Обеспечивать необходимые ресурсы.** Для внедрения Scrum требуются время, усилия и деньги. Например, людям, которые осваивают гибкую методологию разработки (обучаясь, в частности, приемам автоматизированного тестирования исходного кода), может понадобиться дополнительное время на выполнение их проектов. Поскольку в состав ЕТС входят высшие руководители (того уровня, на котором осуществляется внедрение Scrum), их полномочий, как правило, вполне достаточно для выделения дополнительного времени и денег.
- **Ставить реальные цели.** Перемены, в отношении которых сформулированы четкие, по-настоящему трансформационные и вполне конкретные цели, имеют больше шансов на успех (McKinsey & Company, 2008). ЕТС отвечает за постановку и разъяснение целей, связанных с переходом к Scrum, которые могут (и, наверное, должны) меняться с течением времени по мере совершенствования организации. ЕТС может ставить такие цели, как переход от одного ежегодного выпуска к квартальным, 50-процентное снижение количества дефектов и т.п.

См. также В главе 21, “Как далеко вы продвинулись в деле внедрения Scrum”, приведены советы относительно возможных показателей, с помощью которых можно измерять прогресс в деле внедрения Scrum.

- **Вовлекать в процесс преобразований буквально всех сотрудников организации.** У Scrum — длинные щупальца, которые протягиваются в самые отдаленные уголки организации. ЕТС должно позаботиться о том, чтобы внедрение Scrum не ограничивалось какой-то одной группой. В рамках групп, в которых внедряется Scrum, приветствуется как можно более широкое участие.

Дополнительные обязанности

Помимо поощрения людей к участию в переходе к Scrum, ЕТС имеет следующие дополнительные обязанности.

- **Прогнозировать и решать проблемы гуманитарного характера.** ЕТС должно пытаться прогнозировать, какие группы или отдельные лица будут испытывать наибольшие проблемы с переменами, порождаемыми Scrum, и работать с этими людьми в режиме упреждения. В этом отношении весьма полезен многофункциональный состав ЕТС, который помогает людям смотреть на проблемы с разных точек зрения.
- **Прогнозировать и устранять препятствия.** Члены ЕТС отвечают за устранение любых организационных препятствий на пути внедрения или успешного функционирования Scrum. ЕТС должно не просто устранять препятствия, о которых ему становится известно, а пытаться устранивать их еще до того, как они начнут создавать проблемы.

- **Заострять внимание одновременно на практических методах и принципах.** Внедрение Scrum предполагает применение новых технических приемов и следование новым принципам. Организация не может внедрять эти технические приемы, не следя в то же время принципам, положенным в основу этих технических приемов. С другой стороны, она не может внедрять эти принципы, не пользуясь соответствующими техническими приемами. Эффективное ETC стремится выявить дисбалансы между этими двумя составляющими. Если одна из этих составляющих опережает другую, ETC может “подравнять” их, перераспределив обсуждения, внимание и ресурсы в пользу “отстающего”.

Если ETC успешно справляется с этими задачами, сообщество не только будет способствовать самостоятельному движению организации ко все более широкому применению Scrum, но и вызывать интерес и энтузиазм у тех сотрудников, которые еще не охвачены внедрением этой методологии. Чтобы задействовать этот энтузиазм с пользой для дела, люди, стремящиеся усовершенствовать свою организацию определенным образом (например, за счет широкого внедрения автоматизированного тестирования), формируют собственное сообщество, целью которого является такое усовершенствование, а затем выполняют собственные спринты. Такие сообщества известны как *сообщества в поддержку усовершенствования организации* и являются темой следующего раздела.

ВОЗРАЖЕНИЕ

“Организация не поддерживает меня в создании ETC. Не является ли это непреодолимым препятствием на пути внедрения Scrum?”

Нет, не является. Начните с любой сферы влияния, доступной для вас. Начинайте внедрять Scrum в своей команде. Если вы добьетесь успеха, это не останется незамеченным окружающими. Возможно, какая-то другая команда пожелает внедрить у себя Scrum и будет нуждаться в ваших советах. Может быть, вам удастся заинтересовать кого-то из руководителей. Когда в вашей организации появится круг заинтересованных лиц, создайте для начала неформальное сообщество, состоящее из нескольких человек, которые будут время от времени собираться, чтобы обсудить, как идет внедрение Scrum и что можно усовершенствовать в этом отношении. Такой “стихийный” подход весьма эффективен (правда, действует медленнее, чем целенаправленные методы).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Если у вас еще нет ETC или подобной ей группы, выявите несколько человек, которым следовало бы стать членами ETC. Если вы входите в их число, начинайте формировать эту группу. Если вы не входите в их число, поделитесь идеей о ETC и сообществах в поддержку усовершенствования организации с другими сотрудниками своей организации, которые могут помочь вам в формировании таких групп.

Сообщества в поддержку усовершенствований

Сообщество в поддержку усовершенствований (Improvement Community — IC) — это группа лиц, которые объединяются с целью сотрудничества в деле улучшения использования Scrum в своей организации. IC может быть создано, когда люди обращают внимание на какую-то запись в журнале совершенствования ETC и решают совместно работать над достижением соответствующей цели. IC может быть создано и в том случае, когда люди замечают какую-то возможность, связанную с улучшением, которая ускользнула от внимания ETC. В IBM, например, имеется пять IC, предметами внимания которых являются автоматизация тестов, непрерывная интеграция, разработка программного обеспечения на основе составления тестов, роль владельца продукта и использование Scrum как таковой.

ПРИМЕЧАНИЕ

Сообщество в поддержку перехода к Scrum (ETC) и сообщества в поддержку усовершенствования организации (IC), о которых я веду речь, представляют собой специализированные типы того, что принято называть "группами по интересам" (communities of practice; Wenger, McDermott, and Synder, 2002). Группа по интересам — это группа единомышленников или людей одной профессии, добровольно объединившихся вследствие своей увлеченности определенной технологией, подходом или общим для них представлением об окружающей действительности. В этой книге нам встретятся и другие "группы по интересам". Подробное их обсуждение приведено в главе 17, "Изменение масштаба Scrum".

Взаимосвязь между одним ETC определенной организации и множеством ее IC представлена графически на рис. 4.1. ETC обеспечивает процесс перехода, но не направляет его и не управляет им. В значительной мере роль ETC сводится к формированию среды, в которой образуются и органически растворяются IC, стремящиеся усовершенствовать способ создания продуктов в данной организации.

Степень охвата таким подходом следует либо увеличить, либо уменьшить в зависимости от величины организации, осуществляющей этот переход. В отделе разработки программного обеспечения, насчитывающем 30 человек, вполне достаточно было бы ETC, включающего пять человек. В случае проектного подразделения, насчитывающего 200 разработчиков и внедряющего Scrum в масштабе всей компании, можно сформировать ETC, насчитывающее 10 человек (включая представителей не только проектного подразделения, но и других групп), плюс несколько сообществ в поддержку усовершенствований в каждый момент времени. Численность каждого такого сообщества в поддержку усовершенствований может колебаться в очень широких пределах (например, некоторые из сообществ в поддержку усовершенствования в IBM насчитывают свыше 800 членов).

Большинство членов IC тратят на участие в делах сообщества лишь незначительную часть своего рабочего времени. Они могут читать сообщения, помещенные в перечень обсуждений, добавлять комментарии и этим и ограничиваться. Количество времени, затрачиваемого членом IC на участие в делах сообщества, определяется самостоятельно каждым работником, его начальником или организационной культурой.

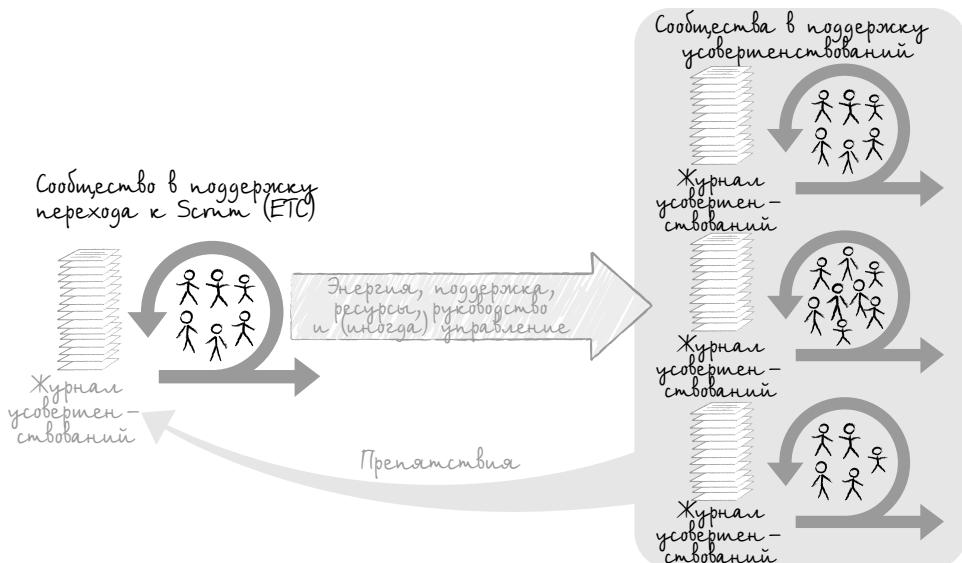


Рис. 4.1. Сообщество в поддержку перехода к Scrum (ETC) руководит внедрением Scrum, но большая часть работы выполняется множеством сообществ в поддержку усовершенствований организации

ВОЗРАЖЕНИЕ

“Считается, что Scrum-команды — самоорганизующиеся коллективы. Не вступает ли ETC в противоречие с таким определением? Не следуют ли командам определиться с тем, что именно они хотят усовершенствовать?”

Самоорганизация является естественной реакцией на проблему, которую собирается решить группа лиц. В случае какого-либо проекта, связанного с разработкой, компания может обратиться к одной из команд со следующим предложением: “Разработайте эту программу так, чтобы она работала быстрее и занимала меньше места в оперативной памяти, чем ее нынешняя версия, и выполните эту работу на два месяца быстрее, чем вы сделали это в прошлый раз”. Затем эти люди самоорганизуются для достижения поставленной цели. То же самое происходит и в случае ETC. ETC формулирует цель, связанную с усовершенствованием чего-либо (при этом вовсе необязательно указывать, как именно добиться такого усовершенствования). Ответ на вопрос “Как именно?” должны давать либо сообщества в поддержку усовершенствований, либо Scrum-команды.

Кроме того, не нужно забывать, что главной целью ETC является формирование такой среды, чтобы сообщества в поддержку усовершенствований организации сами выявляли для себя цели и спонтанно самоорганизовывались для их достижения. Подробнее о самоорганизации мы поговорим в главе 12, “Управление самоорганизующимся коллективом”.

Катализаторы усовершенствований

Сообщества, когда их деятельность используется как составная часть деятельности по внедрению и совершенствованию использования Scrum, становятся катализаторами усовершенствований. Рассмотрим случай Google, где сообщества в поддержку усовершенствований называются “групплетами” (grouplets). Группплет тестирования был сформирован в Google “с целью ускорения внедрения тестирования, выполняемого самим разработчиком” (Striebeck, 2007). Бхарат Медиратта (Bharat Mediratta), который сформировал это сообщество, так описывает его деятельность.

Мы начали с того, что раз в две недели собирали инженеров со всей компании для проведения сеансов мозгового штурма. Мало-помалу мы начали превращаться в активистов, планирующих конкретные действия по усовершенствованию автоматизированного тестирования исходного кода. Мы начали создавать более совершенные инструменты и проводить неформальные обсуждения с различными техническими группами (Mediratta, 2007).

Обратите внимание: несмотря на то что это сообщество собиралось поначалу для проведения сеансов мозгового штурма, вскоре они почувствовали себя активистами, вынашивающими планы реальных усовершенствований. Сообщества в поддержку усовершенствований — это сообщества действия. Вот почему их не называют специальными комиссиями по изучению того или иного вопроса, рабочими группами, комитетами или какими-либо иными словами, которые слишком часто вызывают у людей ассоциации с неэффективными группами. Если бы группплет тестирования в Google занимался лишь подготовкой и проведением презентаций, посвященных достоинствам тестирования, выполняемого самим разработчиком, или решил бы убеждать кого-то из влиятельных вице-президентов в необходимости принудительного внедрения такого тестирования, его деятельность оказалась бы безрезультатной.

Вместо этого сообщество тестирования в Google решило изыскать непосредственные способы оказания помощи командам. Бхарат Медиратта вспоминает, как, помимо создания инструментов, сообщество изыскало уникальный способ предоставления конкретных, кратких примеров и советов, касающихся тестирования.

Однажды, ближе к окончанию одного продолжительного сеанса мозгового штурма, кому-то из нас пришла в голову идея вывешивать в туалетных кабинках короткие (на одну страницу) истории, которые получили у нас название эпизодов, в которых обсуждались бы новые интересные методы тестирования. Кто-то тотчас же назвал это “тестированием на стенах туалета”. Эта идея сразу же захватила нас (2007).

Самыми эффективными сообществами обычно оказываются те, которые создаются не в ответ на требования руководства, а вследствие самой культуры компании или вследствие того, что ЕТС удалось создать такую среду, в которой сообщества возникают как бы сами по себе. Дж. Ф. Ансон (J.F. Unson), работавший наставником в Yahoo! во время крупномасштабного внедрения Scrum в этой компании, утверждает, что именно так и случилось в одном из отдаленных офисов Yahoo!.

В Yahoo! (в нашем кампусе в Санта-Монике) все, кого заинтересовала гибкая методология разработки, организовали проведение ежемесячной встречи Scrum-

мастеров. Это произошло как бы само собой, без какого-либо принуждения со стороны ETC (2008).

Разумеется, не все сообщества образуются таким органичным, стихийным способом. ETC придется стимулировать создание сообществ в поддержку усовершенствований (особенно в течение первых недель или месяцев внедрения Scrum), подчеркивая важность той или иной цели и уповая на то, что вокруг этой цели сплотится группа энтузиастов (т.е. сформируется соответствующее сообщество). Подчас ETC приходится просить кого-либо сформировать сообщество, которое должно сплотиться вокруг определенной цели.

Два показателя эффективности

Профессор Джейфри Голдстейн (Jeffrey Goldstein) писал: “К переменам не нужно принуждать, им просто нужно дать возможность совершиться” (1994, 32). Оценить, насколько успешно ETC справляется с предоставлением возможности совершиться переменам, можно с помощью двух показателей.

1. Количество сообществ в поддержку усовершенствования организации, сформировавшихся без непосредственного “подталкивания” со стороны ETC
2. Доля таких сообществ в общем количестве сообществ в поддержку усовершенствований

Если количество спонтанно сформировавшихся сообществ в поддержку усовершенствований высоко, и особенно если они составляют большинство в совокупном количестве сообществ, то это указывает на сильную заинтересованность в Scrum и в переменах, к которым приводит внедрение этой новой методологии. Если эти показатели увеличиваются или хотя бы остаются со временем достаточно высокими, можно сделать вывод, что соответствующая организация успешно продвигается к освоению гибкой методологии разработки. Следует, конечно, учитывать и другие показатели. Но лично мне нравятся именно эти два.

Спринт сообщества в поддержку усовершенствований

Как и можно было ожидать, IC также выполняют свою работу спринтами. Как и в случае ETC, каждое IC может выбрать наиболее подходящую для себя длительность спринта, но, вообще говоря, рекомендуются спринты продолжительностью две недели. IC, которое сформировалось стихийно, как правило, исполняет роль владельца собственного продукта, причем члены сообщества вольны посвящать свое время именно тем усовершенствованиям, которые вызывают у них наибольшую заинтересованность. С другой стороны, IC, которое сформировалось в ответ на ту или иную цель, указанную ETC, как правило, планирует свои спринты в сотрудничестве с кем-либо из членов ETC, выполняющим функции владельца продукта.

С учетом сказанного сообщество в поддержку усовершенствований не занимается обслуживанием ETC. IC занимается обслуживанием своих клиентов — команд разработчиков, создающих продукты или системы с помощью методологии Scrum. Несмотря на то что кто-то из членов ETC будет выполнять роль владельца продукта по отношению к тем или иным сообществам в поддержку усовершенствований и будет служить официальным владельцем продукта при проведении обзоров спринтов,

следует исходить из того, что активными участниками будут также члены заинтересованных команд разработчиков. Кроме того, дальновидное ЕТС понимает, что наилучшие результаты будут достигнуты в случае, когда сообществам в поддержку усовершенствований будут предоставлены достаточно широкие полномочия в деле достижения их целей. На практике это означает, что ИС, даже если оно сформировано в ответ на ту или иную цель, указанную ЕТС, будет нести приоритетную ответственность за свою работу, в то же время не забывая о потребностях организации, связанных с ее усовершенствованием конкретными способами, и о стремлении ее членов работать над решением этих проблем.

В ходе совещания, посвященного планированию спринта, каждое сообщество в поддержку усовершенствований выбирает проблему, которую оно намерено решить к окончанию данного спринта. Если сообщество в поддержку усовершенствований было сформировано в ответ на ту или иную цель, указанную ЕТС, то планирование спринта начинается с выбора соответствующей записи (проблемы) в журнале ЕТС и ее разбиения на более мелкие задачи, которые будут занесены в журнал усовершенствований соответствующего ИС. Лучше всего это можно пояснить на примере.

Журнал усовершенствования ЕТС, представленный в табл. 4.1, включает пункт “Разработать собственную программу обучения Scrum-мастеров”. Сообщество в поддержку усовершенствований сформировано через месяц после того, как ЕТС поместило соответствующую запись в журнал усовершенствований и оповестило остальных сотрудников компании о желательности разработки такой программы. Поначалу в этом сообществе было три человека, но этого было вполне достаточно, чтобы добиться значительного прогресса в достижении указанной цели. В ходе своего первого совещания, посвященного планированию спринта, они обсудили цель ЕТС (“Разработать собственную программу обучения Scrum-мастеров”) и создали собственный журнал усовершенствований, отразив в нем все, что должно быть сделано ими для достижения указанной цели. Этот журнал представлен в табл. 4.2.

Таблица 4.2. Журнал сообщества в поддержку усовершенствований для разработки собственной программы обучения Scrum-мастеров

Что сделать	Примечание
Выяснить, как определять подходящих кандидатов на получение квалификации Scrum-мастера (дополнение к тем, кто сам изъявил желание участвовать в этой программе)	
Разработать собственную программу наставничества	
Составить программу обучения в компании. Какие курсы должны войти в эту программу? Кто должен вести эти курсы? Составить программу обучения самостоятельно или у нас есть возможность закупить готовый учебный материал?	
Определить, какие курсы мы можем вести собственными силами	
Составить бюджет обучения силами сторонних преподавателей на следующий год. Сколько дней? По какой суточной ставке предположительно может осуществляться обучение?	Джеймс уже выяснял ставки у трех преподавателей
Выяснить, что мы могли бы сделать в сотрудничестве с местными группами пользователей в плане разделения затрат на приглашение лекторов	У Саванны есть контакт с одной местной группой, которая уже приступила к внедрению Scrum

Кроме того, в ходе планирования спринта члены сообщества, рассмотрев некоторые из пунктов табл. 4.2, определили задачи, необходимые для выполнения каждого из этих пунктов. Например, для последнего пункта табл. 4.2 (работа с местными группами пользователей по разделению затрат на приглашение лекторов) сообщество определило следующие задачи.

- Путем веб-поиска определить, какие группы пользователей находятся в нашей области
- Составить смету расходов
- По внутренним почтовым рассылкам отправить сообщения, чтобы определить, связан ли с этими группами кто-либо из лиц, получающих рассылки
- Позвонив каждому из них по телефону, представиться и рассказать, чем мы занимаемся
- Выяснить, не разделяли ли ранее какие-либо из групп расходы на приглашение лектора в город с какой-нибудь другой компанией; выяснить, не желает ли кто-либо из них сотрудничать с нами в этом вопросе
- Встретиться с Сьюзен для обсуждения и утверждения сметы

Как и в случае совещания команды разработчиков, посвященного планированию спринта, сообщество оценило каждый из пунктов и решило, что эти задачи можно решить в течение планируемого спринта. Спустя две недели в ходе обзора спринта эта команда показала своему владельцу продукта (являющемуся членом ЕТС) список местных групп пользователей и план работы с одной из них дважды в году, разделяя с ними расходы на приглашение лекторов, пользующихся заслуженным авторитетом.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Добавьте в свой журнал усовершенствований записи, взяв за основу заголовки разделов глав этой книги. Многие из них были придуманы мною именно в расчете на такую возможность.
- Рассмотрите любые примечания, которые можно найти в ретроспективах последних спринтов. Они зачастую являются превосходным источником записей в журнале усовершенствований.

Сосредоточьтесь на целях, имеющих непосредственное отношение к практике

Чтобы сообщество в поддержку усовершенствований было максимально влиятельным, его члены должны сосредоточиться на целях, имеющих самое непосредственное отношение к практике, т.е. наибольшее практическое значение для команд разработчиков, использующих или планирующих приступить к использованию Scrum. Добиться этого будет проще всего, если члены сообщества в поддержку усовершенствования организации будут работать бок о бок с членами команды разработчиков над чем-то важным для этой команды разработчиков. Именно так и поступают “тестовые наемники” в Google. Тестовые наемники — это члены сообщества тестирования, которые являются опытными инженерами, имеющими ярко выраженную склонность к

тестированию и владеющие обширными знаниями в области тестирования. На протяжении трех месяцев они тратят до 20% своего времени на какой-либо “посторонний” (т.е. не их собственный) проект. В течение этого времени они добавляют тесты и выполняют рефакторинг кода, оказывая таким образом непосредственную помощь соответствующей команде разработчиков.

Я полагаю, что вместо этого тестовые наемники могли бы тратить свое время на создание презентаций и популяризацию тестирования, выполняемого разработчиками. Однако что-то подсказывает мне, что им будет намного легче достичь своих целей, сотрудничая с командой разработчиков, а не занимаясь проповедью. Команда разработчиков, которая может воспользоваться помощью тестовых наемников, получает возможность создавать более совершенный код и выполнять большее число тестов. Кроме того, она получает возможность сосредоточиться на тестировании, выполняемом разработчиками, и воспользоваться преимуществами такого тестирования. Это создает дополнительные (и весьма существенные) мотивации для тех членов команды разработчиков, пользующихся Scrum, которые желают продолжить деятельность в этом направлении после переключения тестовых наемников на другую команду разработчиков.

Кроме того, концентрация на оказании практической помощи командам разработчиков удерживает членов сообществ в поддержку усовершенствований от соблазна “читать проповеди” командам разработчиков. Типичная проблема, связанная с внедрением Scrum, заключается в том, что “первоходцы” зачастую становятся настоящими фанатиками, пытающимися обратить в свою веру всех остальных. Между тем эти фанатики нередко забывают о том, что адаптация людей к Scrum и переменам, которые влечет за собой эта методология, требует определенного времени. Когда другим не удается мгновенно “перековаться”, фанатики зачастую воспринимают такую задержку как сопротивление переменам. Поскольку такой фанатизм нередко приносит больше вреда, чем пользы, членам сообществ в поддержку усовершенствований важно помнить, что они должны выступать в роли консультантов, а не проповедников (Allen-Meyer, 2000c, 25).

Члены сообщества в поддержку усовершенствований

Эксперт по организационным переменам Гленн Аллен-Мейер говорит, что перемены должны осуществляться при поддержке как можно большего количества людей (2000b). Именно поэтому так важно, чтобы членом сообщества становился каждый, кто искренне заинтересован в усовершенствовании своей организации намеченным способом. Таким образом, членство в сообществе не должно ограничиваться лишь высшими руководителями организации. Широкое участие в сообществах в поддержку усовершенствования организации помогает каждому работнику этой организации почувствовать, что перемены осуществляются *в сотрудничестве* с ними, а не по отношению к ним. Количество людей, участвующих в любом из сообществ в поддержку усовершенствований, не должно ограничиваться. Число членов сообщества зачастую намного превышает сто, причем уровни индивидуального участия со временем изменяются в зависимости от нагрузки, обусловленной основными должностными обязанностями каждого сотрудника.

Участие сотрудника в том или ином сообществе не означает, что он посвящает работе в этом сообществе все свое время; это скорее дополнительная нагрузка к его непосредственным должностным обязанностям. В IBM лидерам сообществ в поддержку

усовершенствования организации предлагается уделять работе в своем сообществе два часа в неделю, хотя многие посвящают этому гораздо больше времени, стремясь как можно скорее добиться ощутимого прогресса. Однако менеджер участника, владелец продукта и руководитель Scrum-проекта обязаны позаботиться о том, чтобы подлинным энтузиастам перемен, в поддержку которых создано данное сообщество, было предоставлено достаточно времени для работы над целями, которые поставило перед собой их сообщество. В Google эту проблему решили, предоставив каждому работнику возможность тратить еженедельно пятую часть своего рабочего времени на занятие, представляющее для данного работника наибольший интерес. Это время можно потратить, например, на исследование идеи, касающейся какого-либо нового продукта, или на участие в деятельности сообщества.

Компания Salesforce.com, успешно внедрившая у себя Scrum, использует не менее новаторский подход под названием PTON (произносится как “пи-ти-он”). Эта аббревиатура означает “оплачиваться должно все, чем занимается человек в рабочее время”. Программа PTON компании Salesforce.com — названная по ассоциации с политикой многих компаний, известной под аббревиатурой PTO (произносится как “пи-ти-оу”), которая означает “оплачивается только исполнение непосредственных должностных обязанностей”, — позволяет человеку заниматься в рабочее время реализацией инициатив по его собственному выбору. В соответствии с программой PTON каждому работнику ежегодно предоставляется возможность тратить на реализацию подобных инициатив одну неделю своего рабочего времени. Работники компании Salesforce.com могут использовать время PTON для работы в сообществах, исследования идей, касающихся каких-либо новых продуктов, или выполнять иную работу по собственному усмотрению.

Политика “20% рабочего времени”, предусмотренная в Google, и программа PTON в Salesforce.com изначально не предполагали предоставления людям возможности работать в каком-либо из сообществ в поддержку усовершенствования организации. Вообще говоря, организациям нет нужды идти на столь решительные меры лишь для того, чтобы приступить к внедрению Scrum. Руководству компаний достаточно лишь высвободить несколько часов в неделю для тех, кто желает работать в каком-либо из сообществ в поддержку усовершенствования организации.

ВОЗРАЖЕНИЕ

“Мы уже целый год работаем над этим новым продуктом. Мы должны передать его заказчику через четыре недели. Мне как владельцу продукта нужно, чтобы в течение этих четырех недель мои люди полностью сосредоточились на выполнении этого задания”.

Именно так и должно быть! Членам команды, наверное, это уже известно. Они, наверное, понимают, что на эти четыре недели им придется (полностью или частично) отказаться от работы в каких бы то ни было сообществах. Члены команды, которым предоставлялась возможность реализовать те или иные долгосрочные инициативы в своих сообществах и которые вместе с тем понимают, насколько важен их вклад в своевременное завершение данного проекта, с готовностью минимизируют в течение этих четырех недель свое участие в деятельности этих сообществ. Впоследствии они наверняка смогут наверстать время, которое им пришлось уделить выполнению своих непосредственных должностных обязанностей.

ВОЗРАЖЕНИЕ

“По-моему, все эти сообщества в поддержку усовершенствования организации очень смахивают на группы совершенствования процесса разработки программного обеспечения (Software Engineering Process Groups — SEPG), созданные в нашей компании для осуществления CMMI (Capability Maturity Model Integration — набор моделей [методологий] совершенствования процессов в организациях разных размеров и видов деятельности). Не является ли это просто новым названием старой идеи?”

Не совсем так, но я понимаю, почему вам пришла такая мысль. И IC, и SEPG призваны помочь организации усовершенствовать способ разработки программного обеспечения в организации. Однако несмотря на схожесть целей IC и SEPG принципиально различаются между собой.

- SEPG рассматривает процесс разработки программного обеспечения и пытается ответить на вопрос “Как можно усовершенствовать этот процесс?” Члены сообщества в поддержку усовершенствования организации рассматривают свои проекты и пытаются ответить на вопросы “Что мы могли бы усовершенствовать?” и “Каким своим положительным опытом нам следовало бы поделиться с другими?”
- Некоторые SEPG принуждают работников следовать определенному процессу; сообщества в поддержку усовершенствования организации не наделены правом принуждать других к соблюдению каких-либо процедур или правил.
- Некоторые SEPG обязаны анализировать лишь какие-то части процесса разработки в целом. В то же время IC при изыскании возможностей для улучшения не следует ограничиваться лишь процессом разработки продукта.
- Сообщества в поддержку усовершенствования организации являются самомотивирующими и самоорганизующими коллективами. Обычно никого не принуждают к вступлению в то или иное сообщество. (Правда, иногда к такой мере прибегают, чтобы инициировать создание нового сообщества.)
- В деле совершенствования процессов члены сообщества в поддержку усовершенствования организации в большей степени склонны к экспериментальному подходу (“изменим что-нибудь и посмотрим, что из этого выйдет”).
- Сообщества в поддержку усовершенствования организации по своей природе спонтанны и органичны. Они формируются, когда людей сближает стремление к какой-то общей цели. SEPG создаются формализованно, и их спонтанное функционирование не поощряется.

Распуск сообщества

Большинство сообществ со временем распускается. Например, сообщество, сформированное, чтобы способствовать распространению автоматизированного тестирования, может существовать годами (при этом состав его может постоянно обновляться), пока соответствующая организация нуждается в как можно более широком внедрении и совершенствовании автоматизированного тестирования. Рано или поздно автоматизированное тестирование в этой организации выйдет на такой уровень, при котором члены данного сообщества смогут приносить своей организации больше пользы, вливвшись в какие-то другие сообщества в поддержку усовершенствований.

Что же касается ЕТС, то его следует распустить после того, как организация внедрит у себя Scrum и перейдет в фазу постоянного совершенствования. ЕТС существует только на время переходного периода, который в случае крупномасштабного перехода может длиться несколько лет.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Выберите усовершенствование, которое вызывает у вас особый интерес. Попросите двух-трех своих коллег помочь вам. Создайте журнал усовершенствований и спланируйте первый спринт. Даже если вы можете посвятить этому лишь один час в неделю, все равно приступайте к делу. Когда вы добьетесь некоторого прогресса, внедрите свои усовершенствования в работу своей команды или предложите их какой-то другой команде. Попытайтесь вызвать у людей интерес, рассказав им о своих достижениях или, возможно, продемонстрировав их.

Универсального рецепта не существует

В этой главе я изложил подход к внедрению Scrum, основанный на использовании сообществ. Руководящее сообщество — сообщество в поддержку перехода к Scrum — выполняет определенную работу по осуществлению такого перехода, но самое важное заключается в том, что ЕТС создает среду, способствующую формированию других сообществ. Эти сообщества — которые носят название сообществ в поддержку усовершенствований — формируются, когда некоторая группа сотрудников решает совместно работать над совершенствованием способа использования Scrum в своей организации. Оба типа сообществ используют методологию Scrum для скорейшего внедрения гибкой методологии разработки в своей организации.

Однако универсального рецепта, который годился бы на все случаи жизни, не существует. Подход, который я описывал в этой главе, приносит положительные результаты при внедрении Scrum средними и крупными подразделениями. Для подразделений меньшего размера нужно вносить соответствующие корректизы. Например, отдел разработки программного обеспечения в составе 20 специалистов может располагать одной группой энтузиастов гибкой методологии разработки, которые будут служить локомотивом перемен и усовершенствований в своем отделе. Эта группа будет выступать в роли ЕТС и ИС в одном лице.

Что дальше

В главах, которые составляют вводную часть этой книги, мы обсуждали причины трудностей, связанных с переходом к Scrum. Вместе с тем мы указывали на то, что, преодолев эти трудности, вы не пожалеете о затраченном времени. Мы говорили о действиях, которые сопутствуют переменам, и инструментах, которыми можно воспользоваться, чтобы помочь людям перейти к Scrum. Мы обсуждали модели, которыми можно руководствоваться в своем подходе к внедрению Scrum. Наконец, мы рассмотрели, как соединить всю эту информацию и сам Scrum-процесс и использовать полученные сведения для управления внедрением Scrum, каким бы ни был размер

подразделения, в котором планируется внедрить Scrum. В главах 1–4 этой книги я неоднократно указывал, что, в отличие от инициатив, связанных с какими-либо другими переменами, мы не можем говорить об “окончательном” завершении процесса внедрения Scrum. Иными словами, у процесса внедрения Scrum не может быть конечного состояния. Напротив, использование Scrum требует постоянного совершенствования, которое обеспечивается посредством сообществ в поддержку усовершенствования организации, также использующих в своей работе методологию Scrum.

В следующей главе мы обсудим способы выбора первого проекта и первой команды и рассмотрим, как приступить к использованию гибкой методологии разработки под названием Scrum.

Дополнительная литература

Conner, Daryl R. 1993. *Managing at the speed of change: How resilient managers succeed and prosper where others fail*. Random House.

В этой книге описываются восемь важнейших моделей поведения людей во время организационных перемен. Одной из целей предлагаемого процесса управления изменениями является воспитание у людей и организаций гибкости и устойчивости к внешним воздействиям. Точка зрения автора на такую гибкость и устойчивость совместима с представленным в данной книге пониманием изменений как непрерывного процесса, и гибкой методологии разработки как чего-то такого, к чему нужно приближаться постепенно, шаг за шагом.

Katzenbach, Jon. R. 1997. *Real change leaders: How you can create growth and high performance at your company*. Three Rivers Press.

Эта книга основана на многочисленных интервью с людьми, которые показались автору истинными инициаторами перемен в организациях. Таким образом, в полном соответствии с названием этой книги речь в ней идет о “подлинных лидерах перемен”. Книга содержит много увлекательных историй о людях, которые могли бы стать незаменимыми членами сообществ в поддержку усовершенствований.

Kotter, John P. 1996. *Leading change*. Harvard Business School Press.

Весьма авторитетная книга Коттера является классическим трудом по организационным переменам. Автор описывает процесс реализации изменений, состоящий из восьми этапов. Что касается второго этапа описываемого процесса, то Коттер отстаивает необходимость создания руководящей коалиции, во многих отношениях напоминающей ЕТС. Кроме того, в статье Коттера, опубликованной в *Harvard Business Review* (1995), излагается краткое содержание этой книги.

Schwaber, Ken. 2007. *The enterprise and scrum*. Microsoft Press.

В этой книге Швабер (один из тех, кому принадлежит идея Scrum) описывает, что именно необходимо для перехода всей организации к использованию Scrum. Автор дает советы по ведению журнала усовершенствований и по формированию команды для перехода предприятия к Scrum, подобной сообществу в поддержку перехода к Scrum, о котором говорится в моей книге.

Wenger, Etienne, Richard McDermott, and William M. Snyder. 2002. *Cultivating communities of practice*. Harvard Business School Press.

Венгер является признанным авторитетом в вопросах создания и функционирования сообществ практических пользователей Scrum. В этой чрезвычайно популярной книге рассказывается обо всем, что нужно знать, чтобы приступить к формированию сообществ в организации. В частности, в ней есть глава, в которой приводятся советы для координаторов сообществ.

Woodward, E. V., R. Bowers, V. Thio, K. Johnson, M. Srihari, and C. J. Bracht. Forthcoming. Agile methods for software practice transformation. *IBM Journal of Research and Development* 54 (2).

Члены организации Quality Software Engineering в компании IBM пользуются для распространения гибкой методологии разработки подходом, очень похожим на тот, который описан в данной главе. В этой превосходно написанной статье показано, как они работают в качестве сообщества в поддержку перехода к Scrum (ETC) и описаны способы стимуляции формирования сообществ в поддержку усовершенствований и использование схемы Scrum для усовершенствования методов ее применения.

Глава 5

Ваши первые проекты

Если только вы не действуете в скрытом режиме, глаза всех окружающих будут прикованы к первому проекту, который реализуется с помощью методологии Scrum. Особенно повышенным это внимание будет во время первых спринтов. Вот почему так важно правильно выбрать первый проект, который будет выполняться с помощью Scrum. Ваш первый Scrum-проект должен быть значимым и важным для вашей организации, чтобы результаты выполнения этого проекта невозможно было проигнорировать. Вместе с тем он не должен быть настолько большим, чтобы в процессе его выполнения возникли специфические трудности, обусловленные именно объемом проекта. Членов команды следует подбирать таким образом, чтобы они не только отвечали критерию взаимной совместимости, но и стремились опробовать что-то новое.

См. также В главе 3, “Модели внедрения Scrum”, читатели познакомились со скрытым вариантом перехода к использованию Scrum.

Когда начнется выполнение первого спринта, ожидания, связанные с высокой эффективностью Scrum, могут оказаться заоблачными. Иногда это является следствием всеобщего оптимизма, а иногда обусловлено непомерным энтузиазмом первоходцев гибкой методологии разработки в данной организации. Такой энтузиазм заставляет других думать, что Scrum является настоящей панацеей от всех бед. Вот почему очень важно умело управлять такими ожиданиями и вовремя их корректировать. В противном случае первоначальный проект, в безусловном успехе которого вы кровно заинтересованы, может быть воспринят, как оглушительный провал, если окажется, что он не отвечает завышенным ожиданиям.

В настоящей главе мы рассмотрим такие важные вопросы, как оптимальный выбор первого проекта и подбор идеальной команды. Кроме того, мы поговорим о тонком искусстве формирования у людей реалистичных ожиданий.

Выбор пилотного проекта

Я собирался начать этот раздел примерно с такой сентенции: “Последние четыре года пилотные Scrum-проекты становятся все большей и большей редкостью. Выгоды использования Scrum стали настолько очевидными, что компании все чаще предпочитают отказываться от реализации пилотных проектов и сразу же берут быка за рога”. Но затем я решил еще раз уточнить определение *пилотного проекта*. Возможно, я не совсем правильно понимаю, что такое пилотный проект? В результате моих изысканий оказалось, что существуют два несколько отличающихся друг от друга определения пилотного проекта. Согласно одному из них пилотный проект представляет собой тест, результаты которого используются для выяснения необходимости проведения дальнейшего тестирования. Именно такого типа пилотных проектов сейчас стремится избегать большинство компаний: они твердо знают, что хотят использовать Scrum, и не нуждаются в выполнении пилотных проектов, которые подтверждают правильность такого выбора.

Другое найденное мною определение заключается в том, что пилотный проект выполняется лишь для того, чтобы понять, *как* нужно выполнять последующие проекты. Иными словами, реализация пилотного проекта позволяет нам лучше уяснить новый способ выполнения проектов. Лично меня интересует именно второе определение пилотного проекта: пилотный проект — это не тест, а инструмент, который позволяет лучше уяснить дальнейший способ действий. В нашем распоряжении уже есть множество свидетельств высокой эффективности Scrum. Каждая отдельная организация, приступающая к внедрению Scrum, желает понять, как использовать Scrum в своих конкретных условиях. Таким образом, организации выполняют один или более пилотных проектов как *обучающие* проекты.

Четыре характеристики идеального пилотного проекта

Правильный выбор пилотного проекта может оказаться не такой уж простой задачей. Джек Хониус (Jeff Honious), вице-президент по инновациям в Reed Elsevier, возглавлял переход своей компании к использованию Scrum. Он вместе со своим коллегой Джонатаном Кларком (Jonathan Clark) пишет о трудностях, с которыми они столкнулись при выборе подходящего пилотного проекта.

Выбор подходящего пилотного проекта оказался самой ответственной и сложной задачей. Нам требовался весьма содержательный проект, который не воспринимался бы людьми как специально подобранный, “особый” случай. Вместе с тем нам не нужен был и проект, который охватывал бы все возможные проблемы, — слишком уж многое зависело от успеха этого проекта (2004).

ПРИМЕЧАНИЕ

Я, конечно, помню о том, насколько успех пилотного проекта зависит от его исполнителей. Эта тема обсуждается ниже, в разделе “Выбор пилотной команды”.

Далеко не каждый проект можно выбрать в качестве первого, пилотного проекта. Идеальный пилотный проект определяется идеальным сочетанием величины

(масштаба) проекта, продолжительности его выполнения, его важности и степени участия бизнес-спонсора, как показано на рис. 5.1. Может оказаться, что вы не в состоянии выбрать “идеальный” пилотный проект. И это нормально. Рассмотрите все проекты, имеющиеся в вашем распоряжении, и постарайтесь обеспечить разумный баланс между четырьмя факторами, представленными на рис. 5.1. Значительно лучше выбрать проект, близкий к идеальному, и приступить к его реализации, чем ожидать несколько месяцев появления идеального пилотного проекта.

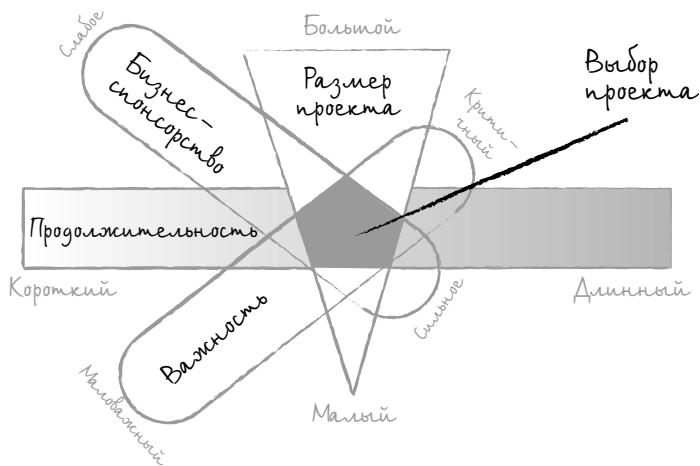


Рис. 5.1. Четыре характеристики идеального пилотного проекта

Продолжительность выполнения. Если вы выберете непродолжительный по времени выполнения проект, скептики будут заявлять, что Scrum можно использовать только для коротких проектов. С другой стороны, если вы выберете слишком продолжительный проект, вам придется очень долго ждать его завершения (ведь только после завершения проекта вы сможете заявить о своем успехе). Многие проекты, рассчитанные на срок от 9 до 12 месяцев и выполняемые традиционным способом, в конечном счете не укладываются ни в бюджет, ни в запланированные сроки. Если то же самое случится с вашим первым Scrum-проектом, вряд ли это убедит кого-то в целесообразности перехода к использованию Scrum.

Мне кажется, что лучше всего выбрать проект, продолжительность выполнения которого равняется приблизительно средней (“нормальной”) продолжительности выполнения проектов в вашей организации. В идеале (и чаще всего) эта продолжительность составляет три-четыре месяца. Такой выбор предоставляет команде достаточно времени, чтобы, не перенапрягаясь (и даже получая определенное удовольствие), приспособиться к выполнению готовых частей работы в ритме спринтов, уяснить преимущества такого способа организации работы для самих себя и для пользы дела. Проекта, рассчитанного на три-четыре месяца, обычно достаточно для того, чтобы заявить, что использование Scrum приведет к аналогичному успеху и в случае более продолжительных проектов.

Размер проекта. Выберите проект, выполнение которого можно начать силами одной команды. Желательно, чтобы все члены этой команды были сосредоточены на одной “рабочей площадке”. Начните с одной команды, даже если впоследствии

К выполнению пилотного проекта придется подключать другие команды. Постарайтесь выбрать пилотный проект, который не разрастется до такого масштаба, что его придется выполнять силами более чем пяти-семи команд (даже если такие проекты типичны для вашей организации). Дело не только в том, что координаирование работы слишком большого числа команд — это роскошь, которую нельзя позволить себе при выполнении пилотного проекта. Дело еще и в том, что вам, наверное, просто не хватит времени нарастить число участников проекта с одной до более чем пяти команд, если, конечно, вы намерены ограничиться проектом, продолжительность выполнения которого не превышает три-четыре месяца.

Важность. При выборе пилотного проекта очень часто возникает соблазн остановить выбор на каком-нибудь малозначительном проекте, не связанном с серьезным риском. Если ситуация начнет развиваться в нежелательном направлении, потери будут не очень велики. В конце концов, люди могут даже не заметить провала по какому-нибудь малозначительному проекту. Однако поддаваться этому соблазну не следует. Напротив, нужно выбирать именно важный проект. Малозначительный проект не привлечет необходимого внимания со стороны окружающих. К тому же команде, переходящей к использованию Scrum, придется “напрягаться”, преодолевать трудности; если команда будет выполнять малозначительный проект, это будет “расхолаживать” людей, и они окажутся неготовыми делать для успешного выполнения проекта все, что от них зависит. Один из первоходцев гибкой методологии разработки и изобретатель процесса Adaptive Software Development (адаптивная разработка программного обеспечения) Джим Хайсмит (Jim Highsmith) советует: “Не выбирайте в качестве учебного какой-нибудь маловажный проект. Выберите проект, очень важный для компании. В противном случае вам будет очень трудно реализовать все множество непростых вещей, которых потребует от вас внедрение Scrum” (2002, 250).

ПРИМЕЧАНИЕ

Scrum-проекты предполагают наличие так называемого владельца продукта, подробнее о котором мы поговорим в главе 7, “Новые роли”. Спонсор, который упоминается здесь, может и не быть владельцем продукта. Как минимум это должен быть некто с “бизнес-стороны” проекта, кто должен будет признать этот проект успешным (или не очень...).

Участие бизнес-спонсора. Внедрение Scrum требует изменений и с “бизнес-стороны” уравнения разработки, а не только с “технической стороны” этого уравнения. Очень важно, чтобы на “бизнес-стороне” оказался человек, имеющий время и желание работать с командой. Активный бизнес-спонсор способен оказать команде помощь, если ей придется действовать вопреки традиционным бизнес-процессам и преодолевать сопротивление целых подразделений или отдельных лиц. Кстати, никто не принесет такой пользы в деле пропаганды успехов проекта (после его завершения), как спонсор, получивший то, на что он рассчитывал. Один спонсор, рассказавший другому, что проект, выполненный посредством методологии Scrum, принес гораздо лучшие результаты, чем предыдущие проекты, выполнявшиеся посредством традиционной методологии, принесет делу внедрения и распространения Scrum огромную пользу, поскольку убедит других спонсоров просить свои команды опробовать этот новый подход.

Выбор времени для внедрения Scrum

То обстоятельство, что новые комплексы физических упражнений и диеты многие люди начинают применять в первый день Нового года, отражает неистребимое желание человека согласовывать начало тех или иных перемен в своей жизни с какими-либо внешними факторами (например, с особыми датами в календаре). Точно так же, как многим из нас может казаться, что новый комплекс физических упражнений следует начинать именно 1-го января, мы можем полагать, что новый процесс разработки программного обеспечения следует применять с первого дня выполнения нового проекта. Выбор нового проекта (или повторное начало провалившегося проекта) в качестве пилотного проекта позволяет вам “начать жизнь с чистого листа”. Команды, которые предпочитают именно такой подход, начинают с журнала запросов на выполнение работ. Такая команда обычно откладывает начало своего первого спринта до тех пор, пока не будет создан журнал запросов на выполнение работ, который должен содержать описание всех функциональных возможностей, известных на данный момент. Тронд Вингард (Trond Wingård), руководитель проектов, выполняющих посредством гибкой методологии разработки, придерживается именно такого подхода.

В одном из моих первых проектов, выполненных посредством гибкой методологии разработки, наш клиент уже успел к тому времени потратить целый год и примерно 150 тысяч долларов на составление (с помощью другого подрядчика) классической документации с перечислением всех требований. Мне удалось убедить этого клиента в том, что такую документацию нужно заменить историями пользователей. В результате 150-страничный документ был заменен журналом запросов на выполнение работ, содержащим 93 истории пользователей. Не сделав этого, нам не удалось бы применить гибкую методологию разработки.

У старта “с чистого листа” есть лишь один серьезный недостаток: нужно ждать появления нового проекта и надеяться, что он будет подходящим кандидатом на роль пилотного Scrum-проекта. Это приводит к неоправданным задержкам с использованием выгод, которые способна принести методология Scrum.

Воскрешение какого-либо уже провалившегося проекта также может создать вашему пилотному проекту имидж “чистого листа”. Потратив несколько дней на создание журнала запросов на выполнение работ для такого проекта, вы поможете своей команде сконцентрироваться на предстоящей работе, возродить интерес всех, кто так или иначе причастен к этому проекту, а также привлечь внимание остальных сотрудников своей организации. Начиная проект “с чистого листа”, помните о том, что у вас есть шансы увязнуть на недели (или даже на месяцы!) в создании предварительного варианта журнала запросов на выполнение работ. Вы только представьте себе начало своего перехода к использованию Scrum с двухмесячной фазы сбора требований. Начиная проект “с чистого листа”, постарайтесь как можно быстрее создать журнал запросов на выполнение работ (не беда, если при этом он получится несколько поверхностным).

Когда приближается час расплаты

Подчас начать проект “с чистого листа” не представляется возможным или целесообразным. Если проект находится в середине своего выполнения и может только выиграть от применения методологии Scrum, я не вижу причин, которые мешали бы

применить эту методологию. Лично я отдаю предпочтение пилотным проектам, которые неумолимо движутся к провалу, но у меня еще остается достаточно времени, чтобы изменить ситуацию к лучшему и в конечном счете добиться успеха. Несмотря на то что подобный подход кажется весьма рискованным, этот риск, по сути, не так уж велик: традиционным способом спасти проект не удастся все равно, так почему бы не попробовать спасти его с помощью Scrum? Если вы сможете завершить его с запозданием, это, скорее всего, все равно будет рассматриваться как успех. Если же вам удастся завершить его в срок, это будет рассматриваться как грандиозный успех. Вследствие высокой сконцентрированности и интенсивности, порождаемых работой короткими спринтами, а также акцента на создании по крайней мере небольшого заблаговременного прогресса Scrum зачастую идеально подходит для проектов такого рода, особенно когда команда может положиться на опытного Scrum-мастера или консультанта.

Занимая пост главного технолога в Sammy Studios (в настоящее время — High Moon Studios), Клинтон Кит (Clinton Keith) понимал, что требуются какие-то сильнодействующие средства. Его команда разрабатывала видеоигру под названием Triple-A для Sony PlayStation и Microsoft Xbox. Команды работали не покладая рук, но дело продвигалось не так быстро, как того хотелось владельцам этой проектной организации. Чтобы предотвратить провал проекта, нужно было вносить какие-то изменения.

К счастью, примерно в это же время Кит узнал о существовании Scrum и решил познакомить с этой методологией свои команды. Работники студий, в которых разрабатываются видеоигры, отличаются исключительным индивидуализмом, поэтому внедрение процесса, который предполагает высокую степень сотрудничества, коллективных обсуждений, проведение ежедневных совещаний разработчиков (все это в полной мере присуще Scrum), было весьма непростым делом. Однако Кит выбрал для внедрения Scrum самый подходящий момент, когда члены команд начали осознавать, что используемый ими подход не приведет к нужному результату.

Еще одним удачным моментом, когда можно сыграть на риске “приближающегося часа расплаты”, является возможное банкротство компании или (в случае более диверсифицированной компании) ситуация возможного аннулирования проекта и утраты компанией выгодного заказа, если разработка будет продолжаться прежними темпами. Каждый раз, когда сохранение статус-кво может повлечь за собой серьезные последствия, демонстрация пагубных последствий бездействия может ускорить внедрение Scrum. В конце концов, если сохранение существующего порядка вещей может привести лишь к провалу, вам будет гораздо легче убедить членов команды попробовать что-нибудь новое, поэкспериментировать с использованием других подходов и перейти к Scrum, чему они в противном случае наверняка сопротивлялись бы.

Открывая людям глаза на грядущий крах, вы применяете сильнодействующее, но опасное средство. Чтобы оно сработало, опасность, грозящая проекту или организации, должна быть вполне реальной. В свое время мне пришлось работать в компании, глава которой славился тем, что каждый новый проект он называл судьбоносным, утверждая, что будущее компаний целиком зависит от этого проекта. Иными словами, не слишком часто кричите “Пожар!”, иначе люди перестанут вам верить. Не следует преувеличивать опасности. Но если проект действительно катится под откос и его могут спасти только решительные и радикальные меры, об этом нужно говорить, не стесняясь. Люди, наверное, и без ваших напоминаний догадываются об этом, но просто не желают признавать этой суровой действительности. К тому же, если члены

команды утрачивают живой интерес к своему проекту и своей работе, я иногда указываю им на вероятную опасность, которая может возникнуть, если все оставить в неизменном виде. Недавно я применил этот прием к команде, которой было известно, что руководство их компании ведет с конкурентом переговоры о слиянии. “Итак, — спросил я у членов команды, — когда слияние состоится и большие начальники объединенной компании начнут выяснять, какие проекты являются избыточными и каким командам можно доверить выполнение новых, перспективных проектов, как они, на ваш взгляд, отнесутся к *вашему* проекту и *вашей* команде?” Именно такое “прояснение сознания” требуется некоторым командам.

Выбор пилотной команды

Пересечение четырех факторов, представленное на рис. 5.1, а также приведенное выше обсуждение наиболее подходящего момента для выполнения пилотного проекта не учитывают, вероятно, самый важный фактор успеха пилотного проекта — людей, которые будут его выполнять. Я совершенно сознательно решил исключить человеческий фактор из дискуссии о выборе наиболее подходящего пилотного проекта, исходя из того, что пилотный проект и команду, которая будет его выполнять, можно выбирать независимо. Иначе говоря, мы можем выбрать в качестве своего пилотного Scrum-проекта наиболее подходящий для нас проект, а затем подобрать для него наиболее подходящую команду. Я отдаю себе отчет в том, что для многих организаций такой подход является непозволительной роскошью: проект и команда зачастую выбираются, как принято говорить, “в пакете”. Если вы не в состоянии разделить решения, касающиеся выбора идеального пилотного проекта и идеальной команды для реализации этого пилотного проекта, просто учтите все факторы при выборе лучшего из имеющихся вариантов пилотных проектов.

Формируйте исходные команды с учетом совместимости их членов, их умения вести между собой конструктивную полемику, готовности и способности к обучению и адаптации, владения требуемыми навыками, коммуникативности и т.п. Из всех этих качеств самым важным соображением при выборе команды, которой предстоит выполнять пилотный проект, является готовность ее членов к экспериментированию, испытанию каких-то новых способов работы. В идеальном случае к этому времени все они уже должны были пройти этапы процесса ADAPT, которые мы обозначили как осознание и возникновение желания (см. главу 2, “ADAPТация к Scrum”). Когда мне предоставляется возможность подбирать членов команды, которой предстоит выполнять пилотный проект, я пытаюсь обеспечить сочетание следующих типов людей.

- **Лоббисты Scrum.** Возможно, ваш проект окажется не настолько большим, чтобы в него вошли все без исключения энтузиасты перехода к Scrum, но лично я всегда пытаюсь включить в пилотную команду как можно большее их число. Таким специалистам было бы обидно осознавать, что ими пренебрегли, хотя они могли бы внести значительный вклад в успех проекта.
- **Сдержаные оптимисты.** Эти люди понимают, что требуется какой-то новый подход к разработке, но в прошлом они не были готовы активно отстаивать необходимость перехода к Scrum. Сейчас они понимают преимущества Scrum, эта методология кажется им многообещающей и они были бы рады внести посильный вклад в успех Scrum-проекта.

- **Честные скептики.** Мне не хотелось бы, чтобы в команде, которой предстоит выполнять пилотный проект, появились саботажники, сознательно противодействующие успешному выполнению проекта или создающие помехи слаженной коллективной работе, которая является необходимым условием эффективного функционирования Scrum-команды. Однако это не означает, что я вообще не приемлю скептиков. На самом деле очень неплохо, если в составе вашей команды есть уважаемый, толковый скептик, особенно если в прошлом он неоднократно демонстрировал готовность признать свою неправоту и изменить свое мнение, если оно оказывалось несостоительным. Такие люди нередко становятся самыми горячими сторонниками перехода к использованию новой методологии, если на собственном опыте убеждаются в ее достоинствах.

Разумеется, все это должно сочетаться с наличием у человека совокупности навыков, необходимых для успешной работы над соответствующим проектом. Если целью вашего пилотного проекта является, например, разработка новой видеоигры, то в состав команды нужно включить аниматора. Кроме того, я отдаю предпочтение людям, которые уже имеют определенный опыт успешной совместной работы. Иногда удается найти готовую команду, которая может приступить к работе над пилотным проектом. В других случаях приходится обращаться к прошлому опыту работы и подбирать людей, которые уже работали вместе над теми или иными проектами и добивались успеха.

ВОЗРАЖЕНИЕ

“Все эти меры, направленные на подбор подходящей команды, создают заведомые преимущества для исполнителей пилотного проекта. Разумеется, такая команда имеет более высокие шансы на успех. Но после того как мы внедрим Scrum, далеко не каждый проект можно будет укомплектовать людьми, которые являются энтузиастами новой методологии и которые в прошлом успешно работали вместе”.

Разумеется, “меры, направленные на подбор подходящей команды, создают заведомые преимущества для исполнителей пилотного проекта”. Я уже говорил, что пилотный проект выполняется не для того, чтобы проверить работоспособность Scrum. Мы и без того уверены в работоспособности Scrum. Существует немало свидетельств и надежных данных, подтверждающих это. Но мы не знаем, какими окажутся особенности применения Scrum в данной организации. Пилотный проект — это отнюдь не двойное контрольное испытание, которым обычно пользуются в клинической практике. Это лишь попытка использовать некий новый подход к выполнению важного проекта. Именно поэтому мы создаем для такого проекта благоприятные условия и пытаемся извлечь из его выполнения полезные уроки.

Если пилотный проект неудачен

Как быть, если — после принятия соответствующих решений, планирования и кропотливой работы — пилотный проект завершился неудачно? Во-первых, не следует возлагать слишком больших надежд на один крупный пилотный проект. Напротив, следует выполнить несколько пилотных проектов и помнить, что цель пилотного проекта — продемонстрировать путь, которого нужно придерживаться при выполнении пилотных проектов. Самые успешные пилотные проекты должны давать ответы

на следующие вопросы: *что следует делать и чего делать не следует?* Если команды, участвовавшие в пилотном проекте, поняли в результате его выполнения, что следует делать и чего делать не следует, уточнили, какие аспекты Scrum можно легко внедрить в их организации, уяснили типы и источники сопротивления, специфические для данной организации, и получили любую другую информацию такого рода, то я отказываюсь называть такой пилотный проект неудачным.

Но как быть, если пилотный проект не принес ожидаемых результатов?

В таких случаях я начинаю с оценки, были ли достаточно реалистичными ожидания, связанные с этим проектом. Возможно, до начала его выполнения нам казалось, что эти ожидания реалистичны, но к моменту завершения проекта мы поняли, что ошиблись в своих оценках. Если это действительно так, честно сообщите об этом всем заинтересованным лицам. Однако не следует представлять это как оправдание своей неспособности обеспечить ожидаемый результат. Все заинтересованные стороны должны знать, что команда берет на себя ответственность за роль, которую она играла в составлении и утверждении слишком оптимистичных планов. Важно только, чтобы все заинтересованные стороны понимали следующее: несмотря на то, что пилотный проект не оправдал всех возлагавшихся на него ожиданий, он мог бы принести ожидаемые (или даже еще более высокие) результаты, если бы не были допущены просчеты, столь естественные в каждом новом деле.

В конце выполнения пилотного Scrum-проекта его результаты часто сравнивают с нереалистичными предположениями относительно идеально выполняемого последовательного проекта (проекта, выполняемого по методу “водопада”). У вас могла сохраниться старая диаграмма Гантта, отражающая план проекта, который предусматривает два месяца анализа, месяц проектирования и два месяца кодирования и завершается месяцем тестирования. Эти весьма идеализированные шесть месяцев сравниваются затем с реалиями первого Scrum-проекта, выполнение которого заняло, скажем, тоже шесть месяцев. Противники Scrum могут сказать: “В чем же тут преимущество Scrum? В обоих случаях выполнение проекта заняло шесть месяцев. Между тем старый процесс привел к более удачному проекту. К тому же он более удобен с точки зрения использования в долгосрочной перспективе”. В этом случае мы имеем дело с нечестным сравнением реалий Scrum-проекта, выполнявшегося шесть месяцев, и плана, который относится к проекту, выполняемому по методу “водопада” в соответствии с таким же календарным планом. Не нужно сравнивать реалии одного проекта с мифом другого проекта.

Формулировка ожиданий и управление ими

Это подводит нас к следующей теме: формулировка ожиданий и эффективное управление ими. В 1994 году я возглавлял команду, которая выполнила проект, считавшийся многими сторонними специалистами весьма успешным. Для компании, которая выполняла этот проект, полученный результат представлял огромный шаг вперед. Продукт, созданный в ходе выполнения этого проекта, заключал в себе гораздо больше функциональных возможностей, чем его предшественник. Он был создан с помощью самых передовых технологий (ранее компании не приходилось пользоваться такими технологиями) и включал разработку трех центров данных. Между тем этот проект рассматривали как практически полный провал.

Полученный продукт использовался в многочисленных колл-центрах, каждый из которых обслуживался более чем 300 телефонными операторами. Он должен был прийти на смену несколько неуклюжей, но испытанной системе, которая была для компании “вчерашним днем”. Операторы ожидали от новой системы чего-то необыкновенного. В ходе ежемесячных обзоров спринтов, проводившихся с участием операторов, я был просто шокирован их завышенными ожиданиями (достаточно сказать, что некоторые из них были просто технически невыполнимыми). Когда до завершения проекта, выполнение которого длилось почти год, оставалось примерно три месяца, я понял, что нужно что-то менять. С того момента я тратил почти все свое время на управление ожиданиями. Я встречался с операторами в каждом из колл-центров и как можно подробнее рассказывал о том, чего следует и чего не следует ждать от новой системы. Я популярно объяснял им, что использование новой системы не повлияет ни на мир во всем мире, ни на глобальное потепление, ни на объем талии каждой из них. Если бы не мои терпеливые объяснения, наша новая система воспринималась бы пользователями как полный провал.

С того времени я уже никогда не забывал о важности эффективного управления ожиданиями для окончательного успеха любого проекта. Формулировка ожиданий и управление ими, возможно, даже еще важнее в начале проведения какой-нибудь крупной трансформации, такой, как внедрение Scrum. Мне кажется, что при инцициировании перехода к использованию Scrum было бы полезным сформулировать ожидания (и надлежащим образом управлять ими), касающиеся следующих четырех вещей: прогресса, предсказуемости, отношения к Scrum и участия всех заинтересованных сторон.

Ожидания, касающиеся прогресса

Если заинтересованные стороны, не принимающие непосредственного участия в проекте, а также “посторонние лица” уже слышали кое-что о Scrum, то есть шансы, что команда будет действовать быстрее. Я сам был свидетелем этому, когда меня пригласили выступить с лекцией в одной из крупных компаний Силиконовой долины, в которой к тому времени уже успел побывать консультант по Scrum, рассказывавший руководству этой компании о достоинствах Scrum. Когда я пришел, чтобы выступить перед этими же людьми, то начал с вопроса, что им уже известно о Scrum. Все, что они могли вспомнить из своего общения с консультантом по Scrum, сводилось к следующему: “Команды будут работать быстрее и мы сможем вернуться к своему традиционному методу разработки в любой момент, когда пожелаем”. Придя в себя от удивления, я сказал им, что они совершенно правы, правда не следует забывать о том, что все это потребует от них напряженной работы, а если они будут слишком часто шарахаться из одной крайности в другую, то за это придется заплатить значительным снижением производительности труда.

Что же касается ожиданий, связанных с ускорением работы команды, то совет Джима Хайсмита (Jim Highsmith) кажется мне гораздо более консервативным и реалистичным.

При выполнении проекта, рассчитанного на шесть месяцев, цель может заключаться в том, чтобы не опуститься ниже уровня производительности, исторически сложившегося для данной команды (снижение производительности в начале

выполнения проекта и ее повышение в конце), при условии повышения качества и более точного соответствия ожиданиям клиента. Оказание на команды с самого начала слишком большого давления очень часто заставляет их отказываться от осваиваемых ими новых методов и возвращаться к испытанным методам, в которых команда по-прежнему уверена (2005).

Повысится ли производительность команды, в значительной мере зависит от того, насколько успешно эта команда действовала до внедрения Scrum. Команда, которая и без того действовала достаточно успешно (научившись преодолевать несовершенства, присущие текущей организации и процессу), вероятнее всего, как утверждает Джим Хайсмит, поначалу несколько снизит свою производительность. Напротив, команда, которая ранее испытывала большие трудности, вполне может с самого начала повысить свою производительность.

Однако, как показывает мой собственный опыт, есть две вещи, присущие с самого начала почти всем командам.

- **Большинство команд ожидает слишком многое от своего первого спринта.** Если у команды нет солидного опыта работы итерациями, строго ограниченными временными рамками, ее члены могут подумать, что за пару-тройку недель они в состоянии сделать гораздо больше, чем это возможно в действительности. Например, команда может взять на себя коллективное обязательство выполнить 850 часов запланированной работы в течение ближайшего четырехнедельного спринта. Впоследствии команда приходит к выводу, что из-за внезапных перерывов в работе, появления незапланированной работы, корпоративных накладных расходов и прочих факторов они могут выполнить лишь 725 часов этой запланированной работы. Они работали так же напряженно, как и планировали, но смогли выполнить несколько меньше, чем было запланировано, поскольку не учли возможное влияние других факторов.
- **Большинство команд оказываются более полезными.** Я использовал здесь слово “полезными” в качестве синонима слова “производительными”. Но термин “производительные” подразумевает количество (объем) произведенного продукта; обычно при разработке программного обеспечения это количество измеряется числом строк программного кода. Несмотря на то что я не выступаю категорически против подсчета строк программного кода (мне самому это приходится иногда делать для определенных целей), я не хочу сказать, что Scrum-команды начинают писать больше строк программного кода за определенное время (особенно если учесть, что большее число строк программного кода отнюдь не является абсолютным благом — подчас даже наоборот). Я хочу лишь сказать, что с самого начала большинство команд начинает делать больший объем полезной работы уже вскоре после перехода к Scrum. Это объясняется тем, что спринты фокусируют внимание команд на том, “что мы можем сделать в течение следующих стольких-то недель”. Многие традиционные проекты “топчутся на месте”, пытаясь найти “наилучшее” или “правильное”, или “универсальное” решение. Scrum-команда в большей степени склонна находить “достаточно хорошее” решение, проверять его, уяснить нечто важное для себя и вносить соответствующие изменения.

Ожидания, касающиеся предсказуемости

Когда я руководил организациями разработчиков (а не консультировал их, как делаю сейчас), поддержание высокой производительности моих команд было не единственной моей заботой. В не меньшей степени меня интересовало то, смогу ли я с достаточной степенью точности предсказать, сколько времени потребуется моей команде, чтобы завершить проект. Во многих отношениях я предпочитал иметь дело с командой, которая поддерживала приемлемый для меня, устойчивый (и, следовательно, предсказуемый) темп выполнения работы, а не с командой, которая временами работает на удивление быстро, а временами топчется на месте. При выполнении пилотных Scrum-проектов следует заранее предупредить все заинтересованные стороны о том, что поначалу темп выполнения работы будет менее предсказуемым, чем при выполнении проектов в той же организации традиционными, привычными для нее методами.

Scrum-команды измеряют прогресс с помощью показателя, известного как *скорость* (velocity), который представляет собой объем работы, выполненной (или запланированной к выполнению) в течение данного спринта. Скорость выражается в таких единицах, как “баллы историй” (story points) или идеальные человеко-дни. Скорость отличается особой изменчивостью на протяжении нескольких первых спринтов команды или организации. В конце концов, команда учится работать по-новому и многие из ее членов, возможно, пока еще только учатся работать вместе.

См. также Более подробно скорость описана в главе 15, “Планирование”.

Важно разъяснить всем заинтересованным сторонам, что ранние подсчеты с использованием такого показателя, как скорость, будут особенно ненадежными. После того как команда поработает и накопит определенную статистику, можно будет сказать что-то вроде “Средняя скорость этой команды равна 20, причем наиболее вероятный ее диапазон составляет от 15 до 25”. Эти величины можно будет затем сравнивать с совокупной оценкой размера проекта с целью получения оценки продолжительности его выполнения. Например, если практика показывает, что скорость находится в диапазоне от 15 до 25, то оценка времени выполнения проекта, состоящего из 150 баллов историй, — от 6 до 10 спринтов.

Однако до тех пор, пока команда не накопит достаточный объем статистики, подобные прогнозы могут быть чрезвычайно рискованными. Это означает, что контракт, заключающий в себе высокую степень риска и предполагающий крупные штрафы за нарушение сроков поставки готового продукта, — это не совсем то, что требуется для пилотного Scrum-проекта. (Вообще говоря, такой контракт вообще не очень подходит для любого изменения методики работы.) Так какой же объем статистики вам потребуется, прежде чем вы сможете делать подобные прогнозы? Проще всего ответить “Чем больше, тем лучше”. Вы можете приступить к составлению прогнозов после того, как команда завершит свой первый спринт, но при этом не следует забывать о вероятности большой ошибки, обусловленной этим первым значением скорости, установленным вами. Полезно знать, что скорость большинства команд вполне стабилизируется после выполнения трех-четырех спринтов. Разумеется, это не является правилом. Если в среде, где выполняется проект, меняется очень многое (например, появляются новые технологии, происходит частая ротация членов команды и т.п.), то и скорость может колебаться в широких пределах.

Ожидания, касающиеся отношения к Scrum

Получив достаточно времени, чтобы приспособиться к работе по-новому, большинство разработчиков отдает предпочтение Scrum. Например, опрос сотрудников компании Yahoo! показал, что 85% членов всех команд продолжили бы применять методологию Scrum, которой они уже научились пользоваться, если бы соответствующее решение зависело исключительно от них (Deemer et al. 2008, 16). Но изменение отношения к Scrum обычно начинается не с этого. Инициаторы перехода к использованию Scrum должны быть готовы к тому, что с первых же дней на них обрушится поток возражений и нареканий. Ниже перечислены типичные жалобы.

- Все рабочее время тратится на ежедневные совещания разработчиков.
- В конце каждого спринта много времени тратится на тестирование продукта, несмотря на то что продукт отправляется клиенту не после каждого спринта.
- Руководители не в состоянии оценить меня достаточно объективно, поскольку не могут сказать точно, какую именно работу выполнил я.
- Система разваливается уже через шесть месяцев после ее выпуска, поскольку мы не создаем адекватной документации по ее сопровождению.

При возникновении первых же проблем и затруднений возникает соблазн по-быстрее отказаться от новой методологии и вернуться к испытанному способу разработки. Вот что писал по этому поводу Дэрил Коннер, автор книги *Managing at the Speed of Change*: “Не так уж сложно заставить людей уяснить необходимость перемен и приступить к их осуществлению. Значительно сложнее заставить их не сворачивать с выбранного пути после того, как возникнут первые серьезные трудности” (1993, 116). Один из наилучших способов избежать возврата к старым методам работы — предвидеть возникновение подобных проблем, загодя поговорить об этом с людьми и взять с них слово, что, когда возникнут серьезные трудности, они не откажутся от использования Scrum несмотря на дискомфорт и временные проблемы.

Ожидания, касающиеся участия всех заинтересованных сторон

Одно из самых важных ожиданий, к управлению которыми следует приступить как можно раньше, касается участия в процессе внедрения Scrum всех заинтересованных сторон. Многие из тех, кто связан с проектом разработки программного обеспечения, привыкли к традиционному взгляду на их роль в таком проекте, очень похожую на роль владельца автомобиля, который явился с ним в автомастерскую: от вас лишь требуется сказать, чего вы от них хотите, и прийти в назначенное время, чтобы забрать отремонтированный автомобиль и оплатить счет. Заинтересованные стороны — особенно те, которым отведена роль владельца продукта — должны понять, что это вовсе не тот способ, с помощью которого можно создавать сложные программные продукты.

Обязательно обсудите ожидания с владельцем продукта и с другими заинтересованными сторонами, чье участие потребуется вам либо во время спринтов, либо во время обзоров спринтов. Удостоверьтесь в том, что каждая из заинтересованных сторон знает, на какой уровень ее участия в проекте рассчитывает команда.

Scrum — это не волшебная палочка, которая решает все проблемы организации-разработчика. Нужно с самого начала позаботиться о том, чтобы “градус ожиданий” не повышался до недопустимого уровня. Управление ожиданиями — едва ли не самое важное, чем следует заняться буквально с первых дней выполнения проекта. Если же вы не уделите должного внимания этой проблеме, то рискуете тем, что переход к использованию Scrum, успешный во всех отношениях, будет воспринят как провал.

Речь идет о пилотном проекте

Пит Димер (Pete Deemer), независимый консультант по Scrum, занимал в Yahoo! должность вице-президента по производству, когда им была инициирована программа перехода к Scrum. Димер исходил из того, что любой пилотный проект является своего рода экспериментом и цель этого эксперимента — получить знания, которые помогут компании добиться успеха при выполнении последующих проектов. Димер также признавал, что, называя эти проекты пилотными, он вполне допускает, что их выполнение далеко не всегда будет проходить гладко. Он говорил, что всегда надеялся на то, что “в случае появления серьезных проблем люди энергично возьмутся за дело и попытаются найти эффективное решение возникших проблем”. Димер использовал определение *пилотный*, чтобы таким образом создать некий пояс безопасности вокруг реализации данного процесса.

Димер признавал высокую ценность такого пояса безопасности. Этот пояс создавал комфортную зону, в которой команды могли спокойно экспериментировать, находя наиболее эффективные способы использования Scrum. Однако даже спустя год с момента начала внедрения Scrum в Yahoo! и даже после того, как этой новой методологией пользовалось уже более ста команд Yahoo!, Пит Димер по-прежнему называл каждый проект пилотным. Я спросил у него, когда же он прекратит называть их пилотными. Димер сказал, что до тех пор, пока *каждый* проект в Yahoo! не будет выполняться с помощью этой новой методологии, и до тех пор, пока они не будут знать *все*, что им нужно знать о Scrum, он будет продолжать называть их пилотными.

Рассматривая каждый свой проект как пилотный, нужно помнить, что самыми важными являются несколько первых спринтов. Вы можете способствовать тому, чтобы эти начальные спринты послужили надежной стартовой площадкой для ваших команд, тщательно выбрав первый проект и команду, которая будет выполнять этот проект, точно сформулировав ожидания, связанные с выполнением этого проекта, и эффективно управляя этими ожиданиями.

Дополнительная литература

Karten, Naomi. 1994. *Managing expectations*. Dorset House.

Полезная и легкая в чтении книга, содержащая множество ценных советов. Основное внимание в ней уделяется взаимодействию с клиентами, но почти все содержащиеся в ней советы применимы и к другим отношениям в организации. Предлагаются советы по таким вопросам, как умение выслушать собеседника, прояснение отдельных моментов, умение избегать конфликтных заявлений и выработка взаимоприемлемых решений.

Little, Todd. 2005. Context-adaptive agility: Managing complexity and uncertainty. *IEEE Software*, May-June, 28–35.

Автор (Тодд Литтл, член совета директоров Agile Alliance и один из основателей Agile Project Leadership Network) представляет на суд читателей схему разделения проектов на Bulls (“быки”), Colts (“жеребята”), Cows (“коровы”) и Skunks (“скунсы”) в зависимости от степени неопределенности и сложности, присущих соответствующему проекту. Эту схему можно применить к выбору исходного Scrum-проекта, когда следует избегать выбора того типа проектов, который Тодд Литтл обозначает как “быки” (проекты, характеризующиеся высокими неопределенностью и сложностью).

ЧАСТЬ II

Люди

Нам нужно научиться ценить... людей и взаимодействие между ними больше, чем процесс и инструменты.

— The Agile Manifesto

Глава 6

Преодоление сопротивления

В своей статье, опубликованной в 1969 году в журнале *Harvard Business Review*, Пол Лоуренс (Paul Lawrence) отмечал, что в переменах “присутствует как технический, так и социальный аспект. Технический аспект перемен заключается в осуществлении какой-либо значимой, поддающейся измерению модификации в физических методах выполнения работы. Социальный аспект перемен относится к тому, как люди, которых эти перемены коснулись, представляют себе возможное влияние этих перемен на отношения, уже сложившиеся в организации”. Сталкиваясь с сопротивлением, обычно делают акцент на выгодах технического аспекта перемен. В конце концов, мы сами уже убедились в достоинствах предлагаемых перемен, поэтому легко предположить, что единственное, что нам остается сейчас сделать, — это убедить остальных. Нам кажется, что если мы перечислим “железные” аргументы в пользу предлагаемых перемен, сопротивление тех, кто сомневается, испарится само собой. Пол Лоуренс спорит с этой сомнительной логикой: “Подчас нам хочется, чтобы обоснованность технического аспекта перемен была единственным критерием его приемлемости. Однако нам никуда не деться от того факта, что именно социальный аспект перемен определяет наличие или отсутствие сопротивления этим переменам” (1969, 7).

Несмотря на то что именно социальный аспект перемен может обусловливать сопротивление, источником этого сопротивления являются конкретные люди. Переменам сопротивляются не команды или целые подразделения, а конкретные люди. Таким образом, в этой главе рассматриваются эффективные методы, позволяющие преодолеть сопротивление “человеческого материала” в лице отдельных работников. Сначала мы рассмотрим, нельзя ли предвидеть возникновение сопротивления и предпринять превентивные меры, позволяющие его устраниить. Затем мы рассмотрим, как рассказать людям о грядущих переменах, и попытаемся понять, почему те или иные “послания” лучше всего доводятся до людей определенными “рупорами”. Наконец в этой главе мы рассмотрим, как и почему люди сопротивляются переменам, а затем воспользуемся этой информацией, чтобы определить эффективные приемы, которые позволят преодолеть их сопротивление.

Можно ли предвидеть возникновение сопротивления

Нет ничего удивительного в том, что некоторые люди сопротивляются переходу к Scrum. Вообще говоря, многие люди сопротивляются любым переменам. Подозреваю, что если вы приедете в какую-нибудь компанию и объявите, что зарплата каждого ее сотрудника будет повышенна на 20–50%, сопротивление возникнет даже в этом случае. Одни заподозрят начальника в каких-то тайных мотивах (наверняка начальство выдвинет какие-то дополнительные условия!). Другим покажется несправедливой сама величина прибавки к зарплате (почему мне добавили меньше, чем моим коллегам?).

Столь радикальная трансформация, как переход к Scrum, может вызвать в организации серьезные катаклизмы. Расширяется круг обязанностей сотрудников, меняется система субординации, происходит перераспределение властных полномочий, меняются ожидания. Кто-то рассчитывает добиться личного или профессионального выигрыша от таких перемен, а кто-то заранее сокрушается, не ожидая от них ничего хорошего для себя. Понять, как эти перемены повлияют на вашу организацию, очень важно — хотя бы для того, чтобы правильно прогнозировать появление возможных источников сопротивления.

Такие выводы подтверждаются результатами исследования, проведенного в 2007 году. Авторы этого исследования пытались понять, почему люди сопротивляются переменам. Они выявили, в частности, что главной причиной сопротивления переменам со стороны менеджеров является опасение потери контроля и власти (Creasey and Hiatt). Основные причины сопротивления переменам со стороны рядовых работников и менеджеров представлены в табл. 6.1.

Таблица 6.1. Основные причины сопротивления переменам со стороны рядовых работников и менеджеров

Номер	Рядовые работники	Менеджеры
1	Недостаточная информированность	Опасение потери контроля и власти
2	Страх перед неизвестностью	Нехватка времени
3	Опасение потерять работу	Устраивает существующее положение вещей
4	Недостаточная поддержка со стороны руководства	Непонятно, “что лично я буду иметь от этих перемен”
5		Невозможность участвовать в принятии решений

Кто будет сопротивляться

Для того чтобы предсказать возможные источники сопротивления переменам, полезно попытаться ответить на перечисленные ниже вопросы.

- Кто может потерять (властные полномочия, престиж, влиятельность и т.п.) в результате успешного перехода к Scrum?
- Какие могут возникнуть “коалиции сопротивления переменам”?

Выявив конкретных людей, которые могут понести определенные потери в результате перехода к Scrum, а также потенциальные коалиции сопротивления переменам,

вы поймете, куда следует направить первоначальные усилия, призванные ослабить такое сопротивление.

Несмотря на то что одна часть сотрудников будет сопротивляться переменам, другая часть с готовностью их примет. Мицельвайт (Musselwhite) и Ингрэм (Ingram) классифицируют людей исходя из их отношения к переменам. Их способ классификации представлен на рис. 6.1 (Luecke, 2003). На одном полюсе расположились консерваторы, которые отдают предпочтение предсказуемости, сосредоточиваются на подробностях и процедурах, действуют осмотрительно и осторожно, дисциплинированы и организованы, предпочитают перемены, которые не затрагивают текущую структуру организации. По некоторым оценкам консерваторы составляют примерно 25% от всего населения.

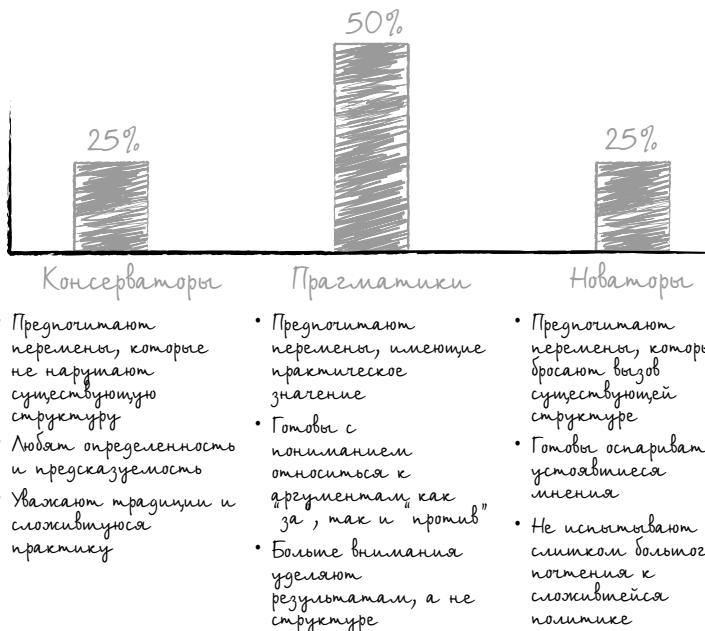


Рис. 6.1. Индивидуальное отношение к переменам

На другом конце этой шкалы находятся новаторы, которые также представляют примерно 25% от всего населения. Новаторы могут показаться неорганизованными и недисциплинированными людьми, склонными к риску и авантюрам. Они без особого почтения относятся к сложившейся политике и предпочитают перемены, которые бросают вызов сложившейся структуре. Промежуточное положение между новаторами и консерваторами занимают прагматики, на долю которых приходится примерно половина населения. Прагматики, как правило, практичны, говорчивы и гибки; их гораздо больше интересуют результаты, а не структура. Обычно они в большей степени приспособлены к работе в команде, чем новаторы и консерваторы; готовы с пониманием относиться к аргументам как "за", так и "против" и являются великолепными посредниками между новаторами и консерваторами.

Я пришел к выводу, что понимание отношений этих трех категорий работников к переменам помогает определить, кто, вероятнее всего, станет в вашей организации

источником сопротивления внедрению Scrum. Несомненно, консерваторы будут противиться внедрению Scrum. Изменения в способах работы, взаимодействии между членами команды и ожиданиях, которые влечет за собой внедрение Scrum, — именно тот тип перемен, который вступает в противоречие с природой консерватора.

ПРИМЕЧАНИЕ

Деление людей на консерваторов, прагматиков и новаторов, конечно, несколько упрощает истинное положение вещей. Разумеется, каждого человека нужно рассматривать как уникальную, неповторимую личность. Однако знание этих категорий поможет вам сформулировать стратегии преодоления сопротивления переменам. Понимание роли каждого отдельного человека в организации позволяет уяснить, почему те или иные работники сопротивляются переменам. Многие из причин, заставляющих людей сопротивляться переменам, описаны в главах 7, “Новые роли”, и 8, “Изменившиеся роли”.

Однако не только консерваторы будут противиться внедрению Scrum. Кое-кто из прагматиков также будет оказывать сопротивление. Поскольку прагматики в гораздо большей степени готовы с пониманием относиться к аргументам как “за”, так и “против”, а затем склоняться к поддержке правой, по их мнению, стороны, то ранние успехи, обусловленные переходом к использованию Scrum, будут способствовать превращению прагматиков в сторонников Scrum. Попытайтесь прибегнуть к следующим мерам, которые должны помочь вам превратить прагматиков в сторонников Scrum.

- Формируя команду для выполнения пилотного проекта, не забудьте включить в нее прагматиков.
- Позаботьтесь о том, чтобы прагматики, не попавшие в состав команды для выполнения пилотного проекта, ознакомились с результатами этого проекта.
- Позаботьтесь об обучении прагматиков.
- Ознакомьте прагматиков с успехами других компаний, предложив им поучаствовать в конференциях, войти в состав региональных групп, объединенных интересом к гибкой методологии разработки, и т.п.
- Не пытайтесь скрывать от людей недостатки и трудности, связанные с использованием Scrum, представляя эту новую методологию как универсальный способ решения всех проблем.
- Привлекайте прагматиков к участию в сообществах в поддержку усовершенствований, которые были описаны в главе 4, “Движение в направлении гибкости”.

Заблуждения и фобии

Многие из доводов против использования Scrum, которые часто приходится слышать, вполне предсказуемы и типичны для большинства организаций. Другие, конечно же, будут уникальными для вашей организации. Зачастую нетрудно предвидеть аргументы, которые вам придется слышать от противников перехода к Scrum, просто проанализировав проблемы, специфические для вашей организации и отрасли в целом, ее культуры и людей, применяемых технологий и выпускаемой продукции.

Выполняя такой анализ, вы обнаружите, что многие из возражений (как универсальных, так и специфических для вашей организации) можно разделить на заблуждения, связанные с отрицательным опытом работы по методу “водопада”, и фобии, связанные с гибкой методологией разработки. Заблуждение, связанное с отрицательным опытом работы по методу “водопада”, представляет собой ошибочное представление (или идею) относительно гибкой методологии разработки, или Scrum, обусловленное слишком долгой работой над проектами по методу “водопада”. Ниже перечислены примеры подобных заблуждений.

- Scrum-команды не составляют планы, поэтому мы не сможем обещать что-то определенное своим клиентам.
- Scrum требует, чтобы каждый участник проекта был универсалом.
- Наша команда разбросана по всему миру. Принцип самоорганизации входит в противоречие с некоторыми типами культур, поэтому мы не сможем пользоваться гибкой методологией разработки.
- Наша команда разбросана по всему миру, а Scrum требует регулярного личного общения членов команды.
- Scrum игнорирует архитектуру, что повлечет катастрофические последствия для того типа системы, которую мы проектируем.
- Scrum подходит для простых веб-сайтов; наша же система слишком сложна для этого.

Фобия, связанная с гибкой методологией разработки, — это сильный страх или неприязнь по отношению к гибкой методологии разработки, обусловленная обычно неопределенным характером перемен. Ниже перечислены некоторые из фобий, с которыми вам наверняка придется столкнуться.

- Я боюсь, что окажусь не у дел.
- Я боюсь, что меня уволят, если принятые нами решения окажутся неправильными.
- Я опасаюсь возникновения конфликтных ситуаций и неудачных попыток достичь консенсуса.
- Я боюсь, что люди увидят, как мало я в действительности делаю.
- Намного легче и безопаснее, когда кто-то рассказывает мне, что именно мне следует делать.
- Намного легче и безопаснее, когда я могу рассказать людям, что именно им следует делать.

Несмотря на то что заблуждения, связанные с отрицательным опытом работы по методу “водопада”, зачастую удается устраниТЬ с помощью рациональных доводов, историй из жизни и практических примеров, фобии, связанные с гибкой методологией разработки, обычно носят гораздо более личный и эмоциональный характер. Иногда людям просто требуется, чтобы их возражения были услышаны.

В этой книге я пытался предупредить как можно больше заблуждений, связанных с отрицательным опытом работы по методу “водопада”, и фобий, связанных с гибкой методологией разработки. Во многих главах содержатся врезки возражений, в которых излагаются мои советы о том, как решить типичные проблемы и избежать типичных заблуждений, связанных со Scrum.

Информация о переменах

Если вы еще раз обратитесь к табл. 6.1, то заметите, что основной причиной сопротивления людей переменам является недостаточная информированность. Однако я уверен, что если бы мы просмотрели папки с уничтоженными сообщениями электронной почты всех, кто принимал участие в этом исследовании, то нашли бы как минимум одно письмо, разъясняющее необходимость соответствующих перемен. Однако узнать о причине и понять причину — не одно и то же. Большинству из нас нужно несколько раз услышать какое-то соображение (причем желательно, чтобы каждый раз это соображение было сформулировано по-разному), чтобы оно дошло до нашего сознания и прочно в нем закрепилось. Важно, однако, не только услышать послание несколько раз, но и услышать его от авторитетного источника, которым, как правило, является лидер коллектива. Посланиям, которые исходят от равных вам по статусу людей, как правило, придается меньшее значение, чем посланиям лидеров.

Послания лидеров

Неудивительно (и это подтверждают результаты исследований), что работники предпочитают получать разные типы информации от разных людей (Hiatt 2006, 12). Работники предпочитают слышать разъяснения о необходимости тех или иных перемен в организации от человека, занимающего в ней более высокую должность, чем они сами. Те же работники предпочитают узнавать о том, как эти изменения повлияют на них лично, от своего непосредственного начальника. Это означает, что в то время как президент компании или глава подразделения является идеальным источником информации о причинах перехода к Scrum, людям нужно предоставить возможность встретиться со своим непосредственным начальником, чтобы обсудить влияние перехода к Scrum на их личные судьбы. Вместе с тем существуют послания, идеальными источниками которых являются товарищи по работе.

Если вы — формальный лидер в своей организации (или признаны лидером неформально), значит, именно вам надлежит сообщить людям о предстоящих переменах. Когда вы станете рассказывать им о будущем, вам наверняка зададут вопросы, на которые вы не сможете ответить. Следует ли ожидать сокращений? Кто кому будет подчиняться? Кто будет составлять свою характеристику за истекший год? Если вы не знаете ответа на тот или иной вопрос, не пытайтесь гадать. Главное — быть всегда честным с людьми. Помните: одна маленькая ложь рождает большое недоверие, а для восстановления доверия людей могут понадобиться годы.

Кроме того, рассказывая людям о предстоящих переменах, прислушивайтесь к их мнениям. Кем бы вы ни были, формальным или неформальным лидером, вы должны не только рассказать людям о том, что их ожидает, но и выслушать их возражения, если такие будут высказаны явно или хотя бы будут подразумеваться в соображениях, которые они выскажут вам. Взгляните еще раз на перечень типичных причин сопротивления переменам (см. табл. 6.1). Обратите внимание: ни одна из этих причин не была сформулирована как “мне кажется, что эти изменения нам не нужны”. Разумеется, в организации найдутся и такие, кому покажется, что переход к использованию Scrum является плохой идеей, но людей, которые сопротивляются в силу иных, более личных причин (социальные аспекты перемен, упоминавшиеся в начале этой главы), всегда будет намного больше. Общаюсь с людьми, тратьте больше времени на

выслушивание их мнений, а не на изложение собственного. Разговаривая с каждым, кто сопротивляется переменам, пытайтесь закончить вместо своего собеседника такое предложение: “Я не могу использовать Scrum, поскольку это означает, что мне...” Существует бесконечное множество способов закончить его. При выполнении заказа одного из своих недавних клиентов я смог закончить указанное выше предложение таким образом для некоторых из работников, с которыми я встречался в тот день.

- Пришлось бы “пахать” больше, чем мне хотелось бы сейчас
- Пришлось бы прекратить выполнение той части моего задания, которая мне очень нравится
- Пришлось бы больше разъезжать по командировкам, чтобы находиться в более тесном контакте со своими подчиненными, которые трудятся в разных городах
- Не удалось бы скрывать, что я уже не настолько хороший программист, как прежде
- Не удалось бы руководить таким большим коллективом, как в настоящее время

Ни одно из этих соображений не было произнесено вслух людьми, с которыми я встречался в тот день. Но мне удалось прочитать эти тайные мысли благодаря тому, что я внимательно выслушал все, что они мне говорили. Уяснение истинных причин сопротивления людей поможет вам впоследствии устраниТЬ это сопротивление.

Мнение коллег

Любой успешный план общения с людьми включает предоставление широких возможностей для всех, кто не надеется услышать мнение своих коллег. Подобное соображение высказывает автор статьи, опубликованной в *MIT Sloan Management Report*.

Наилучшим направлением воздействия на других, особенно в период неопределенности, является воздействие сбоку, а не сверху. Для лидеров это означает, что нужно предоставить возможность тем, кто еще не осознал необходимость перемен, ознакомиться с мнением тех, кто эту необходимость осознал. Наиболее подходящей площадкой для такого обмена мнениями являются, наверное, собрания коллектива. Хотя бы один раз выслушать мнение своего товарища по работе для человека гораздо важнее, чем десяток раз выслушать мнение какого-нибудь начальника (Griskevicius, Cialdini, and Goldstein, 2008, 86).

Интересный случай, касающийся важности мнений товарищей по работе, связан с Сильваном Голдманом (Sylvan Goldman), который в 1937 году изобрел тележку для покупок в магазинах самообслуживания, после того как заметил, что покупатели в магазине самообслуживания, где часто приходилось бывать самому Сильвану, направлялись в кассу сразу же после того, как корзинки, которые они несли в руках, оказывались наполненными до краев. Как ни странно, тележки для покупок, предложенные Сильваном Голдманом, поначалу не пользовались особой популярностью. Они оставались не у дел до тех пор, пока Голдман не нанял двух актеров, мужчину и женщину, которые просто расхаживали по магазину с тележками, делая вид, что совершают покупки. После того как покупатели увидели людей с тележками, которых они приняли за обычных покупателей, т.е. за таких же, как и они сами, они также начали пользоваться тележками. В наши дни тележки для покупок стали непременным атрибутом всех супермаркетов.

Рассмотрим пример, более близкий моим читателям. Вспомните случай, когда вы присутствовали на какой-либо конференции или отраслевой выставке и наблюдали группу посетителей, столпившихся вокруг кого-либо из участников конференции или возле какого-то выставочного стенда, чтобы послушать выступающего. Вполне допускаю, что вы подошли поближе к ним, чтобы послушать, о чем там говорят. Вспомните случай, когда вы видели начало выступления какого-нибудь уличного исполнителя (музыканта, певца или жонглера). Постояв неподалеку какое-то время, вы могли заметить, что с каждой минутой количество зрителей становится все большим и большим.

Эти примеры демонстрируют всю мощь влияния людей, равных вам по статусу. Если кто-то из коллег взахлеб рассказывает товарищам по работе о преимуществах нового способа работы, люди охотно прислушиваются к его словам. Переход к использованию Scrum создает множество возможностей для обсуждения между сослуживцами всего, что связано с этим переходом. Многие из таких обсуждений будут неформальными и спонтанными (например, такие обсуждения могут вестись за обеденным столом). Но эффективные лидеры перехода не полагаются на волю случая, а сами стремятся предоставлять своим подчиненным дополнительные возможности для таких обсуждений. Это может достигаться поощрением участия работников в сообществах практических пользователей Scrum или даже планированием более формальных презентаций, проводимых сотрудниками для своих товарищей по работе в обеденное время. Страйтесь, насколько это возможно, чтобы выступающий соответствовал своей аудитории. Примите к сведению следующий совет, который дают авторы исследования, в ходе которого изучалось влияние работников на своих коллег.

Пытаясь добиться, чтобы голоса сторонников перемен были услышаны их товарищами по работе, руководители часто отдают предпочтение тем из работников, которые умеют красиво и грамотно излагать свои мысли, в то время как следовало бы отдавать предпочтение тем, чьи конкретные обстоятельства в наибольшей степени соответствуют обстоятельствам членов коллектива, все еще сомневающихся в необходимости перемен. Так, если наибольшее сопротивление переменам оказываются работники с самым продолжительным стажем работы в данной организации, то самым подходящим для них агитатором за перемены может оказаться человек, уже давно работающий в этой организации, а вовсе не какой-нибудь краснобай, работающий в данной организации без году неделю (Griskevicius, Cialdini, and Goldstein, 2008, 86).

Тонкости индивидуального сопротивления

Люди сопротивляются переходу к Scrum по разным причинам. Одни сопротивляются потому, что их вполне устраивают традиционные способы работы и коллектив, в котором они трудятся. Возможно, им понадобилось много лет, чтобы освоить эти методы работы, приобрести высокую квалификацию, смыкнуться с коллективом и научиться ладить со своим нынешним руководителем. Другие могут сопротивляться переходу к Scrum из-за страха перед неизвестностью. Их девиз — “Лучше иметь дело со злом, которое вы знаете в лицо, чем с неизвестным злом”. Третья могут сопротивляться переходу к Scrum из-за недоверия к этой методологии, которое они не могут объяснить никакими рациональными соображениями. Четвертые могут объяснить

свою неприязнь к Scrum тем, что итеративный способ создания сложных продуктов без проектирования неминуемо закончится катастрофой.

Точно так же, как существует множество причин, в силу которых люди сопротивляются переходу к Scrum, существует множество способов такого сопротивления. Кто-то может сопротивляться переходу к Scrum открыто, выдвигая вполне обоснованные, разумные доводы и “железную” логику. Кто-то может сопротивляться “втихую”, саботируя меры, направленные на внедрение Scrum. “Вам кажется, что не следует составлять документацию? Прекрасно, не будет вам никакой документации”, — может размышлять пассивный противник перехода к Scrum, отказываясь документировать что бы то ни было, в том числе и отчеты об ошибках, выявленных в программном коде (несмотря на то что команда приняла решение продолжать составлять такие отчеты и хранить их в системе отслеживания ошибок). Кто-то может сопротивляться, молчаливо игнорируя любые перемены, продолжая как можно дольше работать старыми методами и рассчитывать на то, что появится какой-нибудь “еще более новый” метод, который заставит команду отказаться от использования Scrum.

Каждый акт сопротивления заключает в себе информацию о том, как люди относятся к внедрению Scrum. Являясь агентом перемен или лидером соответствующей организации, вы должны уяснить подлинные причины сопротивления того или иного работника, извлечь из этого надлежащие выводы и помочь человеку понять необоснованность причин, которые толкают его на сопротивление. Существует много методов, с помощью которых можно достичь этой цели. Важно лишь выбрать правильный метод — в противном случае ваши усилия могут не принести нужного результата. Облегчить выбор наиболее подходящего метода вам поможет размышление над тем, *как и почему* человек сопротивляется переменам. Причины, заставляющие человека сопротивляться переходу к использованию Scrum, можно разделить на две категории.

- Людям не хочется менять существующий порядок вещей
- Людям не нравится методология Scrum как таковая

Причины сопротивления, защищающие традиционный подход, относятся к первой категории. Этот тип сопротивления переходу к Scrum возникает независимо от того, какой именно тип перемен предполагается осуществить. Причины сопротивления относятся ко второй категории, если они являются доводами против конкретных последствий, вызванных началом перехода к использованию гибкой методологии разработки. В табл. 6.2 и 6.3 приведены примеры причин сопротивления и указана категория, к которой относится каждая из этих причин.

Таблица 6.2. Люди могут сопротивляться переходу к Scrum, поскольку их устраивает существующий порядок вещей

Примеры положительного отношения к существующему порядку вещей

Мне нравятся люди, с которыми я работаю

Мне нравятся власть и престиж, которые являются результатом моей нынешней роли

Именно так меня научили работать, а по-другому работать я не умею

Мне не нравятся никакие перемены

Я не хочу участвовать в очередной инициативе, связанной с какими-то переменами, поскольку такие инициативы всегда заканчиваются провалом

Таблица 6.3. Люди могут сопротивляться переходу к Scrum, поскольку им не нравится сама методология Scrum

Примеры неприязни к Scrum

Мне кажется, Scrum — это лишь модное увлечение. Через три года нас заставят внедрять какую-то очередную “новацию”

Scrum не подходит для наших продуктов

Я настолько хорошо освоил наш нынешний метод проектирования, что могу нацепить наушники и работать совершенно автономно от своих коллег

Scrum не подходит для таких рассредоточенных коллективов, как наш

Классификация того, *как* люди сопротивляются переменам, еще проще: является ли сопротивление активным или пассивным? Активное сопротивление имеет место, когда человек выполняет конкретные действия, направленные на то, чтобы помешать переходу к Scrum или хотя бы затормозить его. Пассивное сопротивление имеет место, когда человек не выполняет какие-то конкретные действия даже после того, как ему приказали это сделать. Объединив две общие причины, в силу которых люди могут сопротивляться переходу к Scrum, с двумя способами, которыми может осуществляться такое сопротивление, мы получим стандартную матрицу 2 2, показанную на рис. 6.2.

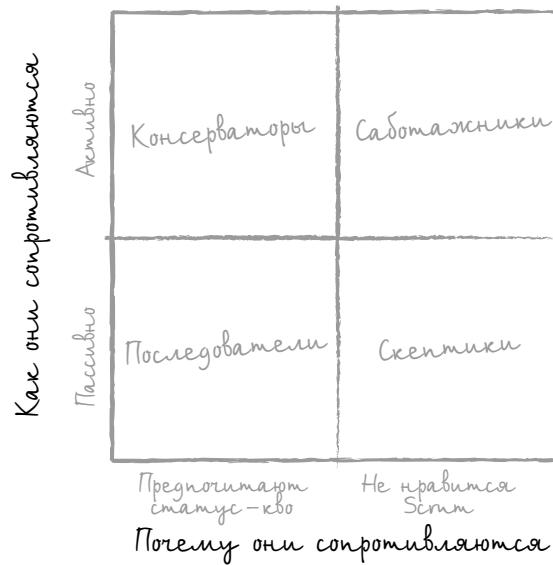


Рис. 6.2. Четыре категории лиц, сопротивляющихся переменам

Каждому квадранту на рис. 6.2 присвоено условное название, описывающее тип личности, которая сопротивляется способом, указанным обозначениями на осях. Скептик — это человек, не согласный с принципами или методами Scrum, но сопротивляющийся переходу к Scrum лишь пассивно. Скептики — это люди, которые вежливо возражают против перехода к Scrum, несколько чаще, чем следовало бы, “забывают” посещать ежедневные совещания разработчиков и т.п. Здесь я имею в виду людей, которые всерьез пытаются остановить переход к Scrum, а не людей, которые придерживаются вполне приемлемой точки зрения, выражющейся примерно такими

словами: “Это не очень-то похоже на то, что я делал до сих пор, но все это довольно интересно. Давайте-ка попробуем это в деле и посмотрим, что из этого получится”.

Непосредственно над скептиками на рис. 6.2 располагаются саботажники. Подобно скептикам, саботажники сопротивляются переменам не столько по причине поддержки традиционного способа разработки программного обеспечения, сколько из-за неприязни к Scrum. В отличие от скептиков саботажники *активно* сопротивляются переходу к Scrum, пытаясь активно противодействовать мерам, направленным на внедрение Scrum (например, продолжая составлять пространную документацию предварительного проектирования и т.п.).

Слева на рис. 6.2 располагаются те, кто сопротивляются переходу к Scrum потому, что их устраивает существующий порядок вещей. Их вполне устраивает то, как они работают в настоящее время, им нравятся собственный престиж и товарищи по работе. В принципе, они не имели бы ничего против Scrum как таковой, однако они сопротивляются любым переменам, которые создают опасность для сложившегося порядка вещей, для своего нынешнего статуса. Те, кому нравится статус-кво и кто активно сопротивляется переменам, способным нарушить статус-кво, называются консерваторами, или “твердолобыми”. Нередко они пытаются предотвратить перемены, сколачивая вокруг себя коалиции из числа недовольных.

В нижнем левом углу на рис. 6.2 располагаются так называемые последователи, которых устраивает существующий порядок вещей и которые оказывают пассивное сопротивление переменам. Обычно последователей не выводят из себя перспектива перемен, поэтому чаще всего они просто надеются, что Scrum окажется не более чем очередным модным увлечением, которое пройдет так же быстро, как и любые другие модные увлечения. Им достаточно продемонстрировать, что методология Scrum стала “новым статус-кво”.

Скептики

У Тэда не было иного выбора, кроме как переходить к использованию Scrum. Компания Тэда была поглощена другой компанией, и новые владельцы приказали немедленно приступить к внедрению Scrum. Таким образом, переход к Scrum не был личной инициативой Тэда. Более того, он сильно сомневался в целесообразности такого перехода. В частности, он сомневался в полезности ежедневных совещаний разработчиков, особенно учитывая то обстоятельство, что владелец продукта работал в 600 милях от места своего постоянного проживания. Можно ли создать столь сложный, новаторский и масштабный новый продукт, какой им предстояло разработать, без продолжительной фазы предварительного проектирования? Он мог еще уяснить полезность итеративной фазы разработки, но как можно обойтись без фазы предварительного проектирования, он понять не мог.

Тэд был типичным скептиком. Я был уверен в этом, видя его готовность согласиться с обоснованностью применения Scrum в других областях, для других технологий или в других средах, но только не в области, где работал он сам. Тэд полностью признавал пригодность Scrum для веб-проектирования, но сильно сомневался в пригодности Scrum для разработки научных приложений, которые были основной сферой деятельности его компании.

Как самый опытный член своей команды и один из самых опытных разработчиков в своей организации Тэд был типичным лидером мнений. Другим было очень важно

увидеть, как поведет себя Тэд в условиях, когда его заставят внедрять Scrum. Тэд демонстрировал здоровую меру скептицизма: нельзя ожидать от людей, что они с готовностью поменяют привычные для себя способы работы, если им будет отказано в праве задавать “трудные вопросы”; нельзя также ожидать, что люди полностью поверят в высокую эффективность Scrum, до тех пор, пока они не поработают в какой-либо Scrum-команде и на собственном опыте не убедятся в преимуществах этой методологии. Однако чувство неопределенности, которое испытывал Тэд, наполнило его настолько, что он начал сопротивляться переходу к Scrum. Это сопротивление проявлялось в мелочах — впрочем, в достаточно важных мелочах.

Не усматривая особого смысла в ежедневных совещаниях разработчиков, Тэд не настаивал на обязательном проведении каждого из них. В конце одного из совещаний он заявил: “Нам предстоит работа, которую мы вряд ли успеем завершить сегодня. Поэтому предлагаю провести следующее совещание не завтра, а послезавтра. Впрочем, следующее совещание можно провести и в любой другой день”. Подчас, хотя и далеко не всегда, его Scrum-мастеру удавалось опровергнуть эти доводы — в конце концов, Scrum-мастер тоже был новичком в этом деле.

К тому же, подобно многим скептикам, Тэд время от времени выступал с заявлениями в поддержку Scrum, но даже после этого продолжал работать по старинке. Например, он сказал, что поддерживает итеративный принцип работы, и заявил, что признает высокую ценность создания потенциально готового продукта к концу каждого спринта. Однако в действительности Тэд не верил, что все части их продукта можно спроектировать, закодировать и протестировать в рамках одного спринта. Таким образом, он по привычке заставлял команду брать на себя больший объем работы, чем они могли выполнить в течение одного спринта. Принятие на себя завышенных обязательств было его излюбленным способом добиваться, чтобы некоторые из функциональных возможностей реализовались в течение по меньшей мере двух спринтов.

Ниже перечислены некоторые инструменты, позволяющие преодолеть сопротивление скептиков.

- **Позвольте времени сделать свое дело.** Предоставив возможность процессу внедрения Scrum идти своим чередом, вы увидите, что у вас постепенно начнут накапливаться факты, свидетельствующие о преимуществах Scrum. Даже если эти свидетельства представляют собой набор разрозненных фактов, их наличие неминуемо снизит степень сопротивления, оказываемого скептиками.
- **Предоставьте людям возможность обучаться.** Сопротивление скептиков частично обусловлено отсутствием у них практического опыта применения соответствующей методологии или хотя бы возможности наблюдать применение этой методологии со стороны. Значительную помощь оказывает обучение — то ли на подготовительных курсах, то ли с помощью стороннего наставника, приглашенного для консультирования команды; скептики получают возможность увидеть, “как это делается”, на примере действий квалифицированного специалиста.
- **Попросите поделиться своими историями людей, имеющих опыт использования Scrum.** Если вы сами никогда не использовали Scrum, но кто-то из ваших друзей или знакомых уже имеет такой опыт, попросите их поделиться своими историями. Если другие команды в вашей организации уже имеют опыт успешного использования Scrum, попросите их рассказать о нем скептикам. Если в вашей организации еще нет команд, имеющих опыт успешного использования Scrum, пригласите специалистов, имеющих богатый опыт использования Scrum, со

стороны. Приглашение кого-либо из местных разработчиков программного обеспечения прочитать во время обеденного перерыва небольшой доклад об успехах его компании в деле применения Scrum убедит ваших собственных скептиков в полезности этой новой методологии больше, чем что-либо другое.

- **Выберите из числа скептиков наиболее яркого представителя.** В своей книге *Fearless Change: Patterns for introducing new ideas* Мэри Линн Маннс (Mary Lynn Manns) и Линда Райзинг (Linda Rising) предлагают выбрать кого-либо из сотрудников компании на роль “самого главного скептика”, т.е. наиболее яркого представителя этой категории (2004). Этот “самый главный скептик” должен быть влиятельным, уважаемым и коммуникабельным членом коллектива. Важно, однако, чтобы он не демонстрировал откровенной враждебности по отношению к переменам. “Самый главный скептик” должен участвовать во всех совещаниях и иметь право высказывать собственную точку зрения, а также указывать на проблемы, вызванные переходом к Scrum. Используйте эту информацию для разрешения сомнений, высказываемых “самым главным скептиком”. Поступая так, вы продемонстрируете свою способность мыслить широко и предотвратите перерастание любых сомнений в кризис.
- **Используйте скептика в качестве инструмента внедрения Scrum.** Назначьте скептика ответственным за какую-либо часть внедрения Scrum. Допустим, вы пытаетесь преодолеть сопротивление скептически настроенного тестера, который не верит, что тестирование может быть выполнено в рамках того же спринта, что и проектирование и программирование некоторой функции. Предложите этому тестеру указать пять способов, которые помогли бы его команде выполнить тестирование в рамках того же спринта. Вряд ли этому тестеру будет безразлично, справится ли он с вашим заданием: дело в том, что он будет бояться, что другой тестер, которому вы поручите это же задание, успешно с ним справится. Затем либо попросите команду испытать все пять способов, предложенных тестером, либо выберите вначале один-два способа, которые кажутся вам самыми многообещающими.
- **Способствуйте осведомленности людей.** Допустим, вы решили осуществить такие непростые перемены, как внедрение Scrum, потому что у вас возникла настоятельная необходимость в переменах. Возможно, у вашей организации появился новый конкурент; возможно, выпуск вашего предыдущего продукта занял слишком много времени; возможно, у вас появились другие, не менее важные причины для перехода к Scrum. Позаботьтесь о том, чтобы все, кто будут задействованы в этом переходе, знали о “светлом будущем”, которое ожидает их в случае успешного перехода.

ПРИМЕЧАНИЕ

Другие инструменты, помогающие преодолеть сопротивление скептиков, будут описаны в разделах о саботажниках, консерваторах и последователях. Хотя я не исключаю, что любой из этих инструментов можно применить к любой категории сопротивляющихся, я указывал эти инструменты применительно к той категории сопротивляющихся, в отношении которой эти инструменты представляются мне наиболее эффективными.

Что касается Тэда, то нам удалось преодолеть его скептицизм путем использования скептика в качестве инструмента внедрения Scrum. Мы справились с его пассивным сопротивлением итеративному подходу, перейдя к более коротким спринтам. Его команда использовала четырехнедельные спринты, но на каждом совещании по планированию спринтов рассматривала примерно шестинедельные объемы работ. Я сказал, что мы хотим испытать вариант двухнедельных спринтов, чтобы им проще было понять, какой объем работы они в действительности могут выполнить в течение одного спринта. Тэд не одобрил эту идею. На следующем же совещании по планированию спринтов Тэд, чтобы продемонстрировать нам нецелесообразность работы столь короткими спринтами, предложил своей команде взять смехотворно малый, на его взгляд, объем работы. Между тем именно такой объем оказался самым подходящим: впервые за все время его команде удалось выполнить все запланированное в течение одного спринта. После того как члены команды увидели, как важно выполнить в течение одного спринта всю запланированную работу, слабые попытки Тэда заставить команду брать на себя завышенные обязательства пресекались твердым намерением команды брать на себя лишь такой объем работы, который они действительно в состоянии выполнить в течение одного спринта.

Несмотря на то что использование скептика в качестве инструмента внедрения Scrum в случае Тэда сработало, самым важным фактором в преодолении его сопротивления было время. Чтобы изменить отношение Тэда к Scrum, потребовалось только время (и накопление “критической массы” свидетельств, подтверждающих возможность использования Scrum в этом случае).

Саботажники

Саботажника, вообще говоря, легко спутать со скептиком. Я сам однажды попался на крючок, проводя курс обучения гибкой методологии разработки в одной из компаний. Елена, один из слушателей этих курсов, задавала мне много дальних, интересных вопросов. Мне была неизвестна ее роль в этой организации, но поскольку многие относились к Елене с большим уважением, я понял, что она далеко не последний человек в этой организации, и поэтому не жалел времени для ответов на ее вопросы. Я рассуждал так: если я прав и Елена играет в этой организации роль “лидера мнений” и если бы мне удалось переубедить ее, последовательно опровергая все ее возражения, это имело бы решающее значение для изменения отношения к Scrum всей организации.

В конце первого дня занятий я встретился с директором, который, собственно, и пригласил меня провести курс обучения в этой компании. Мы поговорили с ним о том, как прошел первый день на курсах. Я поделился с директором своими соображениями насчет Елены. Директор сказал: “Мне следовало бы с самого начала предупредить вас о Елене. Должен сказать, что она ненавидит Scrum. Она возглавляет группу проектирования пользовательского интерфейса и встречает в штыки буквально все, что касается Scrum. Она противодействует Scrum все шесть месяцев, в течение которых мы внедряем эту методологию. Я был крайне удивлен, когда узнал, что она решила посещать ваши курсы”.

Итак, Елена была саботажником. Она активно сопротивлялась переходу к Scrum. Подобно большинству саботажников, она пыталась перетащить на свою сторону и остальных сотрудников компании. Несмотря на появление в этой компании все большего числа свидетельств, указывающих на то, что Scrum помогает создавать более

качественные продукты в более сжатые сроки, Елена продолжала настаивать на своем. Я спросил у нее напрямую, почему она столь ожесточенно противится переходу к Scrum. Она столь же прямо ответила: “Я занимаю лучшую каюту на «Титанике» и не намерена ее покидать!”

В дополнение к ряду инструментов, которые можно использовать для преодоления сопротивления скептиков, ниже описаны инструменты, продемонстрировавшие свою высокую эффективность в отношении саботажников.

См. также О спринтах и, в частности, о важности создания к концу каждого спринта чего-либо потенциально пригодного для передачи заказчику, подробно рассказывается в главе 14, “Спринты”.

- **Стремитесь достичь успеха.** Пока существуют сомнения относительно того, является ли Scrum эффективным подходом, саботажники будут использовать их для повышения градуса сопротивления. “Да, Scrum доказала свою эффективность в наших веб-проектах, — могут они соглашаться нехотя, — но она непригодна для наших серверных проектов”. Успешная реализация разнотипных проектов является самым надежным способом развенчания подобных аргументов.
- **Лишите саботажников даже малейших надежд.** Саботажники должны знать, что компания твердо намерена перейти к использованию Scrum. Любой признак слабости, и — подобно льву, пожирающему взглядом аппетитную антилопу — саботажники готовы броситься в атаку. Когда компании приходится сталкиваться с сопротивлением большого числа саботажников, заявление со стороны как можно более высоких эшелонов власти о неизменности курса на внедрение Scrum даст понять саботажникам, что сопротивляться бесполезно.
- **Устраните саботажников.** Если возможно, найдите какую-то другую команду, проект или подразделение и переведите саботажника туда. Если ваша организация не слишком мала или если она не совершает переход к Scrum по принципу “все вместе”, то вполне вероятно, что саботажник продолжит быть полезным членом команды в каком-нибудь другом месте — во всяком случае до тех пор, пока Scrum не начнет применяться в этой команде, к данному проекту или в данном подразделении.
- **Увольте саботажников.** Это, конечно, крайний вариант избавления от саботажника. Но если человек не согласен с направлением движения, которое выбрала для себя организация, и активно сопротивляется этому движению, то увольнение остается, пожалуй, единственным разумным выходом из ситуации.
- **Позаботьтесь о том, чтобы агитацией в пользу Scrum занимались подходящие люди.** В главе 4, “Движение в направлении гибкости”, вы ознакомились с идеей создания сообществ в поддержку усовершенствований как способа выявления и распространения в организации передового опыта использования Scrum. Совокупность сообществ “по интересам”, специализирующихся на определенных темах, может сыграть ключевую роль в создании силы, способной преодолеть сопротивление саботажников. Распространение в организации сведений об успехах сообществ практических пользователей Scrum ослабляет решимость саботажников продолжать свое сопротивление.

Елене повезло в том отношении, что она работала в крупной организации, руководству которой удалось перевести Елену в другое подразделение, продолжавшее занимать выжидательную позицию по отношению к Scrum. Со временем она привыкла к своей новой команде и стала полезным ее членом. Правда, и по сей день она втайне надеется, что со временем все вернется на круги своя и организация продолжит эффективно работать по старинке.

Консерваторы

Кэтрин работала на должности директора показателей и измерений в крупном подразделении одного провайдера финансовых данных. Мне сказали, что она поддерживает переход своего подразделения к Scrum. Правда, у нее было ко мне несколько вопросов. Ответы на них должны были, по ее собственным словам, помочь ей успешнее справляться с задачей сбора показателей о процессах и продуктах. Я охотно согласился ответить на эти вопросы, поскольку это моя профессиональная обязанность. К тому же обсуждение таких вопросов, как правило, позволяет мне узнать что-то новое. Я предвкушал, что беседа с Кэтрин даст мне возможность обсудить какие-нибудь креативные, новаторские показатели.

Как же я заблуждался! Кэтрин в совершенстве овладела искусством имитации. Она делала вид, что поддерживает переход к Scrum, и в то же время всеми силами пыталась сохранить существующий порядок. За три года до нашей встречи отдел разработки программного обеспечения в этой организации постоянно срывал сроки разработки новых продуктов и выдавал “на гора” продукты с множеством ошибок. Разумеется, такое программное обеспечение ни в коей мере не отвечало ожиданиям заказчиков. В то время Кэтрин, исполнявшая обязанности тест-менеджера, была в этой организации сравнительно новым сотрудником. Она способствовала внедрению новых процедур, которые существенно улучшили положение вещей. В результате команды начали укладываться в запланированные сроки разработки программного обеспечения (главным образом за счет умелого маневрирования календарными планами) и разрабатывать более качественные продукты (за счет создания отдельной группы тестирования, которая месяцами тестировала продукт, поступающий в ее распоряжение).

Учитывая личные заслуги Кэтрин в решении этих проблем, ее повысили в должности, назначив руководителем подразделения, которое, по сути, представляло собой отдел управления проектами (Project Management Office — PMO). После того как она рассказала мне о своем служебном списке, а также о том, как она помогала своей компании, предлагая руководству те или иные варианты усовершенствования процесса, я решил, что наконец-то обрел верного союзника в деле перехода к Scrum. Как бы не так! В конце концов я пришел к выводу, что на моем пути оказался человек, построивший для себя этакую персональную империю. (На ранних этапах такой результат вовсе не противоречил целям компании.) Однако к тому времени, когда я познакомился с Кэтрин, она уже была до такой степени удовлетворена своим текущим статусом, числом своих подчиненных и престижностью места, которое она занимала в компании, что не нуждалась в каких-либо переменах. Даже если бы кто-то предложил ей заведомо выигрышный вариант перемен, вряд ли Кэтрин согласилась бы на них.

Подобно другим консерваторам, Кэтрин сопротивлялась внедрению Scrum не потому, что усматривала в этой методологии какой-то изначальный изъян, а просто потому, что не желала что-либо менять в сложившемся порядке вещей. Кэтрин довольно

активно сопротивлялась переменам, но делала это так, что у нее всегда была возможность заявить об их поддержке.

Типичная тактика консерваторов, к которой охотно прибегала и Кэтрин, — препятствовать переменам за счет управления ресурсами. Это оказывается возможным потому, что консерваторы зачастую занимают посты в среднем и верхнем эшелонах власти, что обеспечивает им определенную степень управления ресурсами. В случае Кэтрин речь шла о контроле над таким совместно используемым ресурсом, как группа тестеров. Это давало Кэтрин возможность препятствовать переменам путем произвольного перебрасывания тестеров с одного проекта на другой. Для этого у нее всегда находились уважительные причины. Какому-то важному проекту требовалась дополнительные тестеры, другому проекту требовался тестер с определенными навыками и т.д. Тактика Кэтрин сводилась к тому, что ни одна команда не сохраняла постоянный состав с самого начала и до завершения проекта, и у многих Scrum-команд не было тестера на протяжении нескольких первых спринтов.

Многие из инструментов, используемых для преодоления сопротивления саботажников, можно применять и против консерваторов. Но против консерваторов можно использовать и дополнительные инструменты.

- **Предоставьте консерваторам стимулы к переменам.** Консерваторы привязаны к существующему порядку вещей из-за выгод (материальных или нематериальных), которые доставляет им этот порядок вещей. Если оказывается, что консерваторы противятся переменам, проанализируйте все стимулы, которые существуют в данной организации, и позаботьтесь о том, чтобы каждый из них оставался в действии и после перехода к Scrum. Речь идет не об одних лишь финансовых стимулах. Нефинансовые стимулы (например, продвижение по службе или иные формы общественного признания заслуг работника) также нужно принимать во внимание. Если наличие у вас большого числа подчиненных обеспечивает вам влиятельность в организации, то не приходится сомневаться, что люди будут сопротивляться переменам, которые влекут за собой сокращение числа их подчиненных.

См. также В главе 20, “Кадры, техническое обеспечение и отдел управления проектами”, приводятся советы, касающиеся многих вопросов управления кадрами.

- **Вызовите у людей неудовлетворенность существующим порядком вещей.** Консерваторы привязаны к существующему положению. Они сопротивляются переходу к Scrum не потому, что видят в этой методологии какие-то недостатки, а потому, что их устраивает статус-кво. Попытайтесь вызвать у людей неудовлетворенность существующим порядком вещей. Я, конечно, не имею в виду искусственное создание кризиса, но если таковой действительно возникнет, укажите людям на причины его возникновения. Если снижается доля рынка, принадлежащая вашей компании, люди должны об этом знать. Если обращения за технической поддержкой становятся все более частыми, сообщите об этом людям. Если в последнем отраслевом информационном бюллетене опубликованы хвалебные материалы о продукте вашего конкурента, разместите копии соответствующей статьи везде, где с ней может ознакомиться как можно большее число

сотрудников вашей компании. Это соответствует рекомендации Стюарта Таббса (Stewart Tubbs), автора учебника по взаимодействию малых групп: “Прозорливый руководитель всегда стремится изыскать способы, с помощью которых его организация может постоянно совершенствоваться. Он постоянно стремится изыскать способы, с помощью которых его организация может повышать свою эффективность, и стремится использовать эти идеи как способ вызвать у людей неудовлетворенность существующим порядком вещей” (2004, 352).

- **Попытайтесь распознать одолевающие людей страхи и сделайте все от вас зависящее, чтобы их развеять.** Консерваторы сопротивляются переменам отчасти из-за неопределенности по поводу своего будущего в условиях применения Scrum. Их, как правило, вполне устраивает положение, которое они занимают в своей организации. Страх перед неопределенным будущим может целиком овладеть человеком. Как изменится моя роль? Как меня будут оценивать? Как эти перемены повлияют на мою карьеру? Эти и подобные им вопросы очень тревожат консерваторов. Если вы в состоянии развеять сомнения, терзающие консерваторов, постараитесь сделать это как можно скорее. Если же вы не можете ответить на такие вопросы, честно сознайтесь в этом, но дайте слово, что ответите на них при первой же возможности (если, конечно, это в ваших силах и если вы цените труд данного консерватора). Можете также попытаться развеять страхи, разъяснив консерватору, *чего* вы ожидаете не только от него самого, но и от его товарищей по работе.

Что же касается случая с Кэтрин, то я вместе с вице-президентом ее компании (Кристиной) подыскал для Кэтрин подходящую роль в новой организации (т.е. в организации, которая возникла после внедрения Scrum). Мы рассказали ей о своей уверенности в том, что ее прошлый позитивный опыт (помощь компании в значительном усовершенствовании процессов) наверняка позволит ей сыграть ключевую роль в оказании помощи компании в новых условиях. Кристина прояснила для Кэтрин ее роль в новой организации. К сожалению, ощущение собственной значимости Кэтрин и понимание ею своего места в организации настолькоочно прочно соединились в ее сознании с процессом, который она в свое время помогла внедрить в компании, что она уже просто не могла представить, что вместо этого процесса будет использоваться какой-то другой. В конце концов Кэтрин пришлось уволиться.

Последователи

Подобно консерваторам, последователи в большей степени противятся изменению статус-кво, чем собственно внедрению Scrum. Однако, в отличие от консерваторов, они оказывают пассивное сопротивление переменам. Декстер, программист среднего уровня в компании, занимающейся электронной торговлей, был типичным последователем. Он задавал вопросы, характерные для скептика, но в этих вопросах всегда скрывался подтекст, суть которого сводилась к тому, что Scrum — это плохая идея. В случаях, когда скептик спросил бы “Какова эффективность Scrum применительно к проектам, в которых доведение пользовательского интерфейса до идеального состояния абсолютно критично?”, Декстер спрашивал: “Методология Scrum неэффективна в случае, когда доведение пользовательского интерфейса до идеального состояния абсолютно критично, не так ли?”

Я помню разговор с Декстером, в ходе которого он спросил у меня, сколько раз я намерен посетить его компанию. “По плану я должен побывать у вас в июле и октябре” — ответил я. Разговор состоялся в июне.

“А после этого не собираетесь бывать у нас?” — спросил он.

“Возможно, побываю. Впрочем, мы еще не составляли планы на более отдаленные сроки”.

“Прекрасно. В таком случае надеюсь, что мы покончим со всем этим к концу года”.

На меня произвело впечатление его энтузиазм, но мне показалось, что предполагаемые им темпы внедрения Scrum чрезесчур высоки, учитывая величину его компании. “Как знать, — засомневался я. — Возможно, кое-что придется доделывать уже в следующем году. У вас ведь еще не все даже начали работать спринтами. Правда, не исключено, что в следующем году вам не понадобится моя помощь”.

“О, нет! — запротестовал Декстер. — Я вовсе не то имел в виду. Я имел в виду, что к тому времени у нас уже начнут внедрять очередную, еще более модную, методологию. После завершения очередного сезона рождественских распродаж мы всегда начинаем внедрение очередного нового процесса”.

Еще никто не рассказывал мне об этих ежегодных изменениях процесса до моего первого визита в эту компанию, но учитывая традицию внедрения в начале каждого года нового процесса, мне уже не казалось удивительным, что Декстер занял выживательную позицию в отношении Scrum. Вообще говоря, такого подхода придерживаются многие последователи, рассуждая в том духе, что за этим изменением вскоре последует очередное изменение, а потому имеет смысл пропустить пару-тройку таких изменений.

Декстер как таковой не является серьезным препятствием для успешного перехода к Scrum. Но если в вашей организации наберется критическая масса таких декстеров, они могут помешать успешному переходу к Scrum. К счастью, последователи, как правило, не очень-то решительны в своем сопротивлении. Они оказывают незначительное, главным образом пассивное, сопротивление, надеясь, что перемены как-то “рассосутся” сами собой. Помимо описанных выше инструментов, существует еще несколько способов воздействия на последователей.

- **Измените состав команды.** Одни товарищи по работе оказывают на нас положительное влияние, а другие — отрицательное. Изменение состава команды почти наверняка изменит характер сопротивления. Замена вечно недовольного, ворчливого саботажника скептиком может изменить мотивации последователей к сопротивлению.
- **Похвалите правильное поведение.** Вместо того чтобы сосредоточивать свое внимание на изменении поведения последователей, похвалите те или иные аспекты правильного поведения, если вы обнаружите таковые у “очернителя” или у сторонника перемен. Последователи (по крайней мере, некоторые из них) заметят это и ослабят свое сопротивление.
- **Вовлеките их в процесс внедрения Scrum.** Эффективным способом снижения сопротивления пассивных последователей является их вовлечение в процесс внедрения Scrum. Например, можно предложить последователю вступить в сообщество в поддержку усовершенствования организации, занимающееся автоматизацией тестирования исходного кода применительно к некоему сложному

“унаследованному приложению” (т.е. уже существующему приложению, которое нужно заменить, модифицировать, включить или перенести в новое окружение), или поработать совместно с другими над созданием для группы сбыта презентации, посвященной влиянию Scrum на вашу способность правильно планировать сроки, закладываемые в контракты.

- **Сами станьте примером правильного поведения.** Последователям требуется образец для подражания. Позаботьтесь о том, чтобы в качестве такого образца они выбрали человека, демонстрирующего правильное поведение в отношении гибкой методологии разработки (возможно, выступите сами в качестве такого образца). Учитывая, что важным элементом методологии Scrum является сотрудничество, попытайтесь продемонстрировать это своим взаимодействием с другими членами команды.
- **Выявите подлинное препятствие.** Руководствуясь моделью, описанной в главе 2, “ADAPТация к Scrum”, определите, не сопротивляется ли последователь из-за недостаточной осведомленности, нежелания или неспособности использовать Scrum. Затем помогите ему устраниТЬ это препятствие. Если последователь недостаточно осведомлен о причинах перехода к Scrum, поговорите с ним и разъясните ему эти причины. Если последователю не хватает умения применять на практике гибкую методологию разработки, изыщите возможность организовать его совместную работу с человеком, который поможет ему овладеть соответствующими навыками.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Выявите в своей организации пять человек, оказывающих самое ожесточенное сопротивление переменам.
- Попытайтесь определить, к какой категории (скептик, саботажник, консерватор или последователь) принадлежит каждый из этой пятерки.
- Выявите какую-то одну меру, которую вы можете принять, чтобы ослабить или устранить сопротивление каждого из этой пятерки. Попытайтесь найти такую меру, которая окажется эффективной в отношении сразу нескольких членов этой пятерки.
- Оцените, правильно ли вы создали условия для перехода к Scrum, обеспечив прежде всего осведомленность и вызвав у людей желание. В случае необходимости примите эти меры еще раз.

Сопротивление как полезный предупреждающий сигнал

Приступая к сложным преобразованиям в крупной организации, нужно быть готовым к сопротивлению. Важно, какой будет реакция на него руководства организации. Вот что пишет Пол Лоуренс, которого мы уже цитировали в начале этой главы.

Когда возникает сопротивление, его не следует воспринимать как явление, которое нужно преодолеть. Напротив, сопротивление следует воспринимать как

полезный предупреждающий сигнал, свидетельствующий о каком-то неблагополучии. Если воспользоваться грубой аналогией, то признаки сопротивления в социальной организации полезны так же, как боль в теле, свидетельствующая о том, что в организме не все в порядке. Сопротивление, подобно боли, не указывает нам точно на источник и причины неблагополучия, а лишь свидетельствует о самом факте неблагополучия. В попытках преодолеть такое сопротивление заключено не больше смысла, чем в попытках принять болеутоляющую таблетку, не диагностировав причину недуга. Следовательно, когда возникает сопротивление, нужно внимательно выслушать людей, чтобы уяснить источник и причины неблагополучия. Вместо пространных рассуждений требуется скрупулезное исследование возникших затруднений (1969, 9).

Не следует допускать, чтобы устранение сопротивления превратилось в борьбу “нас” против “них”. Ваша подлинная цель — создать у людей ощущение неотвратимости перехода к Scrum и бессмыслицы сопротивления этому переходу. Однако при этом с вашей стороны было бы непозволительно игнорировать чувства и реакции людей и насилием навязывать им новую методологию. Когда кто-то из сотрудников сопротивляется переменам, эффективный лидер воспринимает этого сотрудника не как проблему, которую нужно решить, а как человека, которого нужно понять (Nicholson, 2003).

Дополнительная литература

Bridges, William. 2003. *Managing transitions: Making the most of change*. 2nd ed. Da Capo Press.

Автор этой книги является экспертом по общему управлению преобразованиями, а не специалистом по разработке программного обеспечения. Его книга представляет собой классическое пособие по проведению преобразований и содержит множество полезных сведений о том, как избавиться от прошлого. В ней также представлена информация о преодолении так называемой “нейтральной зоны”, т.е. отрезка времени, когда от старого подхода мы уже отказались, а новый еще не укоренился.

Emery, Dale H. 2001. Resistance as a resource. *Cutter IT Journal*, October.

Эмери знакомит читателей с оригинальной точкой зрения на сопротивление. Автор полагает, что сопротивление можно рассматривать как реакцию на инициативу, связанную с осуществлением тех или иных преобразований. Автор утверждает, что эта реакция заключает в себе полезную информацию о человеке, оказывающем сопротивление, и позволяет определить способ, с помощью которого этого человека можно вовлечь в процесс преобразований. В статье указаны четыре фактора, которые обуславливают сопротивление переменам.

Manns, Mary Lynn, and Linda Rising. 2004. *Fearless Change: Patterns for introducing new ideas*. Addison-Wesley.

В этой книге представлено 48 шаблонов, которые можно применить к любой инициативе, связанной с осуществлением тех или иных преобразований. В их числе

есть как достаточно широко известные (например, “готовьте еду” [“do food”]), так и не очень известные (например, назначение “главного скептика”), а также многие другие, которые помогают устранять сопротивление переменам.

Reale, Richard C. 2005. *Making change stick: Twelve principles for transforming organizations*. Positive Impact Associates, Inc.

Кое-какие из 12 предложений, содержащихся в этой небольшой книжице, можно использовать для преодоления сопротивления переменам. Особенно полезны, на наш взгляд, разделы о методах, с помощью которых можно побудить людей действовать в нужном вам направлении и помочь им преодолеть страх, заставляющий их сопротивляться переменам. Другие предложения (например, внесение соответствующих изменений в корпоративную культуру) слишком масштабны, чтобы их можно было надлежащим образом осветить на 12 страницах, посвященных рассмотрению этого вопроса.

Глава 7

Новые роли

Как указывалось в предыдущей главе, отдельные команды и целые организации сопротивляются внедрению Scrum по многим причинам. Одной из них является неправильное понимание людьми новых ролей, которые им приходится выполнять при реализации Scrum-проекта. Роли Scrum-мастера (руководителя Scrum-проекта) и владельца продукта являются новыми ролями, которым нет точных аналогов в том варианте организации, который существовал до перехода к Scrum. Как правило, организация, внедряющая у себя Scrum, испытывает проблемы с назначением на эти роли подходящих специалистов. До тех пор, пока люди не поймут, что означают эти новые роли и какие из специалистов обладают квалификацией, необходимой для успешного их исполнения, будет очень нелегко подыскать подходящих людей.

В этой главе я опишу новые роли — Scrum-мастер и владелец продукта. Мы рассмотрим обязанности людей, исполняющих каждую из них, характеристики идеальных кандидатов на эти роли и способы преодоления типичных проблем, порождаемых этими ролями.

Роль Scrum-мастера

О роли Scrum-мастера (ScrumMaster) в устраниении препятствий на пути прогресса команды написано немало (Schwaber and Beedle 2001, Schwaber 2004). Большинство Scrum-мастеров быстро осваивает эту часть своих функциональных обязанностей. На чем многие спотыкаются — особенно в течение важнейших первых 6–12 месяцев использования Scrum, — так это на отношениях со своими командами. (Вот почему мы уделяем этой теме особое внимание.)

Многим из тех, кто впервые оказался в роли Scrum-мастера, приходится преодолевать очевидное противоречие между ролью Scrum-мастера как лидера, который служит своей команде, и как человека, не обладающего властными полномочиями. Это кажущееся противоречие исчезает, когда мы осознаем, что, несмотря на отсутствие у Scrum-мастера каких-либо властных полномочий в отношении членов Scrum-

команды, у него в подчинении находится *процесс*. Несмотря на то что Scrum-мастер не может сказать “Вы уволены,” он *может* сказать “Я решил, что в следующем месяце мы будем работать двухнедельными спринтами”.¹

Scrum-мастер призван помогать команде в использовании Scrum. Эта помощь напоминает действия спортивного тренера, который помогает спортсменам соблюдать спортивный режим и поддерживать на надлежащем уровне спортивную форму. Хороший тренер создает мотивации и в то же время следит за тем, чтобы его подопечные не увиливали от выполнения тяжелых физических упражнений. Однако власть тренера ограничена. Он не может заставить вас выполнять то или иное физическое упражнение, которое вы не желаете выполнять. Вместе с тем тренер напоминает вам о ваших целях и о том, как вы намеревались их достигать. Тренер обладает властью в той мере, в какой этой властью наделил его клиент. Примерно то же самое можно сказать о Scrum-мастерах: они обладают властью, но этой властью их наделяет команда.

См. также Подробнее о смысле программного продукта, потенциально готового к отправке заказчику, рассказывается в главе 14, “Спринты”.

Scrum-мастер может сказать команде: “Итак, к окончанию каждого спринта мы должны создавать программный продукт, потенциально готовый к отправке заказчику. На этот раз нам не удастся создать такой продукт. Что мы можем сделать для того, чтобы подобная ситуация не повторилась в следующем спринте?” Это пример проявления власти Scrum-мастера над процессом. Если команде не удалось к окончанию текущего спринта изготовить программный продукт, потенциально готовый к отправке заказчику, значит, произошел какой-то сбой в работе команды. Но поскольку власть Scrum-мастера распространяется только на процесс, тот же Scrum-мастер не имеет права сказать: “Поскольку нам не удалось к окончанию текущего спринта изготовить программный продукт, потенциально готовый к отправке заказчику, я хочу, чтобы Тод просмотрел весь код до того, как он будет окончательно принят”. Возможно, было бы не так уж плохо, если бы Тод просмотрел весь код, но Scrum-мастер не имеет права принять такое решение, поскольку оно выходит за пределы его властных полномочий и вторгается в сферу полномочий команды.

Поскольку Scrum-мастер уполномочен лишь контролировать соблюдение процесса командой, его роль может оказаться более трудной, чем роль типичного руководителя проекта. Руководители проектов зачастую могут сказать: “Делайте так, потому что я так решил”. Случай, когда Scrum-мастер может сказать что-то подобное, связанный лишь с соблюдением методологии Scrum.

Качества хорошего Scrum-мастера

Современные хирурги — это высокообразованные и высококвалифицированные специалисты, многолетнее формальное образование которых сопровождается продолжительной интернатурой. Но так было не всегда. Пит Мур (Pete Moore) писал, что “первые хирурги плохо знали анатомию, но пользовались доверием пациентов,

¹ В идеальном случае Scrum-мастер пытается добиться, чтобы члены команды сами приняли это решение. Но если команда не в состоянии это сделать, полномочий Scrum-мастера в отношении процесса вполне достаточно для того, чтобы принять такое решение самостоятельно.

потому что располагали острыми инструментами и сильными руками. Зачастую они делали хирургические операции в свободное от основной работы время. Их основной работой была работа в местных парикмахерских или кузницах” (2005, 143).

Многие организации выбирают Scrum-мастеров примерно по такому же принципу. Правда, вместо кандидатов с острыми инструментами и сильными руками организации ищут кандидатов с опытом управленческой работы. Однако, ознакомившись с некоторыми тонкостями Scrum, организации начинают понимать, что при выборе Scrum-мастеров нужно учитывать много других факторов. Чтобы уберечь вас от назначения на роль Scrum-мастеров лиц, единственным достоинством которых является наличие у них острых инструментов и сильных рук, далее я описываю шесть качеств, которые, на мой взгляд, являются общими для лучших Scrum-мастеров, с которыми мне приходилось когда-либо работать.

Чувство ответственности

Хороший Scrum-мастер способен и готов брать на себя ответственность. Я не хочу тем самым сказать, что Scrum-мастера отвечают за успех проектов. Ответственность за успех проекта несет вся команда. Однако именно Scrum-мастер обязан обеспечить максимальную производительность команды и именно он отвечает за оказание помощи членам команды в деле внедрения и использования Scrum. Как отмечалось выше, Scrum-мастер отвечает за это, не прибегая к использованию властных полномочий, которые могли бы облегчить ему решение этой задачи.

См. также Подробнее об ответственности команды в целом читайте в главе 11, “Организация коллективного труда”.

Работу Scrum-мастера можно сравнить с работой дирижера оркестра. И тот, и другой должны в реальном времени руководить действиями коллектива талантливых исполнителей, которые объединились для создания чего-то такого, что никто из них не смог бы осуществить в одиночку. Вот что говорит о своей роли дирижер Бостонского симфонического оркестра Кейт Локхарт (Keith Lockhart): “Людям кажется, что дирижер похож на Наполеона: он должен возвышаться над толпой и проявлять свою власть. Но я вовсе не похож на человека, упивающегося властью. Я больше похож на человека, упивающегося ответственностью” (Mangurian, 2006, 30). Точно так же деятельность хорошего Scrum-мастера основывается на высокой ответственности, причем специфика ответственности Scrum-мастера заключается в том, что у него нет власти над людьми.

Скромность

Хороший Scrum-мастер отличается скромностью. Он не выпячивает свое “я”. Он может гордиться своими достижениями (чаще всего втайне от других), но суть его гордости можно выразить словами “Вот как я помог людям добиться успеха”, а не словами “Вот каковы мои достижения”. Скромный Scrum-мастер — это человек, который отдает себе отчет в том, что его статус в компании определяется не наличием служебного автомобиля или бронированием престижного места для его персонального автомобиля на парковочной площадке компании (например, неподалеку от входа в штаб-квартиру компании). Вместо того чтобы постоянно демонстрировать окружающим

собственную значимость и незаменимость, скромный Scrum-мастер готов делать все от него зависящее, чтобы помочь команде добиться своей цели. Он признает высокую ценность каждого члена команды и на личном примере показывает, что именно так должны рассуждать все члены команды.

Готовность к сотрудничеству

Хороший Scrum-мастер способствует формированию в своем коллективе культуры сотрудничества. Scrum-мастер должен заботиться о том, чтобы члены его команды не стеснялись ставить волнующие их вопросы на открытое обсуждение и могли рассчитывать на его поддержку. Хороший Scrum-мастер каждым своим словом и действием способствует созданию в команде атмосферы сотрудничества. При возникновении спорных ситуаций он побуждает свою команду к принятию решений, приемлемых для каждого, кого эти решения касаются, и стремится избегать решений, которые приводят к появлению победителей и проигравших. Хороший Scrum-мастер стремится подавать пример такого поведения, сотрудничая с другими Scrum-мастерами в своей организации. Кроме того, он устанавливает сотрудничество как стандарт и норму поведения в команде, категорически пресекая поведение, подрывающее сотрудничество (даже если другие члены команды готовы мириться с таким поведением).

Полная самоотдача

Несмотря на то что функции Scrum-мастера не всегда обязывают его отдавать *все* свое рабочее время деятельности на этом посту, Scrum-мастером должен быть человек, готовый полностью отдаваться работе. Он должен ощущать столь же высокий уровень ответственности за конкретный проект и цели текущего спринта, как и каждый из членов команды. В рамках этой ответственности хороший Scrum-мастер стремится как можно более оперативно устранять препятствия, возникающие на пути успешного выполнения проекта, не откладывая их устранение “на потом”. Разумеется, далеко не всегда удается устранять препятствия в тот же день, когда они возникают. Например, чтобы убедить какого-то менеджера выделить для команды тот или иной важный ресурс, может понадобиться несколько дней. Однако в целом, если команда видит, что Scrum-мастер не особенно торопится с устранением препятствий, она обязана напомнить ему о его ответственности.

Свою высокую ответственность и готовность к самоотдаче Scrum-мастер может продемонстрировать, оставаясь в этой роли на все время выполнения проекта. Выйдя из проекта на полпути, Scrum-мастер окажет команде плохую услугу.

Умение влиять на других

Эффективный Scrum-мастер умеет влиять на других (не только на членов команды, но и на посторонних). Для начала ему придется убедить членов команды попробовать воспользоваться Scrum или проявить большую готовность к сотрудничеству друг с другом. Затем ему, возможно, придется убедить членов команды испытать какой-то новый технический прием (например, разработку программного обеспечения на основе составления тестов или парное программирование). Scrum-мастеру следует знать, как влиять на людей, не прибегая к простым решениям наподобие “делайте так, потому что я так считаю нужным”.

Большинству Scrum-мастеров приходится время от времени оказывать влияние и на посторонних. Например, Scrum-мастера могут потребоваться убедить команду, пользующуюся традиционным методом разработки, помочь в реализации проекта, выполняемого Scrum-командой. Или, например, Scrum-мастера могут потребоваться повлиять на начальника службы технического контроля, чтобы тот откомандировал для полноценного участия в реализации проекта, выполняемого Scrum-командой, нескольких квалифицированных тестеров.

Все Scrum-мастера должны уметь разумно использовать свое личное влияние, но идеальный Scrum-мастер должен в определенной степени владеть искусством корпоративной политики. Термин “корпоративная политика” иногда имеет пренебрежительный оттенок. Однако Scrum-мастер, который знает, кто принимает решения в данной организации, как именно эти решения принимаются, какие коалиции сложились в данной организации и тому подобное, может оказаться для команды весьма ценным активом.

Наличие обширных познаний

У эффективного Scrum-мастера должно быть не только четкое понимание Scrum и значительный опыт работы с этой методологией, но и большой багаж технических, экономических и прочих специализированных знаний, которые помогут его команде достичь поставленной цели. Лафасто и Ларсон, изучив практику успешных команд и их лидеров, пришли к выводу, что “глубокое и детальное знание принципов функционирования того, над чем работает команда и ее лидер, повышает вероятность того, что лидер поможет своей команде разобраться в самых тонких технических проблемах, с которыми ей придется столкнуться” (2001, 133). Хотя Scrum-мастера совсем не обязательно должны быть высококлассными специалистами в области маркетинга или программирования, они обязаны знать в достаточной степени и то, и другое, чтобы эффективно управлять своей командой.

Технические лидеры в качестве Scrum-мастеров

Утверждение о том, что Scrum-мастера должны иметь солидные технические знания, вовсе не означает, что мы можем, не мудрствуя лукаво, назначать Scrum-мастерами технических лидеров. Вообще говоря, поскольку Scrum-команды — это самоорганизующиеся коллективы, такой роли, как технический лидер, в компании не должно быть в принципе. Однако при переходе к Scrum у руководства компании нередко возникает соблазн использовать в качестве Scrum-мастеров бывших технических лидеров. Подобный соблазн питается надеждой руководства на то, что успешные лидеры смогут оказывать столь же благотворное влияние и на Scrum-команду, и на ее продукт, какое они оказывали в прошлом на возглавляемые ими коллективы. Несмотря на то что некоторые из таких лидеров становятся впоследствии превосходными Scrum-мастерами, ни в коем случае не следует доверять человеку роль Scrum-мастера только потому, что в прошлом он был хорошим техническим лидером (или замечательно проявил себя в какой-либо другой роли).

Несколько лет назад я организовал в одной компании курс начального обучения Scrum с целью помочь руководству компании решить, будут ли они внедрять у себя эту методологию. Спустя две недели один из руководителей компании позвонил мне и сказал, что мой курс их убедил и они готовы внедрить у себя Scrum. Более того,

он звонил мне во время совещания, на котором решалось, кто у них будет первыми Scrum-мастерами. Именно по этому вопросу они хотели посоветоваться со мной. Затем мой собеседник сказал: “Мы не можем уделить слишком много времени этому обсуждению. У нас только один вопрос: «Может ли технический руководитель каждой из команд стать Scrum-мастером своей команды?» Просто ответьте: да или нет.” Я начал отвечать: “Да, технический руководитель может, вообще говоря, стать Scrum-мастером своей команды, но...” и собирался разъяснить своему собеседнику риски, связанные с таким подходом. Но тот, не дослушав мою мысль, поблагодарил меня и положил трубку.

Заглянув в эту компанию через два месяца, я буквально с порога услышал следующий сердитый вопрос: “Почему вы посоветовали нам назначить Scrum-мастерами технических руководителей?” Да разве я советовал им это? Ясное дело, они столкнулись с проблемами, о которых я пытался их предупредить. Когда возникли проблемы, они поняли, что наличие солидных технических познаний — это *лишь одно* из качеств, которыми должен обладать Scrum-мастер.

Один из рисков, связанных с использованием бывшего технического руководителя в качестве Scrum-мастера, заключается в том, что задачей таких лидеров является управление действиями подчиненных. Более того, члены команды просто обязаны выполнять решения, принимаемые их руководителем. Но поскольку хороший Scrum-мастер не принимает решения вместо своей команды, прошлый опыт технического руководителя в качестве лица, принимающего решения, может лишь препятствовать переходу к Scrum.

Второй риск, связанный с превращением технических руководителей в Scrum-мастеров, заключается в том, что у этих лидеров зачастую нет необходимых навыков взаимодействия с людьми. Конечно, технические руководители обладают определенными навыками взаимодействия с сотрудниками, но Scrum-мастера должны быть посредниками и помощниками, способными направлять действия самоорганизующихся команд (в отношении которых у Scrum-мастеров нет властных полномочий) в нужное русло. Подобные мысли разделяет и автор книги *Collaboration Explained* Джин Табака (Jean Tabaka).

Я работаю главным образом со Scrum-командами и пришел к такому выводу: команды, которые испытывают серьезные проблемы, как правило, используют в качестве Scrum-мастеров либо менеджеров проектов, придерживающихся командного стиля управления, либо технических лидеров, ориентированных на принятие решений. Без поддерживающего режима управления Scrum-командой, при котором лидер выступает в роли помощника, а не начальника, внедрение гибкой методологии разработки окажется лишь слабым прикрытием для беззывных, неспособных к самостоятельному принятию решений и деморализованных команд (2007, 7).

Все сказанное выше совсем не означает, что технические лидеры ни при каких обстоятельствах не должны рассматриваться в качестве кандидатов на роль Scrum-мастеров. Я призываю лишь проявлять осторожность, помнить об этих проблемах и не исходить из того, что *все* технические лидеры в вашей организации станут великими Scrum-мастерами. Возможно, наилучший способ оценить технического руководителя как кандидата на роль Scrum-мастера — рассмотреть, как этот человек воспользовался властью лидера, которой наделял его статус технического руководителя.

Лидеры, придерживавшиеся в прошлом командного стиля управления (“будет только так, как я сказал”), вряд ли смогут стать хорошими Scrum-мастерами. С другой стороны, лидеры, которым была предоставлена возможность использовать командный стиль управления, но которые предпочитали действовать методом убеждения, а не принуждения, вполне способны стать хорошими Scrum-мастерами.

Scrum-мастера: собственные или приглашенные

Нам придется рассмотреть еще один типичный вопрос, касающийся выбора подходящих кандидатов на роль Scrum-мастера. Какой вариант лучше: собственные или приглашенные Scrum-мастера? С точки зрения долгосрочной перспективы здесь и думать нечего: организации очень важно располагать собственными квалифицированными Scrum-мастерами. Иными словами, в долгосрочной перспективе нежелательно пользоваться услугами Scrum-мастеров, приглашенных со стороны.

Но освоить новую методологию нелегко, если в вашем распоряжении нет человека, который мог бы продемонстрировать ее на практике. Трудно научиться руководить действиями людей, не прибегая к командному стилю управления; нелегко понять, когда и как побуждать команду к внедрению новых методов проектирования, в какие моменты вмешиваться в действия команды и т.д. Следовательно, многим организациям выгодно поначалу пригласить в качестве Scrum-мастера какого-либо стороннего консультанта. Этот консультант может играть роль Scrum-мастера команды, но он должен выступать и в роли наставника потенциальных Scrum-мастеров, которых может выдвинуть из своих рядов эта команда. Таким образом, организация сможет воспитать собственные кадры Scrum-мастеров.

Ротация Scrum-мастеров

Некоторые команды, испытывающие проблемы с выбором эффективного Scrum-мастера, приходят к выводу, что оптимальной стратегией в этом случае является ротация кандидатов на роль Scrum-мастера, выдвигаемых из рядов самой команды. Лично я не являюсь сторонником такого подхода, поскольку считаю, что он демонстрирует неуважительное отношение к задачам, которые приходится решать Scrum-мастеру, и значимости его роли. Члены моей семьи по очереди делают уборку в комнатах и моют посуду. Любой из нас может с этим справиться. Однако готовить еду в нашей семье может далеко не каждый. Лучше всех готовит моя жена. Мы хотим, чтобы наша пища была вкусной, поэтому доверяем ее приготовление только моей жене. Ротация здесь исключена. Если вы хотите, чтобы ваша Scrum-команда добивалась наилучших результатов, то постарайтесь все же избегать ротации людей на столь ответственной должности, как Scrum-мастер.

Однако иногда ротация оказывается желательным вариантом. Наиболее типичным таким случаем является ситуация, когда вы хотите предоставить людям возможность обучаться. Например, если члены команды никак не могут понять, в чем именно заключаются обязанности Scrum-мастера, каждому из них было бы полезно побывать в шкуре Scrum-мастера. Лишь так они поймут, что значит быть Scrum-мастером. Аналогично, если команда выявит в своих рядах четыре или пять подходящих кандидатов на роль Scrum-мастера, то было бы неплохо провести ротацию среди этих кандидатов, предоставив каждому из них возможность проявить себя в этой роли. Затем, рассмотрев результаты работы каждого из них, команда сможет выбрать самого достойного.

Боб Шац (Bob Schatz) и Ибрагим Абдельшрафи (Ibrahim Abdelshafi) из Primavera Systems указывают еще одну причину возможной пользы от ротации.

Со временем команда может начать воспринимать Scrum-мастера как своего менеджера. Человек, выступающий в роли Scrum-мастера, как правило, обнаруживает это и с готовностью удовлетворяет потребность команды. Результатом является провал в практике самоуправления команды. Ротация в начале каждого спринта людей, выступающих в роли Scrum-мастера, приводит к распределению этой роли, делает ее совместной обязанностью команды и способствует достижению баланса власти (2006, 145).

Итак, несмотря на то что ротация людей, выступающих в роли Scrum-мастера, вполне возможна, я рекомендовал бы прибегать к ней в силу особых причин — и только на определенное время. Ротация не должна становиться постоянной практикой. Дело в том, что превращение ротации в постоянную практику порождает слишком много проблем, включая следующие.

- Человек, которому поручают выполнять роль Scrum-мастера в порядке ротации, обычно во время спринта выполняет другие обязанности, не имеющие отношения к этой роли, причем эти “другие обязанности” имеют для него более высокий приоритет.
- Трудно обучить достаточное число людей для эффективного исполнения этой роли.
- Кое-кто будет использовать время своего пребывания на посту Scrum-мастера для внесения тех или иных изменений в процесс.
- Назначение человека на роль Scrum-мастера на один-два спринта вовсе не приводит автоматически к тому, что человек начинает ценить эту роль. Это, в свою очередь, может привести к появлению Scrum-мастеров, которые считают Scrum не более чем досадным недоразумением.

Преодоление типичных проблем

Ниже перечислены типичные проблемы, с которыми вы можете столкнуться, пытаясь добиться, чтобы у каждой команды был надлежащий Scrum-мастер, и способы их решения.

Роль Scrum-мастера достается человеку, не пригодному для ее успешного исполнения. Подчас решение о том, кому достанется роль Scrum-мастера, принимается без вашего участия: просто кто-то говорит: “Эту роль буду выполнять я” — и дело с концом. Иногда это не так уж плохо: в конце концов, хорошие Scrum-мастера — это люди, готовые добровольно, без подсказок и принуждения, взять на себя дополнительную ответственность. Но что если человек, готовый добровольно взять на себя эту роль, не подходит для нее? Ваша реакция на это будет зависеть от вашей роли в организации.

Если ваша власть хоть в какой-то степени распространяется на такого “неподходящего” Scrum-мастера, соответствующую команду или на собственно процесс внедрения Scrum, побеседуйте с этим добровольцем и объясните ему, почему эту роль должен играть кто-то другой. Если это уместно, разъясните ему, какие качества он должен у себя выработать, чтобы впоследствии стать подходящим кандидатом на роль Scrum-мастера. А как быть, если неподходящий человек уже является Scrum-

мастером? Несмотря на то что это окажется несколько более трудным делом, я все же предлагаю лишить его этой роли (если, конечно, вы уверены, что он действительно не может быть Scrum-мастером). В любом случае действовать нужно быстро и решительно. “Неподходящего” Scrum-мастера следует заменить как можно быстрее: мне еще не встречалась команда, которая считала бы, что “неподходящего” Scrum-мастера убрали с его поста чересчур быстро.

Если же ваша власть не распространяется на такого “неподходящего” Scrum-мастера, соответствующую команду или на собственно процесс внедрения Scrum, я все равно советую вам поговорить с человеком, который столь опрометчиво взял на себя эту роль. Разговор должен основываться на том, что интересы команды — превыше всего. Попытайтесь сделать упор на достоинствах этого Scrum-мастера и предположите, что при реализации соответствующего проекта он наверняка сможет найти лучшее применение этим достоинствам, если откажется от роли Scrum-мастера.

Scrum-мастер одновременно является программистом, тестером и еще каким-то иным специалистом. Если наличие в команде “освобожденного” Scrum-мастера является непозволительной роскошью, вам придется выбирать между Scrum-мастером, который работает параллельно в нескольких командах, и Scrum-мастером, который является и Scrum-мастером, и программистом, тестером или каким-то еще специалистом в одной и той же команде. Несмотря на то что успешным может оказаться любой из этих подходов, лично я предпочитаю вариант, когда Scrum-мастер работает параллельно в двух командах. Использование в роли Scrum-мастера человека, который работает в той же команде программистом, тестером или другим специалистом, таит в себе немало рисков.

Один из этих рисков заключается в том, что такой Scrum-мастер не сможет уделять достаточно времени обеим своим ролям. Другой риск заключается в том, что человеку, исполняющему несколько ролей, не следует заниматься работами, находящимися на критическом пути, поскольку в любой момент от него могут потребовать переключиться на исполнение обязанностей Scrum-мастера. Менее бросающийся в глаза риск заключается в том, что другие члены команды могут и не знать, с кем они разговаривают в данный момент: со своим Scrum-мастером или с еще одним индивидуальным исполнителем проекта. Еще один риск — Scrum-мастер пользуется меньшим доверием, защищая команду от посторонних. “Освобожденный” Scrum-мастер пользуется большим доверием, когда говорит “Мы не можем вам помочь. Команда слишком загружена текущей работой”, чем Scrum-мастер, совмещающий свои обязанности с обязанностями индивидуального исполнителя проекта, в устах которого те же самые фразы могут быть интерпретированы как “Мы не можем вам помочь. Я слишком занят”.

Несмотря на то что совмещать роли Scrum-мастера и индивидуального исполнителя проекта весьма рискованно для любого человека, такая ситуация довольно типична. Понимание указанных проблем и готовность решать эти проблемы по мере их возникновения зачастую является оптимальным вариантом выхода из подобной ситуации.

Scrum-мастер принимает решения вместо своей команды. Эта проблема может возникнуть по двум совершенно разным причинам: из-за того, что Scrum-мастер не-правильно понимает свою новую роль или чувствует себя в ней некомфортно, и из-за того, что команда привыкла, что вместо ее решения принимает кто-то другой.

В любом случае решение будет одним и тем же: Scrum-мастера нужно отвести в сторонку и напомнить ему, что его задача — давать людям советы и наставлять их на путь истинный, а не предлагать готовые ответы.

Оказавшись в роли начинающего Scrum-мастера, я должен был прежде всего научиться... считать. Да-да, именно считать. Когда у нас проходило совещание, в ходе которого мы пытались решить какую-либо непростую проблему, команда ждала от меня, что я подскажу им готовое решение. Поскольку в прошлом я был лидером команды, я испытывал соблазн выпалить “готовый ответ”. Но я хотел, чтобы команда сама находила правильные ответы, поэтому я старался сидеть спокойно и считать про себя: 1, 2, 3... В некоторых случаях счет шел на сотни, но это помогло мне научиться держать язык за зубами. В то же время это помогло команде не только научиться принимать правильные решения, но и понять, что я не буду принимать решения вместо них.

Владелец продукта

Я представляю себе Scrum-мастера человеком, обеспечивающим эффективное функционирование команды как единого коллектива, способствующим быстрому устранению препятствий на пути движения команды к поставленной цели и помогающим своей команде как можно быстрее достичь этой цели. Владельца продукта (product owner) я представляю себе как человека, который заботится о том, чтобы его команда двигалась к поставленной цели и не отклонялась от нее. Для того чтобы команда добилась успеха, на высоте должны быть исполнители обеих этих ролей. Владелец продукта указывает команде правильную цель, а Scrum-мастер помогает ей как можно быстрее и эффективнее ее достичь.

Роман Пихлер (Roman Pichler), автор книги *Agile Product Management: Turning Ideas into Winning Products with Scrum*, подчеркивает важность роли, которую играет владелец продукта: “Владелец продукта уполномочен поставить перед командой цель и сформировать для команды определенное представление о будущем продукте. Владелец продукта — это не просто руководитель проекта, который, помимо прочего, формулирует требования и правильно расставляет приоритеты”. Представление о владельце продукта как о постановщике цели для команды помогает прояснить определенные аспекты функциональных обязанностей владельца продукта. Например, владелец продукта отвечает за формулирование и приоритизацию журнала запросов на выполнение работ, который выражает цель команды. Аналогично владелец продукта отвечает за то, чтобы соответствующий проект обеспечивал достаточно высокую рентабельность капиталовложений в этот проект.

Обязанности владельца продукта

Составить исчерпывающий перечень обязанностей владельца продукта очень нелегко. Каждое приложение существует в рамках собственного контекста культуры компании, индивидуальных и коллективных компетенций, конкурентных сил и т.п. Этот контекст оказывает сильное влияние на то, как именно будет исполняться роль владельца продукта в тех или иных компаниях. Поэтому вместо того чтобы составлять перечень обязанностей владельца продукта (например, “должен участвовать в совещаниях, посвященных планированию спринтов”), я счел более оправданным мыслить в категориях двух понятий, которые владелец продукта формулирует для команды: представлений о будущем продукте и границ.

См. также Подробное обсуждение роли владельца продукта можно найти в книге Романа Пихлера *Agile Product Management: Turning Ideas into Winning Products with Scrum*.

Формирование представлений о будущем продукте

Обязанности владельца продукта предусматривают формирование представлений о будущем продукте и доведение этих представлений до сведения каждого члена команды. Лучшие из команд — это те, энтузиазм которых возник в результате появления у них четкого представления о будущем продукте, сформированного владельцем продукта. Кому мы будем продавать этот продукт? Что в нем уникального? Чем занимаются наши конкуренты? Как наш продукт будет эволюционировать с течением времени? Разумеется, конкретные вопросы зависят от того, какое именно приложение или услуга поставляется группе пользователей, но наличие общего для всех членов команды представления очень важно с точки зрения мотивации команды и создания долговременной связи между разработчиками продукта и его потребителями.

Помимо того что у владельца продукта должно сформироваться собственное представление о будущем продукте, он должен довести эти представления до сведения каждого члена команды. Владелец решает эту задачу путем создания, ведения и приоритизации журнала запросов на выполнение работ. Между Scrum-мастерами и командами существует немало разногласий в отношении того, является ли владелец продукта лицом, которое должно вести журнал запросов на выполнение работ. Лично я придерживаюсь того мнения, что это не имеет принципиального значения. Мне все равно, кто именно будет вести журнал запросов на выполнение работ. Важно лишь, что именно владелец продукта должен позаботиться о том, чтобы кто-то вел такой журнал. Если владелец продукта делегирует право ведения такого журнала какому-либо бизнес-аналитику и этот аналитик не справится с данной задачей, отвечать за это будет владелец продукта.

См. также Журнал запросов на выполнение работ представляет собой приоритизированный список функциональных возможностей (features), которые должны быть добавлены в продукт. Подробное описание журнала запросов на выполнение работ можно найти в главе 13, “Журнал запросов на выполнение работ”.

Помимо создания и ведения журнала запросов на выполнение работ, владелец продукта детализирует представления о будущем продукте, отвечая на вопросы, которые возникают у членов команды. Хотите ли вы, чтобы это осуществлялось именно таким образом? Что вы имели в виду, когда сказали то-то и то-то? Несмотря на то что владелец продукта может делегировать или распределить ответственность за ответы на подобные вопросы, он не может передать ответственность за сам факт предоставления ответов. Владелец продукта может сказать: “Если у вас есть вопросы по поводу того, как должны использоваться тележки для покупок и как должен осуществляться расчет с покупателем, поговорите с Томом”. Если же Том не умеетнятно отвечать на вопросы, эффективный владелец продукта постарается сам ответить на эти вопросы, выяснит, почему Том не в состоянии сделать это, поручит отвечать на вопросы кому-то другому или найдет другое решение.

Определение границ

Представления о будущем продукте и границах можно понимать как конкурирующие аспекты проекта. Представления о будущем продукте показывают, каким может стать продукт. Границы описывают реалии, в пределах которых должны быть реализованы эти представления о будущем продукте. Они определяются владельцем продукта и зачастую принимают форму таких ограничений.

- Мне это нужно к июню
- Нам нужно сократить наполовину себестоимость единицы нашей продукции
- Нужно в два раза повысить быстродействие этого продукта
- Он должен использовать не более половины памяти, которую занимает текущий продукт

Нередко, когда я рассказываю группам о том, что владелец продукта имеет право диктовать команде подобные вещи (особенно сроки), мои слушатели начинают бурно выражать недовольство. “Нет, — говорят они, — оценки — это дело команды. А владелец продукта должен заниматься лишь приоритизацией работ”. Несмотря на справедливость подобного утверждения владелец продукта отвечает и за формулирование границ, которые будут определять успех соответствующего продукта.

Самые опытные члены Scrum-команды наверняка согласятся, что владелец продукта имеет право сказать: “Нам нужно реализовать не менее такого-то объема журнала запросов на выполнение работ. В противном случае нечего и говорить об отправке продукта заказчику”. Но многие из тех же опытных членов Scrum-команды наверняка начнут протестовать, услышав то же самое о сроках выполнения работ. Но давайте послушаем, о чем пишут Такучи (Takeuchi) и Нонака (Nonaka) в своем исследовании шести команд, которое заложило фундамент Scrum и стало темой первой статьи о Scrum, опубликованной еще в 1986 году.

Высшее руководство Fuji-Xerox потребовало разработать принципиально новое копировальное устройство и предоставило команде, выполнившей проект FX-3500, два года на создание устройства, которое можно было бы производить с наполовину меньшей себестоимостью, чем у их лучшего на тот момент устройства, при условии сохранения столь же высоких технических характеристик (139).

В данном случае мы имеем дело с командой, которой предстоит решить непростую задачу: обеспечить технические характеристики самого совершенного на данный момент копировального устройства компании, но при половинной себестоимости изделия. При этом заданы весьма жесткие сроки решения этой задачи. В такой постановке задачи нет ничего удивительного. Владельцы продукта ошибаются, чрезмерно ограничивая проблему или делая решение задачи невозможным. Если бы руководство Fuji-Xerox потребовало от команды решить такую же задачу, предоставив для этого лишь один месяц, команда сразу же осознала бы всю тщетность попыток уложиться в столь сжатые сроки и даже не пыталась бы приступить к ее решению. Предлагая команде решить задачу в течение двух лет, руководство компании предоставило ей для этого достаточно времени. Одной из обязанностей владельца продукта, которая является скорее искусством, чем наукой, является формулирование таких границ для проекта, чтобы у команды появилась реальная мотивация для решения сложной задачи; в то же время эти границы не должны быть такими, что решение поставленной задачи окажется просто невозможным.

При поиске решений сложной задачи с помощью метода мозгового штурма типичный совет заключается в том, что мыслить нужно, “не замыкаясь в ящике” (“outside the box”). Однако существуют свидетельства, что наилучшие решения возникают с большей вероятностью, если мыслить “внутри ящика” (“inside the box”), при условии, что границы этого “ящика” установлены надлежащим образом (Coyne, Clifford, and Dye, 2007). Когда нам советуют мыслить, “не замыкаясь в ящике”, как говорят эти авторы, полное отсутствие ограничений должно вызывать опасения.

Представьте, что в ходе типичного сеанса мозгового штурма вы пытаетесь усовершенствовать некий продукт. Предложение “не замыкаться в ящике” в данном случае может означать возможность увеличения или уменьшения размеров этого продукта, увеличения или уменьшения его веса, приданье ему более изящной (или наоборот, более грубой) формы или изменение его внешнего вида каким угодно способом. Пойдя еще дальше, можно было бы удешевить этот продукт или сделать его более дорогим, сделать его конструкцию модульной или изготавливать его на основе каких-то других продуктов. Можно было бы изменить функциональные возможности этого продукта, срок его службы, повысить удобство пользования им или обеспечить его совместимость с другими продуктами. Можно было бы изменить его коэффициент готовности, доступность для потенциальных покупателей или ремонтопригодность. Как узнать, какие из этих характеристик было бы целесообразно рассматривать? Без подсказок и советов иногда трудно судить о том, следует ли продолжать двигаться в прежнем направлении или лучше выбрать другое направление. При отсутствии таких подсказок и советов люди не в состоянии справиться с неопределенностью; у них просто опускаются руки (2007, 71).

Задача владельца продукта заключается в том, чтобы создать новый “ящик” (рамки, границы), в пределах которого будет мыслить команда. Этот “ящик” не позволяет команде утонуть в бесконечном множестве возможных решений и дает им основу для сравнения разных вариантов и осуществления выбора. Границы для этого нового “ящика” определяются самыми важными ограничениями для соответствующей компании, к числу которых могут относиться такие ориентиры, как минимальная гарантированная функциональность, резкое повышение производительности, сокращение потребления ресурсов и, конечно же, во многих случаях — сроки.

См. также Календарный план является одной из сторон пресловутого “железного треугольника” (масштаб, календарный план и ресурсы). Подробнее о “железном треугольнике” читайте в главе 15, “Планирование”.

Каждой команде нужен только один владелец продукта

В команде, которая только приступает к освоению Scrum, работа Scrum-мастера связана с большими затратами времени. Scrum-мастеру приходится заниматься обучением членов команды этой новой для них методологии, побуждать их обдумывать с разных точек зрения проблемы, с которыми им приходится сталкиваться, устранять препятствия, возникающие на пути команды, и т.д. и т.п. В самом начале освоения Scrum — а также если на пути команды возникает слишком много препятствий — команде может даже понадобиться “освобожденный” Scrum-мастер. Однако со

временем ситуация, несомненно, улучшится. Scrum-мастеру удастся преодолеть большую часть препятствий, мешающих его команде успешно выполнять Scrum-проект, а сама команда в достаточной степени овладеет методологией Scrum и уяснит свою самоорганизующуюся природу. Эти перемены приведут к тому, что команда будет все меньше и меньше нуждаться в услугах Scrum-мастера, высвобождая его время для другой работы. Если представить в графическом виде потребность команды в услугах Scrum-мастера, выразив эту потребность в количестве времени, затрачиваемого Scrum-мастером на исполнение его обязанностей, то мы получим диаграмму, подобную показанной на рис. 7.1.

См. также Подробнее о самоорганизации можно прочитать в главе 12, "Управление самоорганизующимся коллективом".

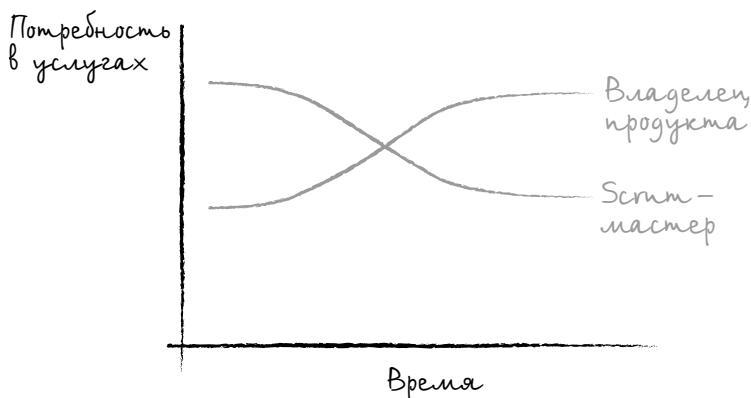


Рис. 7.1. Со временем потребность команды в услугах владельца продукта и Scrum-мастера изменяется по-разному

Сравните эту диаграмму с потребностью команды в услугах владельца продукта. Когда команда начинает переход к использованию Scrum, неудивительно, что многое в этой методологии кажется ей непонятным. Неясно, в частности, насколько подробной должна быть информация, представленная в журнале запросов на выполнение работ, какой объем работ может быть выполнен в рамках одного спринта, как добиться совместной, слаженной работы в рамках спринта и т.п. В первое время команда не сможет работать достаточно быстро (во всяком случае так быстро, как она будет работать после освоения Scrum). По мере повышения производительности команда (в результате совершенствования своей деятельности и по мере устранения препятствий Scrum-мастером) станет выполнять все большие объемы работ в течение каждого спринта. Это означает, что у членов команды будет появляться все больше вопросов к владельцу продукта. Таким образом, по мере повышения производительности команды будет повышаться ее спрос на услуги владельца продукта. Это, вероятно, будет происходить, даже когда члены команды хорошо овладеют своей предметной областью и возьмут на себя большую ответственность.

Эти противоположные одна другой взаимосвязи (между потребностью команды в услугах владельца продукта и ее потребностью в услугах Scrum-мастера) представлены

на рис. 7.1. Анализируя этот рисунок, можно прийти к следующему выводу: несмотря на то, что опытный Scrum-мастер вполне может работать параллельно с двумя или даже тремя командами (в зависимости от того, какой объем помощи требуется каждой из них на соответствующем отрезке времени), совершенно недопустимо, чтобы владелец продукта работал одновременно более чем с двумя командами. Напротив, желательно, чтобы у каждой команды был свой “освобожденный” владелец продукта. Владельцу продукта приходится решать весьма непростые задачи. Частично эти задачи являются “внешними” по отношению к команде: общение с клиентами и отслеживание тенденций на рынке. Частично эти задачи касаются непосредственной работы с командой: совместная работа с командой над созданием соответствующего продукта. Когда работа владельца продукта связана с выполнением как внешних, так и внутренних (по отношению к команде) обязанностей, задача взаимодействия с клиентами, как правило, выходит на первый план. Любой разработчик, отвечающий и за новые разработки, и за поддержку клиентов, подтвердит, что на первый план почти всегда выходит взаимодействие с клиентами.

См. также Тема расширения роли владельца продукта в случае крупных проектов подробно освещается в главе 17, “Изменение масштаба Scrum”.

Точно так, как владелец продукта должен работать только с одной командой, каждая команда должна работать только с одним владельцем продукта. Я знаю случаи, когда использование одной командой двух владельцев продукта приносило успех, но, как правило, это являлось следствием того, что в организации не находилось человека, который взял бы на себя смелость сказать: “Ваш владелец продукта такой-то, и точка”. Очень важно, чтобы такой человек нашелся в каждой организации, назначил команде одного владельца продукта и убедил этого человека оказывать команде необходимую ей помощь (в том числе обеспечивать содействие со стороны тех, кто также мог бы стать владельцем ее продукта).

Команда с двумя владельцами продукта неизбежно попадет в западню типа “мама не разрешает; может быть, разрешит папа?” Разумеется, лишь самые неблагополучные (или, возможно, доведенные до отчаяния) команды, получив “неправильный” ответ от одного владельца продукта, обращаются с тем же вопросом к другому, даже если понимают, что этот неблаговидный поступок станет известен обоим владельцам и будет подвергнут публичному осуждению. Однако большинство команд с двумя владельцами продукта заранее “просчитывают”, кто из двух владельцев продукта даст более приемлемый для них ответ, и обращаются именно к нему.

Команда владельцев продукта

В некоторых случаях роль владельца продукта может оказаться непосильной для одного человека. Исследователи Анджела Мартин (Angela Martin), Роберт Биддль (Robert Biddle) и Джеймс Ноубл (James Noble) пришли к выводу, что человек, оказавшийся в роли владельца продукта, “постоянно испытывает на себе большее давление, чем разработчики и другие исполнители проекта” (2004, 51). С таким мнением согласен Рон Джейфрис (Ron Jeffries), один из изобретателей экстремального программирования и преподаватель методологии Scrum: “Лишь после выхода в свет одной-двух первых книг по XP мы уяснили по-настоящему, какие нагрузки приходится

испытывать отдельно взятому XP-клиенту/владельцу Scrum-продукта. Совершенно ясно, что эти функции должен выполнять не один человек, а группа людей”.

Типичным решением этой проблемы является использование *команды владельцев продукта*. Распределение обязанностей владельца продукта между членами команды владельцев продукта является разумным решением, если в этой команде есть человек, обладающий всей полнотой власти и принимающий на себя всю ответственность, своего рода “козел отпущения”. Даже при наличии команды владельцев продукта у каждой команды разработчиков должен быть “персональный” владелец продукта, к которому они могли бы обращаться с вопросами. Кен Швабер (Ken Schwaber) и Майк Бидли (Mike Beedle) пишут по этому поводу: “Владелец продукта — это конкретный человек, а не комитет” (2001, 34). Позаботьтесь о том, чтобы каждой команде был назначен человек, к которому члены команды могли бы обращаться со своими вопросами. Хорошая Scrum-команда действует слишком быстро, чтобы ожидать, пока какой-нибудь комитет ответит на все ее вопросы. Владелец продукта никогда не сможет незамедлительно отвечать на все вопросы, которые могут возникнуть у команды; нет ничего плохого в том, что время от времени он будет говорить им: “Я должен посоветоваться со своими коллегами”. Но хорошо обоснованное предостережение не должно заменяться фактическим решением комитета.

Качества хорошего владельца продукта

Как и при описании критериев, которыми следует пользоваться при выборе или приеме на работу хорошего Scrum-мастера, пространный перечень качеств, которыми должен обладать хороший владелец продукта, я сократил до пяти абсолютно необходимых качеств.

Доступность. Едва ли не самое частое нарекание команд в адрес своих владельцев продукта заключается в том, что их трудно найти на месте. Когда команде, работающей в быстром темпе, нужно получить ответ на тот или иной вопрос, ожидание ответа в течение двух-трех дней может полностью сбить команду с уже установившегося ритма работы. Постоянно находясь “под рукой” у команды, владелец продукта демонстрирует готовность отдавать все свои силы данному проекту. Наилучшие из владельцев продуктов демонстрируют эту свою готовность, делая все необходимое для создания как можно более качественного продукта. В некоторых проектах это включает такие действия, как оказание помощи в планировании тестов, выполнение тестов вручную и активное участие в работе других членов команды.

Знание бизнеса. Важно, чтобы владелец продукта хорошо разбирался в бизнесе, которым занимается его компания. Выступая в роли лица, которое принимает решения, касающиеся определенного продукта, владелец этого продукта должен хорошо разбираться в соответствующем бизнесе и рыночных условиях, а также знать потребности клиентов и пользователей этого продукта. Как правило, столь глубокое понимание приходит после многих лет работы в соответствующем бизнесе; нередко оно вырабатывается у бывшего пользователя того типа продукта, который в настоящее время разрабатывается командой. Вот почему многие успешные владельцы продукта являются бывшими продукт-менеджерами, специалистами по маркетингу или бизнес-аналитиками.

Коммуникабельность. Владельцы продукта должны быть коммуникабельными людьми, т.е. должны уметь тесно сотрудничать с широким кругом лиц, заинте-

сованных в успешной реализации соответствующего проекта. Владельцу продукта приходится постоянно общаться с пользователями, клиентами, руководством своей организации, партнерами и, естественно, остальными членами команды. Квалифицированные владельцы должны уметь подавать одну и ту же информацию каждой из перечисленных категорий лиц таким образом, чтобы она в максимальной степени отвечала потребностям той или иной категории.

Хороший владелец продукта должен уметь выслушать пользователей, клиентов и, наверное, самое главное — членов команды. По мере того как члены команды будут узнавать все больше и больше о продукте и рынке (а со временем их познания обязательно будут расширяться, особенно при выполнении Scrum-проекта), они смогут выдвигать собственные ценные предложения по поводу этого продукта. Кроме того, у всех команд появится больше возможностей высказать владельцу продукта собственные соображения о технических рисках и проблемах соответствующего проекта. Несмотря на то что владелец продукта действительно приоритизирует всю работу для своей команды, дальновидный владелец продукта обязательно выслушает членов команды, если те пожелают высказать собственное мнение по поводу правильности расстановки приоритетов, основываясь на технических факторах.

Решительность. Еще одно распространенное нарекание команд на владельцев их продукта заключается в том, что очень часто владельцы проявляют нерешительность. Члены команды, обращаясь к владельцу продукта с тем или иным вопросом, хотят получить от него четкий ответ. В условиях Scrum-проектирования командам приходится действовать в очень жестких временных рамках. Члены команды испытывают растерянность, когда на очередной их вопрос владелец продукта отвечает примерно так: “Чтобы ответить на ваш вопрос, мне нужно посоветоваться с коллегами или посовещаться с руководством компании”. Разумеется, иногда без этого никак не обойтись, но команда, как правило, тонко чувствует, когда в таких совещаниях действительно есть необходимость, а когда владелец продукта просто пытается избежать принятия “трудного” решения. Владелец продукта, который раз за разом изменяет свои решения, ничуть не лучше владельца продукта, который избегает принятия решения. Хороший владелец продукта отменяет свое решение лишь в исключительных случаях.

Наличие вполне определенных властных полномочий. Хороший владелец продукта должен обладать вполне определенными властными полномочиями, позволяющими ему принимать решения. Разумеется, он должен нести всю ответственность за принятые им решения. Из сказанного следует, что владелец продукта должен обладать в организации достаточно высоким статусом, чтобы его можно было наделить такими полномочиями. Если решения, принимаемые владельцем продукта, постоянно оспариваются вышестоящими руководителями организации, то члены команды будут обращаться со своими вопросами и проблемами не к владельцу продукта, а к этим вышестоящим руководителям.

Scrum-мастер как владелец продукта

Возникает естественный вопрос: “Может ли один и тот же человек исполнять одновременно роли Scrum-мастера и владельца продукта?” На него я отвечаю без колебаний: “Нет, не может”. В подавляющем большинстве известных мне случаев совмещения ролей Scrum-мастера и владельца продукта в одном лице результат оказывался неизменно плачевным. Дело не только в том, что подобное совмещение приводит к

сосредоточению в одних руках слишком большой власти, но и в том, что такое совмещение приводит к большой путанице. Сама природа этих ролей такова, что между ними постоянно возникают какие-то трения и противоречия. Владельцу продукта постоянно требуется все больше и больше функциональных возможностей. Scrum-мастер защищает свою команду от чрезмерных притязаний со стороны владельца продукта, когда чувствует, что эти притязания могут лишь повредить делу. Когда роли Scrum-мастера и владельца продукта совмещаются в одном человеке, эти трения исчезают сами по себе.

Понимая, что меня могут разоблачить, я чувствую себя обязанным добавить, что в двух самых успешных Scrum-проектах, в которых мне приходилось либо участвовать непосредственно, либо быть сторонним наблюдателем, роли Scrum-мастера и владельца продукта совмещались в одном лице. Просто замечательно, когда один и тот же человек совмещает глубокое понимание рынка с такими качествами, необходимыми хорошему Scrum-мастеру, как превосходные технические знания и способность к сотрудничеству, умея в то же время эффективно уравновешивать в себе все эти качества. Так, Toyota объединяет роли Scrum-мастера и владельца продукта в одном лице — в лице главного инженера. Главный инженер Toyota — это специалист, который, несомненно, является инженером и может разработать любой узел нового автомобиля. Вместе с тем он отличается глубоким пониманием рынка и знанием вероятных покупателей проектируемого автомобиля.

Словом, модель совмещения в одном лице ролей Scrum-мастера и владельца продукта *может* быть успешной. Однако я подозреваю, что лишь очень немногие люди способны успешно играть обе эти роли. Даже если вам кажется, что вы принадлежите к числу этих немногих или знаете о существовании таких людей в своей организации в момент начала перехода к Scrum, я все равно рекомендовал бы вам использовать для этих ролей разных людей (особенно на начальном этапе перехода к Scrum).

Преодоление типичных проблем

При выборе кандидатуры на роль первоначального владельца продукта можно попасть в несколько потенциальных ловушек. Ниже перечислены типичные проблемы, которые возникают на ранних стадиях, и возможные способы их решения.

Владелец продукта делегирует право принятия решений другому, а затем отменяет принятые тем решения. Чтобы совместить исполнение новых обязанностей владельца продукта со своим планом работ, некоторые владельцы делегируют другим право принятия решений, касающихся определенных частей данного продукта. Другие владельцы продукта поручают кому-то из бизнес-аналитиков стать “владельцем функциональной возможности” в отношении некоторой части системы. Это может оказаться удачным вариантом, поскольку у владельца продукта появляется больше времени, которое он может посвятить проблемам, право решения которых делегировано гораздо сложнее.

См. также Формирование иерархии владельцев продукта, подобной описанной, представляет собой типичный метод изменения масштаба и подробно освещается в главе 17, “Изменение масштаба Scrum”.

Проблемы возникают, когда владелец продукта делегирует другим право принятия решений, но при этом продолжает утверждать эти решения и время от времени даже их отменяет. Прежде чем делегировать кому-то право принятия решений, владелец продукта должен осознать, что он безоговорочно делегирует это право и не намерен впоследствии пересматривать принятые решения. Работая в условиях жестко ограниченных по времени спринтов, Scrum-команды зачастую выполняют работу гораздо быстрее, чем до перехода к Scrum. Практически неизбежно то, что некоторые решения, делегированные владельцем продукта, окажутся ошибочными и их придется пересматривать. Однако в действительности необходимо избегать ситуаций, когда владелец продукта говорит: “На все ваши вопросы ответит Дэйв — за эту часть системы отвечает он”, а затем опровергает ответы Дэйва.

Мой совет начинающему и перегруженному работой владельцу продукта таков: освободите для себя время, делегировав кому-то другому чуть больше, чем представляется комфортным для вас. Вы будете приятно удивлены, если окажется, что вам не придется отменять ни одного мало-мальски важного решения. Но время от времени некоторые решения вам все же придется пересматривать. Нередко самое лучшее, что можно сделать в таком случае, — поступить так, как учат поступать начинающих водителей: если ваш автомобиль заносит, рулите в сторону заноса. Иными словами, вместо того чтобы принимать решение, полностью противоположное неправильному (конечно, при условии, что это неправильное решение не является катастрофическим), предоставьте возможность этому решению оставаться в силе до конца спринта и лишь после этого решайте, следует ли его отменять. Если стоимость отмены решения сравнить со всей остальной важной работой, представленной в журнале запросов на выполнение работ, то может оказаться, что с принятым решением вполне можно мириться.

Владелец продукта слишком рьяно понукает команду. Владельцам продукта нередко приходится любой ценой обеспечивать для своей компании необходимые финансовые результаты. Добиваться наискорейшей реализации как можно большего числа функциональных возможностей — один из способов достижения таких результатов. Как я уже говорил, я не имею ничего против владельца продукта, который в самом начале выполнения проекта объявляет команде: “Нам нужно создать более дешевый продукт с меньшими размерами и лучшими техническими характеристиками, чем у нашего конкурента. Для реализации этого проекта я отвожу вам на три месяца меньше времени, чем было потрачено на создание предыдущей версии этого продукта”. Если серьезная цель, подобная этой, подкрепляется предоставлением свободы в выборе конкретных способов ее достижения, команда сделает все от нее зависящее. Проблемы возникают, когда от спринта к спринту команда испытывает постоянно возрастающее давление со стороны владельца продукта. Труднодостижимая цель “сделать эту замечательную вещь за шесть месяцев” во многих отношениях напрягает команду гораздо меньше, чем 13 следующих один за другим двухнедельных спринтов, каждый из которых начинается со слов “Мне нужно больше, больше, больше!” Если ваши владельцы продукта именно так обращаются со своими командами, то Scrum-мастерам нужно осадить этих не в меру зарвавшихся людей, предложив им ставить перед своими командами более долгосрочные цели и в то же время предоставив им определенную степень свободы в выборе конкретных способов достижения этих целей.

Владелец продукта предлагает снизить качество. Снижение качества — весьма со-блазнительное решение, особенно когда требуется реализовать внушительное число

возможностей в сжатые сроки. Это может создать краткосрочное впечатление достижения целей, поставленных в начале выполнения проекта. Однако результаты (точнее говоря, издержки) такого снижения качества будут становиться все очевиднее по мере того, как пользователи начнут выявлять в продукте все больше и больше “проколов”, начнет снижаться производительность команды, а клиенты станут заявлять, что продукт функционирует вовсе не так, как они ожидали.

Кен Швабер называет качество “корпоративным активом” (2006), и никто (возможно, за исключением главы фирмы) не имеет права жертвовать качеством во имя достижения какой-либо краткосрочной цели (например, соблюдения сроков выпуска продукта). Решение о снижении качества может оказаться правильным; я не могу утверждать обратное, не зная всего контекста конкретной ситуации. Однако это решение должно приниматься на как можно более высоком уровне руководства организацией и должно быть настолько гласным, чтобы ни для кого не были неожиданностью негативные последствия, которые почти наверняка возникнут в результате выполнения такого решения.

Выбор владельцев продукта, которые это понимают, подчас является весьма непростой задачей в организациях, которых постоянно волнует лишь ближайшее будущее (в пределах текущего квартала). Борьба с попытками снижения качества является одной из важнейших задач Scrum-мастера. Scrum-мастру нет нужды демонстрировать свое верховенство в этих ранних конфликтах. Однако он должен гарантировать, что решение о снижении качества будет гласным.

Время работает на Scrum-мастера. Если Scrum-мастер обеспечивает гласность решений о снижении качества, то впоследствии он сможет отстоять свою точку зрения, представив убедительные доводы против снижения качества. “Помните, как во время выполнения проекта Gouda я говорил вам, что снижение качества первой версии продукта отрицательно скажется при разработке второй версии продукта?” — может сказать Scrum-мастер. “Итак, существуют диаграммы скорости по этим двум проектам. Обратите внимание, что при разработке второй версии продукта скорость снизилась, несмотря на то что мы включили в состав команды двух опытных специалистов. Это объясняется тем, что в первой версии продукта осталось много не выявленных нами ошибок (а вот и диаграмма, демонстрирующая это), а также тем, что у команды сложилось впечатление, что ей не хватает времени для всесторонней проверки исходного кода. Мы даже отказались от автоматизированного тестирования исходного кода нескольких модулей. Вот сравнение числа дефектов, обнаруженных в течение шести месяцев после выпуска продукта, с указанием, выполнялось ли автоматизированное тестирование исходного кода по каждомуциальному модулю. Разумеется, в следующий раз окончательное решение снова останется за вами, но вы по крайней мере знаете мое мнение по этому вопросу”.

Владелец продукта проживает в другом городе. Учитывая, что все большее число проектов разрабатывается удаленными командами, подобная ситуация становится все более типичной. Как команда, так и владелец продукта должны принять на себя часть бремени дополнительного общения друг с другом. Мне пришлось в свое время работать со многими удаленными владельцами продукта, и ничего страшного в этом нет, при условии, что владелец продукта

- активно участвует в реализации проекта;
- поддерживает хорошие взаимоотношения с командой;

- выполняет обычные обязанности, предусмотренные его ролью;
- оперативно отвечает на телефонные звонки членов команды (для этого иногда достаточно выделить определенное “окно”, причем вовсе необязательно, чтобы оно укладывалось в рамки рабочего дня владельца продукта);
- своевременно отвечает по электронной почте или телефону, если звонок от кого-либо из членов команды не застал его на месте.

См. также Подробнее о проблемах распределенной разработки можно прочитать в главе 18, “Рассредоточенные коллективы”.

Новые роли, прежние обязанности

Роли владельца продукта и Scrum-мастера очень важны для команды, которая желает стать высокоэффективной Scrum-командой. В этой главе мы рассмотрели обязанности людей, исполняющих эти роли, качества, которыми должны обладать хорошие владельцы продукта и хорошие Scrum-мастера, а также способы решения ряда типичных проблем, которые возникают при появлении в организации этих новых ролей.

Несмотря на новизну ролей владельца продукта и Scrum-мастера их обязанности вовсе не новы. Высокоэффективные команды всегда знали, что должны выполнять функции, описанные в этой главе. Члены Scrum-команды отнюдь не ограничиваются исполнением функций, оговоренных в их должностной инструкции; им постоянно приходится теми или иными способами помогать своей команде в достижении поставленных целей. В следующей главе мы рассмотрим, как этот акцент на коллективном труде и совместной ответственности сказывается на ролях, уже существующих в организации.

Дополнительная литература

Davies, Rachel, and Liz Sedley. 2009. *Agile coaching*. The Pragmatic Bookshelf.

В этой книге содержится множество полезных практических советов для Scrum-мастеров любой квалификации. Она охватывает широкий круг тем, начиная с того, как помочь команде повысить свою эффективность, и заканчивая тем, как повысить собственную эффективность.

Fisher, Kimball. 1999. *Leading self-directed work teams*. McGraw-Hill.

В этой книге Фишера речь идет о самоорганизующихся командах, разрабатывающих проекты с помощью гибкой методологии разработки. Эта книга может служить ценным руководством для Scrum-мастеров.

James, Michael. 2007. A ScrumMaster's checklist, August 13. Блог Майкла Джеймса на веб-сайте Danube по адресу http://danube.com/blog/michaeljames/a_scrummasters_checklist.

Доказывая, что хорошей команде требуется Scrum-мастер, который уделял бы ей все свое рабочее время, а не человек, который работает параллельно с несколькими

командами, Майкл Джеймс представляет весьма обширный список работ, которые надлежит выполнять Scrum-мастера.

Kelly, James, and Scott Nadler. 2007. Leading from below. *MIT Sloan Management Review*, March 3. <http://sloanreview.mit.edu/business-insight/articles/2007/1/4917/leading-from-below>.

В этой статье представлена полезная информация для тех, кто хотя и не наделен властными полномочиями, но понимает, что способен влиять на курс своей организации.

Pichler, Roman. Forthcoming. *Agile product management with Scrum: Creating products that customers love*. Addison-Wesley Professional.

Самая полная на сегодняшний день информация о роли владельца продукта. Пихлер разъясняет важнейшие различия между традиционным и гибким управлением продуктом, предлагая в то же время полезные советы владельцам продукта.

Schwaber, Ken. 2004. *Agile project management with scrum*. Microsoft Press.

Во второй книге Швабера читатели найдут немало поучительных историй о командах, которые приобрели как положительный, так и отрицательный опыт использования Scrum. Автор не только описывает роли Scrum-мастера и владельца продукта, но и приводит множество полезных советов, касающихся исполнения этих ролей.

Spann, David. 2006. *Agile manager behaviors: What to Look for and develop*. Cutter Consortium Executive Report, September.

В этом обширном отчете консультант компании Cutter Consortium Дэвид Спенн отвечает на вопрос “Какие качества требуются специалисту, которого Дэвид Спенн называет «менеджером гибкой методологии разработки»?” Эти качества во многом совпадают с качествами Scrum-мастера, описанными в данной главе. Автор начинает с перечисления 22 качеств, которыми должен обладать менеджер гибкой методологии разработки, но затем сокращает этот список до восьми важнейших качеств, которыми должен обладать кандидат на такую должность.

Глава 8

Изменившиеся роли

В предыдущей главе рассматривались две новые роли в Scrum-проекте: Scrum-мастер и владелец продукта. Но перемены в структуре команды, которой предстоит выполнять Scrum-проект, не ограничиваются появлением двух указанных ролей. Например, то, что по своей природе Scrum-команда является самоорганизующейся, исключает роль технического лидера команды; члены команды не должны ограничиваться исполнением лишь своих непосредственных должностных обязанностей (напротив, они должны помогать своей команде всеми доступными им способами); акцент переносится с письменного изложения требований на их обсуждение; от команд требуется создавать нечто работоспособное к концу каждого спринта. Поскольку эти перемены ведут к изменению ролей и отношений как внутри команды, так и в организации в целом, они зачастую усугубляют проблемы, с которыми сталкивается организация, внедряющая у себя Scrum.

В этой главе описаны основные корректизы, которые должны вносить в свою деятельность люди при переходе от исполнения своих традиционных ролей к работе по методологии Scrum. Мы сосредоточим свое внимание не на подробном описании каждой роли, а на их изменении. Например, я не намерен подробно описывать все, что делает тестер в ходе тестирования того или иного приложения. Вместо этого я уделю внимание переменам в работе тестера при выполнении Scrum-проекта. В этой главе мы обсудим роли аналитика, менеджера проекта, архитектора, функционального менеджера, программиста, администратора базы данных, тестера и проектировщика пользовательского интерфейса.

ПРИМЕЧАНИЕ

Читая материал, посвященный этим ролям, нужно помнить, что любой член команды, участвующий в разработке того или иного продукта либо системы программного обеспечения, является, в первую очередь, и главным разработчиком. Используя термин *тестер*, я имею в виду разработчика, специализирующегося на тестировании и обладающего соответствующей квалификацией. Аналогично термин *аналитик* употребляется для обозначения разработчика, который предпочитает заниматься аналитическими задачами, но при этом может решать любую другую высокоприоритетную задачу, важную для его команды.

Аналитики

Досконально зная соответствующий продукт и обладая таким ценным качеством, как коммуникабельность, некоторые аналитики будут тяготеть к исполнению роли владельца продукта. Это особенно характерно для крупных проектов, в которых используется целая иерархия владельцев продукта. Кто-то из бывших продукт-менеджеров, например, может исполнять роль главного владельца продукта по отношению к продукту в целом, затрачивая большую часть своего времени на изучение поведения пользователей и рынков. В то же время кто-то из бывших аналитиков может исполнять роль владельца продукта для тех или иных команд и, взаимодействуя с главным владельцем продукта, отражать собственные взгляды и представления в журналах запросов на выполнение работ своих команд.

См. также Об изменении масштаба роли владельца продукта подробно рассказывается в главе 17, “Изменение масштаба Scrum”.

Многие команды приходят к выводу о том, что аналитик в команде по-прежнему очень нужен. Правда, аналитик в Scrum-команде работает по-новому. В проектах, выполняемых традиционным способом, задача аналитика заключается в том, чтобы как можно сильнее опережать свою команду. В Scrum-проектах целью становится анализ, выполняемый к нужному моменту (*just-in-time*). Новая цель аналитика заключается в том, чтобы опережать свою команду лишь на самую малость, в то же время снабжая ее полезной информацией о текущих и ближнесрочных функциональных возможностях программного обеспечения.

Аналитики могут оказать своим командам существенную помощь в деле смещения акцента с письменного изложения требований на их обсуждение. Поскольку в Scrum-проектах аналитики работают со значительно меньшим опережением своих команд, чем в проектах, выполняемых традиционным способом, они могут совершенно спокойно делиться своей информацией с командой в более неформальной обстановке (т.е. устно), не прибегая к составлению многостраничных документов. Команда должна получать от аналитика в устной форме, в процессе верbalного общения, как можно больший объем информации. Впрочем, какие-то требования аналитики по-прежнему должны документировать, особенно если аналитику приходится работать с рассредоточенной командой. Однако зачастую то, что пишет аналитик, носит менее формальный характер, например представлено в форме вики (общедоступная доска объявлений на базе веб. — *Примеч. пер.*), а не документа со специальной страницей для подписи.

См. также О смещении акцента с документов на устное обсуждение подробно рассказывается в главе 13, “Журнал запросов на выполнение работ”.

В проектах, выполняемых традиционным способом, аналитики нередко становятся посредниками, облегчающими общение между владельцем продукта и остальными членами команды. В случае Scrum-проектов аналитик должен быть не столько посредником, сколько помощником, облегчающим диалог между командой и

владельцем продукта. Члены команды и владельцы продукта должны общаться между собой. Вместо того чтобы исполнять роль простого “звукопровода” или телефона, хороший аналитик Scrum-проекта стремится обеспечить максимальную продуктивность диалога между командой и владельцем продукта, учитывая жесткие временные ограничения, в которых обычно приходится действовать команде и владельцу продукта. Это означает, что аналитик должен направлять диалог между командой и владельцем таким образом, чтобы они обсуждали историю конкретного пользователя (и не переходили к другому), поскольку подобные обсуждения отличаются тем, что их участники, увлекшись, незаметно для себя перескакивают с одной темы на другую. Это может также означать, что аналитик должен довести до сведения команды представления руководства организации о той или иной новой функциональной возможности, прежде чем команда и владелец продукта соберутся вместе для обсуждения касающихся ее деталей.

См. также Истории пользователей — это средство описания возможностей, характерное для гибкой методологии разработки. Подробнее об историях пользователей рассказывается в главе 13, “Журнал запросов на выполнение работ”.

В случае проектов, выполняемых традиционным способом, аналитик может сказать команде: “Я поговорил с одним важным лицом, заинтересованным в нашем проекте, уяснил, чего он хочет, и составил подробный документ с описанием его пожеланий”. В случае Scrum-проекта тот же аналитик должен сказать: “Я поговорил с нашим владельцем продукта, и, мне кажется, я понял, в чем заключаются его пожелания. Я составил вот эти шесть историй пользователей, чтобы вы могли приступить к работе. У меня есть еще ряд вопросов к владельцу продукта. Но мне хотелось бы, чтобы в моей беседе с владельцем продукта участвовали хотя бы двое членов нашей команды”.

Учитывая устоявшееся мнение об аналитиках как о тех, кто смотрит в будущее, можно подумать, что аналитики действуют на один спринт впереди своей команды. Это не так. Грегори Топп (Gregory Topp), аналитик из Farm Credit Services of America, описывает, как Scrum помогает ему сконцентрироваться на текущем спринте: “До того как мы начали использовать Scrum, мне приходилось сосредоточиваться на требованиях, реализовывать которые предстояло через несколько недель, а то и месяцев. Сейчас я сосредоточиваюсь на текущем спринте (у нас спринт длится две недели) и поэтому могу тратить больше времени на подробности историй пользователей, разработку и тестирование”. Главным приоритетом аналитика является достижение целей текущего спринта. Аналитик в типичной Scrum-команде оказывает помощь в тестировании, отвечает на вопросы (или отыскивает ответы на вопросы), касающиеся разрабатываемых функциональных возможностей, принимает участие во всех регулярных совещаниях по планированию спринта и т.п.

Однако вполне возможно, что указанные виды деятельности не отнимают все рабочее время аналитика. Время, не связанное с выполнением работы по текущему спринту, можно посвятить размышлению над будущим. Однако участвовать в работе команды по текущему спринту и тратить какое-то время на то, чтобы заглянуть в будущее, — это не то же самое, что действовать на один спринт впереди своей команды. Грегори Топп объясняет, как забегание слишком далеко вперед фактически может

отбросить вас далеко назад: “Я пытался работать с упреждением на один-два спринта, определяя подробности историй пользователей, и пришел к выводу, что такой подход причиняет вред текущему спринту. Кроме того, я пришел к выводу, что очень часто подробности историй пользователей изменяются к тому моменту, когда команда фактически приступает к работе над соответствующей историей”.

Типичный вопрос можно сформулировать так: “Следует ли включать в журнал спринта время, затраченное аналитиком на то, чтобы заглянуть в будущее?” Моя рекомендация такова: в журнал спринта следует включать любые конкретные задачи аналитика, которые могут быть идентифицированы в ходе планирования спринта. Допустим, что команда работает над приложением, которое принимает или отвергает заявки на получение кредита. Если владелец продукта и команда согласны, что следующий спринг будет включать работу по вычислению кредитного балла заявителя, то связанные с этим задачи предварительного анализа нужно идентифицировать, оценить и включить в журнал спринта. С другой стороны, если работа следующего спринга неизвестна, то никакие конкретные задачи, относящиеся к следующему спрингу, не следует включать в журнал спринта.

В целом многие аналитики одобряют переход к Scrum, несмотря на то что им приходится расстаться с ролью единственного интерпретатора пожеланий клиентов. Спустя два года после перехода к Scrum Грегори Топп высказался о том, как изменились его отношения с членами команды.

Поскольку все мы работаем в одной команде, причем работаем в одно и то же время над одними и теми же историями пользователей, складывается впечатление большей сплоченности команды. До использования Scrum аналитики, программисты, тестеры, администраторы баз данных кивали друг на друга и было непонятно, кто же отвечает за конечный результат, т.е. за работоспособный продукт. При использовании Scrum команда полностью сосредоточивается на небольшой совокупности историй. Люди начинают мыслить, как единая команда, а следовательно, перестают кивать друг на друга.

Руководители проектов

При реализации проекта с помощью последовательного процесса разработки руководитель проекта решает очень непростую задачу — обеспечивает разработку именно того продукта, который нужен клиенту. Чтобы решить эту задачу, руководителю проекта приходится управлять буквально всем, что связано с соответствующим проектом, в том числе масштабом проекта, величиной затрат, персоналом, коммуникациями, рисками, снабжением и т.д. Некоторые из перечисленных обязанностей в действительности должны исполнять другие. Например, управление масштабом проекта по праву принадлежит клиенту. Никто другой не вправе принимать необходимые компромиссные решения, которые придется принимать в ходе разработки продукта по мере изменения приоритетов, скорости работы команды и условий на рынке. Рассстановка приоритетов отнюдь не является статичным, одноразовым действием, которое выполняется руководителем проекта в самом начале его реализации. Проекты последовательного типа то и дело заставляют руководителей проекта строить более или менее обоснованные догадки относительно того, как обеспечить создание именно такого продукта, какой требуется клиенту.

Что же касается Scrum-проектов, то мы признаем неуместность роли руководителя проекта и отменяем ее. Однако отмена роли не означает, что мы можем отмахнуться от работы, которую выполнял руководитель проекта, и его повседневных обязанностей. Как нетрудно догадаться, поскольку самоорганизуемость команды является одним из основополагающих принципов Scrum, значительная часть обязанностей, возлагавшихся ранее на руководителя проекта, в новых условиях перекладывается на Scrum-команду. Так, при отсутствии руководителя проекта, который распределяет задачи между людьми, членам команды приходится самим распределять их между собой. Прочие обязанности руководителя проекта переходят к Scrum-мастеру или владельцу продукта.

Бывшие руководители проекта нередко принимают на себя одну из ролей, которые предполагают исполнение какой-то части их прежних обязанностей: руководитель проекта становится либо Scrum-мастером, либо владельцем продукта, либо одним из членов команды — в зависимости от опыта, квалификации, багажа знаний и интересов.

Кто-то становится руководителем проекта исключительно по соображениям служебной карьеры, хотя управление проектами не приносит им никакого удовлетворения. Таким людям не хватает знаний и квалификации, чтобы работать программистами, тестерами, инженерами баз данных, проектировщиками, аналитиками, архитекторами, и им приходится заниматься руководством. Многих из них вполне устроит ликвидация роли руководителя проекта, поскольку это даст им возможность вернуться к той работе, которая доставляет им наибольшее удовлетворение.

Другие руководители проекта пользуются своими ролями для пополнения багажа знаний о бизнесе и клиентах. Руководитель проекта, оказавшийся в такой ситуации, воспользуется знаниями, полученными в новой роли, чтобы переквалифицироваться во владельца продукта. Это может быть превосходным вариантом, особенно для руководителя проекта, который чувствует себя не очень-то комфортно, полностью отказываясь от возможности руководить действиями своей команды. Владельцу продукта разрешается время от времени подсказывать команде, *что* ей следует делать; главное, чтобы он не подсказывал команде, *как* ей следует это делать. Это может оказаться неплохим компромиссом для бывшего руководителя проекта, которому иногда очень нелегко удержаться от того, чтобы хоть немного не поруководить своей командой.

Если руководителю проекта удается избавиться от своих старых привычек управления командой и принятия решений вместо нее, то у него есть неплохие шансы стать эффективным Scrum-мастером. Роль Scrum-мастера является типичной новой ролью для руководителей проекта в организациях, переходящих к использованию Scrum. Для бывшего руководителя проекта эта роль поначалу может быть довольно трудной: ему нужно научиться держать язык за зубами и предоставить возможность команде самостоятельно решать свои проблемы. Зачастую начинающим Scrum-мастерам приходится учить свои команды тому, в чем они сами не очень-то сильны, — быть гибкими. Ниже перечислены оптимальные стратегии для Scrum-мастера в подобной ситуации.

- **Внедрять Scrum, как можно строже следуя рекомендациям, изложенным в этой книге.** Прежде всего, необходимо как можно строже следовать рекомендациям, изложенным в этой или какой-либо другой книге по Scrum. Можно также воспользоваться услугами приглашенного инструктора или наставника и неуклонно (и по возможности буквально) придерживаться его рекомендаций.

Адаптировать методологию Scrum к специфике своей организации можно лишь после приобретения определенного опыта ее практического применения.

- **Как можно чаще общаться с другими Scrum-мастерами.** Если в вашей организации есть несколько Scrum-мастеров, объединитесь с ними в сообщество практических пользователей Scrum и делитесь как положительным, так и отрицательным опытом использования этой методологии. Постарайтесь выявить что-то общее в опыте этих Scrum-мастеров и извлеките уроки из совместного опыта. Если же вы являетесь единственным Scrum-мастером в своей организации, найдите Scrum-мастеров в других организациях, которые будут готовы поделиться с вами своим опытом. Вы сможете сравнить свои подходы к использованию Scrum с подходами этих Scrum-мастеров и определить, какие из них более эффективны.

См. также Подробнее о сообществах практических пользователей Scrum можно прочитать в главе 17, “Изменение масштаба Scrum”.

- **Стараться научиться как можно большему как можно быстрее.** Читайте книги, статьи, блоги и веб-сайты. Посещайте собрания местных групп, интересующихся гибкой методологией разработки. Посетите одну или несколько крупных конференций, посвященных гибкой методологии разработки или Scrum.

Дорис Форд (Doris Ford), руководитель отдела программных средств компании Motorola, была классически подготовленным руководителем проектов и специалистом по управлению проектами (Project Management Professional — PMP). Несмотря на наличие классической подготовки и опыта традиционного управления проектами подход Дорис отличался тем, что она доверяла своим командам и поддерживала их. Именно поэтому ей было легко сменить роль руководителя проектов на роль Scrum-мастера. Вот что она пишет о том, как изменилось содержание ее работы в результате перехода к Scrum.

За многие годы управления гибкой разработкой проектов я научилась не вникать в подробности тех или иных задач. Будучи традиционным руководителем проектов, я всегда контролировала, кто какие задачи выполняет, каковы взаимозависимости между этими исполнителями и будут ли эти задачи выполнены в срок. Я тратила очень много времени на выяснение этих вопросов, стремясь уложиться в заданные рамки календарного плана, бюджета и качества проекта и докладывая вышеизвестному руководству о ходе его реализации (пользуясь иногда для оценки достигнутого прогресса таким показателем, как полученная прибыль). В условиях гибкой методологии разработки мне пришлось научиться доверять членам команды такую работу, как формулировка и выполнение задач, необходимых для успешного завершения каждого спринта. Поначалу мне было нелегко решиться на это, но я быстро пришла к выводу, что команде вполне по силам справиться с этой работой. Сейчас я трачу большую часть своего рабочего времени на оказание помощи членам команды, убирая препятствия на их пути и устраняя посторонние “шумы”, мешающие им сосредоточиться на своей работе.

Почему изменяется название должности

Если руководитель проекта может стать Scrum-мастером или владельцем продукта какой-либо команды, то какой смысл менять название его должности? Рассмотрим термин *Scrum-мастер*. В прежние времена, когда я выполнял свои первые Scrum-проекты, не было даже такого термина, как *Scrum-мастер*, и мне не приходило в голову называть свою роль как-то иначе, чем *руководитель проекта*. Ничего плохого в этом не было и всех это устраивало. Правда, принимая впоследствии на работу людей, которым предстояло исполнять роль руководителя проекта, я подробно разъяснял, как, по моему мнению, им следует взаимодействовать с командой. Я старался не принимать на работу людей, склонных к доминированию над подчиненными и командному стилю управления. К тому же эти новые руководители проекта находились у меня в подчинении, что позволяло мне корректировать способ их взаимодействия со своими командами. Повторю еще раз: название *руководитель проекта* ничуть меня не смущало.

Поскольку наша компания продолжала преуспевать и расти, мы приступили к поглощению других компаний. В тех компаниях продолжали работать руководители проектов, представления которых об их роли были в основном традиционными. Я пришел к выводу, что должен помочь им переменить их представления таким образом, чтобы они соответствовали гибкой методологии разработки. Оказалось, однако, что это гораздо труднее, чем принять на работу руководителя проекта, придерживающегося принципов сотрудничества с командой, необходимых для налаживания работы самоорганизующихся команд.

Спустя несколько лет в ходе беседы с Кеном Швабером (Ken Schwaber) он помог мне понять, почему трансформация представлений руководителя проекта со стажем оказалась более трудным делом, чем я ожидал. Швабер высказал предположение, что, позволяя руководителю проекта сохранить прежнее название своей должности, я давал ему повод думать, что изменения при переходе к Scrum носят не столь всеохватывающий характер, как это есть на самом деле. В 1997 году Кен Швабер предложил термин *Scrum-мастер* частично потому, что этот термин должен был напоминать каждому о том, что речь в данном случае идет не просто о руководителе проекта, которому несколько функций добавили, а несколько других урезали. Швабер сказал мне, что “словарь Scrum — это словарь перемен. Зачастую предлагаются намеренно огрубленные и неуклюжие термины — «график выполнения оставшихся работ», «журнал запросов на выполнение работ», «Scrum-мастер» — только для того, чтобы напоминать нам о происходящих переменах”.

Хотя я и рекомендую избегать термина *руководитель проекта* в качестве названия должности, делать это вовсе необязательно. Если вам или вашей организации так нравится эти слова, пользуйтесь ими. Но не забывайте о совете Кена Швабера и моем собственном опыте, суть которых сводится к тому, что употребление старых терминов замедляет переход к использованию новой методологии. Использование термина *руководитель проекта* мешает людям мыслить по-новому. Более того, если люди не желают расставаться с таким пустяком, как старое название должности, то они наверняка не согласятся и на куда более серьезные перемены, необходимые для внедрения Scrum.

Архитекторы

Многие архитекторы потратили многие годы своей жизни на то, чтобы получить право занимать столь престижную должность, как *архитектор*. Они по праву гордятся своими знаниями, опытом и умением предлагать элегантные решения технических и экономических проблем. Я пришел к выводу, что тревоги, которые одолевают многих архитекторов, сталкивающихся с необходимостью перехода к Scrum, можно разделить на две следующие категории.

- Будут ли люди по-прежнему реализовывать предлагаемые мною архитектурные решения?
- Как гарантировать, что мы создаем качественный с архитектурной точки зрения продукт при отсутствии фазы предварительной проработки архитектуры?

Ответ на первый из этих вопросов целиком зависит от личности конкретного архитектора. Многие архитекторы приходят к выводу, что в результате внедрения Scrum в их работе вообще мало что меняется. Решения, рекомендуемые этими архитекторами, внедряются потому, что другие разработчики уважают их и прислушиваются к их советам. Если, например, кто-то из моих сотрудников пользовался в прошлом репутацией высококвалифицированного архитектора и я вижу, что он предлагает разумные архитектурные решения для данного проекта, то я наверняка буду обращаться к нему с любыми вопросами, касающимися архитектуры. Я стану поступать так, даже если буду работать в составе самоорганизующейся команды, и никакая сила не заставит меня переменить свое мнение.

Вторая причина, вызывающая беспокойство архитекторов, кажется мне во многом необоснованной. Как будет показано в главах 9, “Технические приемы”, и 14, “Сprintы”, при формировании системы приоритетов в журнале запросов на выполнение работ архитектурные потребности продукта используются в сочетании с экономическими целями. Это дает архитектору возможность сосредоточиться на “архитектурных неопределенностях” в рамках соответствующего приложения. В случае архитектурно сложного или рискованного продукта архитектору придется работать в тесном контакте с владельцем продукта, информируя владельца продукта об архитектурных аспектах пунктов, включенных в журнал запросов на выполнение работ. Все владельцы продукта понимают, что нужно прислушиваться к мнению рынка, пользователей или клиентов, поскольку эти мнения должны сказываться на решениях, касающихся данного продукта. Хорошие владельцы продукта понимают также, что нужно узнат мнение технической команды о приоритетах. Несмотря на то что окончательное решение в любом случае остается за владельцем продукта, при распределении приоритетов работ хорошие владельцы продукта учитывают все точки зрения.

Эндрю Джонстон (Andrew Johnston) из AgileArchitect.org пишет: “При использовании гибкой методологии разработки главной обязанностью архитектора является анализ изменений и сложности, тогда как другие разработчики сосредоточиваются на очередной поставке готовой работы клиенту” (2009). Разумное выстраивание последовательности работ в спринты может помочь команде быстрее добывать нужные знания, минимизировать совокупную стоимость разработки, избегать рисков или выявлять их таким образом, чтобы у команды оставалось достаточно времени для реагирования на эти риски.

Архитектор, не занимающийся кодированием

По-видимому, с самой серьезной трансформацией содержания своей работы придется столкнуться архитекторам, не занимающимся кодированием. Речь идет об архитекторах, которых Скотт Эмблер (Scott Ambler) называет “архитекторами в башне из слоновой кости” (2008b). Само по себе присутствие архитектора, не занимающегося кодированием, является предвестником появления проблем, однако Scrum-проекты в гораздо большей степени от них защищены. Кое-кто из архитекторов, не занимающихся кодированием, рассматривает переход к Scrum как шанс вновь заняться программированием, которым они когда-то с успехом занимались. Такие архитекторы вполне могут пригодиться Scrum-команде. Их будут уважать за глубину их познаний и богатство опыта, а также за то, что они не гнушаются закатать рукава и заняться кодированием.

Опасайтесь архитектора, который противится новой роли, требующей конкретного вклада в реализацию проектов. Во многих случаях эти архитекторы, не занимающиеся кодированием, стали таковыми лишь потому, что не желали заниматься практическим программированием. Один такой архитектор, Том, привел меня в замешательство, когда я познакомился с ним. Он производил приятное впечатление и казался человеком, хорошо разбирающимся в технологиях. Однако он оказался первым из известных мне разработчиков, которому нравилось принимать участие во всевозможных совещаниях. Он вообще старался не пропускать ни одного совещания. Когда я познакомился с ним поближе, то понял, что его технические познания носят довольно поверхностный характер. Он оказался не таким уж хорошим специалистом, как я думал поначалу. Вскоре я понял, почему ему нравится устраивать столь продолжительные совещания со своей командой: во время любого такого необязательного совещания все его участники кажутся в равной мере продуктивными и ценными работниками. Когда же члены команды усаживаются за свои рабочие столы и приступают к реальной работе, тогда-то и начинают проявляться существенные различия между разработчиками. Страсть Тома к ненужным совещаниям была своеобразным механизмом самозащиты: чем больше времени команда тратила на совещания, тем больше времени требовалось каждому, чтобы понять истинный уровень квалификации Тома.

Чтобы быть ценным работником, человеку, носящему звание *архитектора*, вовсе необязательно постоянно заниматься кодированием. Не беда, если в течение одного-двух спринтов архитектор не будет этого делать. Сказанным выше я хотел лишь подчеркнуть разницу между архитекторами, умеющими кодировать, и архитекторами, никогда не блеставшими особыми способностями к кодированию. Архитектор программного обеспечения Йоханнес Бродуолл (Johannes Brodwall) говорил, что “самое крупное изменение в моей роли как архитектора заключалось в том, что с формальной точки зрения архитектор уже не может диктовать разработчикам свои технические решения. Вместо этого ему приходится выступать в роли советника и помощника. Выступая в роли советника, я получаю возможность способствовать лучшему выполнению работы, по поводу которой я консультирую разработчиков”.

Функциональные менеджеры

Функциональные менеджеры (такие, как руководители разработки, начальники службы технического контроля и т.п.), которые привыкли работать “матричным” способом, могут продолжать работать в том же духе и после перехода к выполнению

Scrum-проектов. Типичный функциональный менеджер после перехода к Scrum, наверное, столкнется с некоторым сокращением своих властных полномочий, но в значительной степени это будет зависеть от того, как была определена его роль в данной организации до перехода к Scrum.

Функциональные менеджеры обычно сохраняют за собой функцию распределения людей по проектам. Предполагается, что они продолжат исполнять эту обязанность, исходя из конкурирующих потребностей всех проектов, мест, где выполняются эти проекты, потребностей в разработчиках, карьерных устремлений конкретных исполнителей и т.п. В некоторых организациях функциональные менеджеры не ограничиваются распределением людей по проектам и участвуют в распределении задач между членами групп, которыми они руководят. После перехода к Scrum им уже не придется исполнять эту функцию. Индивидуальный выбор работы является фундаментальным аспектом способа самоорганизации членов команды и должен быть предоставлен самой команде.

Лидерская роль функционального менеджера

Функциональные менеджеры всегда выступали в роли лидеров. Тенденции к лидерству в широком смысле этого слова с годами оказывали влияние на индивидуальный стиль руководства. Например, когда я был еще подростком, мой отец был управляющим магазинов компании Sears. Это было в те далекие времена, когда Sears даже не мечтала о том, чтобы стать крупнейшей в мире сетью розничной торговли. Стиль руководства, которого придерживался мой отец, можно обозначить как командный (его еще называют руководством по принципу “сверху вниз”). Отец формулировал цели, определял квоты и прочие показатели, доводил эти показатели до сведения служащих магазинов, а затем сравнивал результаты труда каждого работника с целевыми показателями. Это была эпоха, когда преобладало мнение, что хороший менеджер может управлять буквально всем. Мне казалось, что опыт, накопленный моим отцом, позволит ему с одинаковым успехом управлять магазином, банком или производственным предприятием. Если исходить из рис. 8.1, то мой отец относился к той категории руководителей, для которой отведен нижний левый квадрант. Схема, представленная на этом рисунке, заимствована из книги Джейфри Лайкера (*The Toyota Way* (2003, 181)).

Другой тип руководителя (или, возможно, руководителя, работающего гораздо позже моего отца) мог бы применять свои управленческие способности для управления людьми по принципу “снизу вверх”. Этот тип руководителя относится к той категории, для которой на рис. 8.1 отведен верхний левый квадрант. В нижнем правом углу этого рисунка отведено место для той категории руководителей, которые отличаются глубоким пониманием конкретной работы и придерживаются управления по принципу “сверху вниз”. Такой тип руководителей (который очень часто встречается среди руководителей проектов, связанных с разработкой программного обеспечения) говорит своей команде, *что делать и как делать*.

В организации, использующей методологию Scrum, функциональным менеджерам следует работать в правом верхнем квадранте, сочетая глубокое понимание конкретной работы и управление по принципу “снизу вверх”. Функциональный менеджер обучает и наставляет членов своей группы. Scrum-мастера и владельцы продукта также занимаются обучением и наставничеством, но их интересы ограничены отдельно взятым

проектом или продуктом. Взгляды функционального менеджера отличаются большей широтой; он может устанавливать стандарты, распространяющиеся на многие проекты, и формировать ожидания, касающиеся качества, удобства обслуживания и эксплуатации, а также многих других показателей или нефункциональных требований.

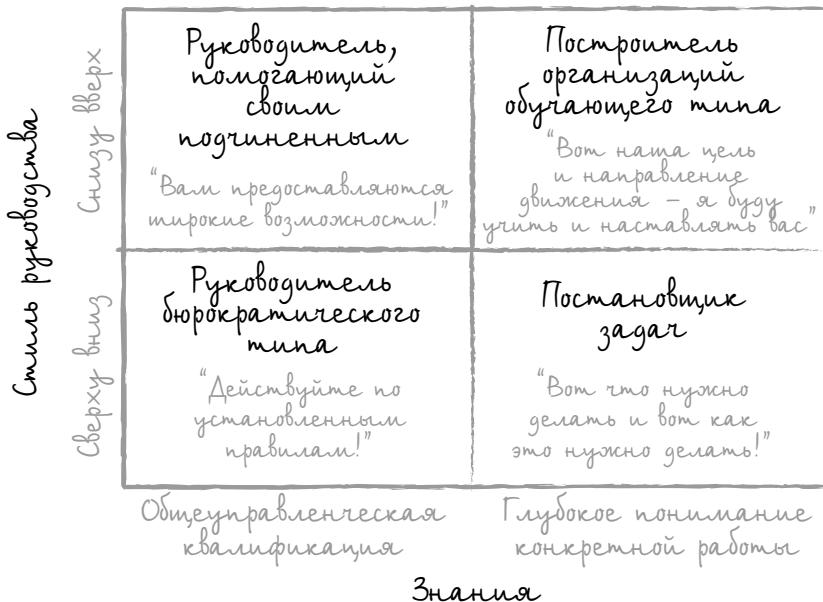


Рис. 8.1. Типы функциональных менеджеров, определяемые типом квалификации и стилем руководства (заимствовано из книги Джонни Лайкера *The Toyota Way*, авторское право на которую принадлежит The McGraw-Hill Companies, Inc.)

Функциональные менеджеры по-прежнему несут ответственность за повышение квалификации и профессиональный рост своих подчиненных. Резервирование средств и времени для участия работников в конференциях, выбор для них подходящих проектов, стимулирование работников к вступлению в сообщества практических пользователей Scrum (или к созданию таких сообществ) — роль функционального менеджера предусматривает исполнение всех этих обязанностей.

См. также О сообществах в поддержку совершенствования организации уже рассказывалось в главе 4, “Движение в направлении гибкости”. Подробнее о сообществах практических пользователей Scrum можно прочитать в главе 17, “Изменение масштаба Scrum”.

Обязанности, связанные с управлением кадрами

В большинстве организаций за функциональными менеджерами остается обязанность проведения периодических аттестаций работников своих отделов. Несмотря на то что при проведении аттестации функциональный менеджер всегда учился (во всяком случае, должен был учитывать) мнения сотрудников и клиентов

аттестуемого работника, после перехода к Scrum эти мнения становятся еще более актуальными, поскольку в результате внедрения Scrum повседневные контакты работника с функциональным менеджером ослабляются.

См. также О периодических аттестациях работников рассказывается в главе 20, “Кадры, техническое обеспечение и отдел управления проектами”.

Во многих организациях функциональные менеджеры продолжают принимать решения, касающиеся приема на работу и увольнения сотрудников. Ни Scrum-мастер, ни владелец продукта таким правом в отношении команд, занимающихся разработкой новых продуктов, не обладают.

После перехода организации к использованию Scrum у большинства функциональных менеджеров появляется больше времени, которое они могут посвятить решению других проблем. В частности, это время они могут посвятить более тесному общению со своими непосредственными подчиненными, ближе познакомиться с каждым проектом, над которым работают члены их группы (например, принимая участие в совещаниях, посвященных обзору спринтов, и т.п.), и уделять больше внимания выработке стандартов, распространяющихся на многие проекты, и направлений будущей деятельности своих групп.

Программисты

Чем занимаются в Scrum-команде программисты? Они занимаются программированием, тестированием, анализом, проектированием. Иными словами, они занимаются всем, что необходимо их команде для успешного завершения каждого очередного спринта. Несмотря на то что наличие специалистов в Scrum-команде можно только приветствовать, эти специалисты должны быть готовы к выполнению работ, не имеющих непосредственного отношения к их специальности, — если, конечно, выполнение этих работ отвечает интересам команды. Как и везде, бывают и исключения. Например, в проекте, связанном с разработкой новой компьютерной игры, может потребоваться участие специалистов по программированию искусственного интеллекта. В силу высокоспециализированной природы их продукта эти специалисты вряд ли смогут заниматься чем-либо другим, кроме исполнения своих непосредственных, прямых обязанностей. Однако большинство программистов в Scrum-команде должно быть готово к выполнению любой работы, которая помогает достичь целей, поставленных перед командой. Это означает, что в случае необходимости им придется заниматься тестированием, время от времени писать программы на “нелюбимых” языках программирования и т.п.

См. также Тема специалистов подробно рассматривается в главе 11, “Организация коллективного труда”.

Одна из наиболее впечатляющих перемен, имеющих непосредственное отношение к программистам Scrum-команды, заключается в том, что они уже не могут, как прежде, сидеть в своих отсеках и ждать, когда им скажут, какие именно функциональные

возможности они должны программировать. Программисты должны стать активными участниками процесса выяснения требований к будущему продукту. Как это ни странно, но очень многие хотят, чтобы им просто сказали, что именно следует делать. Мне приходилось слышать такие высказывания: “Если мне скажут, что мне следует делать, и я сделаю это, то меня не смогут уволить”. Предполагается, что программисты, подобно всем другим членам команды, несут совместную ответственность за успех своего продукта. Если все члены команды ощущают эту ответственность, как правило, бывает легче делать то, что не входит в круг твоих непосредственных обязанностей.

Предполагается, что программисты общаются с клиентами и пользователями. Объем такого общения может корректироваться в ту или другую сторону в зависимости от особенностей конкретного программиста, организации, сильных сторон других членов команды и природы осуществляемого проекта. Программистам нет нужды вырабатывать у себя качества исключительно коммуникабельных и радушных торговых представителей компаний. Но они не должны стесняться говорить (хотя бы по телефону) с кем-либо из клиентов или пользователей.

Точно так же программистам наверняка придется больше общаться и с другими членами команды. Пожалуй, им нужно будет расстаться с привычкой приходить на работу к 11 часам утра, надевать наушники, чтобы отключиться от общения с окружающими, и тихо отправляться домой в 19.00. Скорее всего, программистам придется сидеть в общем помещении вместе с остальными членами команды, участвовать в обсуждениях, помогать другим в решении тех или иных проблем и заниматься парным программированием.

См. также Парное программирование, предполагающее работу двух программистов за одним компьютером, подробно рассматривается в главе 9, “Технические приемы”.

Такие перемены могут оказаться весьма нежелательными для многих программистов (такими они были и для меня), которые выбрали эту профессию именно потому, что она позволяла им работать совершенно автономно. До того как получить свое первое в жизни задание в качестве программиста, я работал в темной, совершенно изолированной от окружающего мира комнатушке размером четыре на шесть футов, целый день проявляя фотопленки. Я покидал эту комнатушку только во время обеденного перерыва и других перерывов, предусмотренных распорядком дня в нашей организации. Все остальное время я работал один в маленьком темном помещении, но это мне нравилось. Перемещение в светлый мир отсеков, где трудились программисты, стало огромным событием в моей жизни. Переход из тихого отсека в мир повседневного живого общения с множеством сотрудников стал для меня еще более грандиозным событием. К такой перемене в своей жизни должен быть готов каждый из программистов, которому предстоит работать в Scrum-команде. К счастью, подобные перемены для большинства из нас не станут фатальными. Кому-то из нас нравится работать в одиночестве, однако участие в структурированных обсуждениях (как во время совещаний и в ходе принятия решений при выполнении типичного Scrum-проекта) все же не так давит на психику, как участие в неструктурированных обсуждениях (например, во время какой-нибудь званой вечеринки, где присутствует много незнакомых вам людей).

Программистам почти наверняка придется столкнуться не только с переменами, связанными с общением и взаимодействием, но и с переменами, касающимися способа выполнения ими своей работы. Многие из технических приемов, описанных в главе 9, “Технические приемы”, покажутся им необычными. На первое время команда может отказаться от использования некоторых из этих приемов (в отдельных случаях команда может вообще от них отказаться), но я все же порекомендовал бы попробовать их все, прежде чем отказываться от некоторых.

Администраторы базы данных

Специалисты по данным, какое бы звание они ни носили (администраторы базы данных, инженеры базы данных и т.п.), могут оказаться в числе самых ярых противников перехода к Scrum. Многое из того, о чем говорилось в предыдущем разделе о программах, применимо и к администраторам базы данных. Кроме того, специалистам по данным придется научиться постепенно делать то, что традиционно рассматривалось как подготовительная работа к реализации проекта.

Стандартный совет при разработке базы данных — провести полный анализ потребностей соответствующей системы, разработать логическую, или концептуальную, базу данных, а затем, в ходе разработки физической базы данных, отобразить эти концепции на ограничения реальной базы данных. Успех этой последовательности шагов основывается на полном и точном предварительном анализе. Этот традиционный подход специалиста по данным превосходно резюмировал для меня человек, вместе с которым я однажды летел из Чикаго в Сакраменто. Он занимал пост вице-президента по разработке баз данных одной довольно крупной компании, представлявшей здравоохранение. Его взгляд на мир можно выразить следующими словами: “Приложения появляются и исчезают, вечны лишь данные”.

Подобный тип мышления неминуемо заставляет нас сосредоточиться на проведении полного предварительного анализа. В теории все выглядит просто замечательно, но на практике, пока мы затрачиваем какое-то время на проведение полного анализа, мир не топчется на месте, а продолжает меняться. Меняются потребности пользователей, конкуренты выпускают новые продукты. Базы данных должны развиваться, чтобы поддерживать изменяющиеся приложения, которые пользуются этими базами данных.

В главе 14, “Спринты”, я покажу, что проектирование пользовательского интерфейса, а также архитектура и разработка баз данных являются особыми случаями одной и той же ситуации: работы по методу постепенного наращивания и совершенствования чего-либо, что рассматривается в целостном виде. Большая часть повседневной работы администратора базы данных не претерпевает существенных изменений; радикальные изменения касаются лишь того, как администратор базы данных подходит к этой работе и планирует ее. Эти радикальные изменения мы рассмотрим подробно в одном из разделов главы 14, “Спринты”.

Тестеры

Многие годы типичный подход к тестированию основывался на определении качества, предложенном Филиппом Кросби (Philip Crosby): качество — это соответствие требованиям (1979, 16). Если качество — это соответствие требованиям, то было бы

неплохо, чтобы эти требования были сформулированы в письменном виде. Это обусловливало горячее стремление многих тестеров получить в свое распоряжение некий идеальный документ с перечислением требований, руководствуясь которым, они могли бы оценивать качество системы. Однако сколь бы справедливым ни казалось нам указанное выше определение качества как соответствия требованиям, соответствие потребностям пользователей представляется еще более точным определением качества. При использовании Scrum мы можем утверждать, что идеально предсказать все потребности пользователей не представляется возможным.

Точно так же, как программисты уже не могут сказать: “Дайте мне самые полные спецификации и оставьте меня в покое до тех пор, пока я разработаю систему, которая будет делать именно то, что вы требуете от меня”, тестеры уже не могут сказать: “Дайте мне документ с полным перечнем требований — и я проверю, делает ли система все то, что указано в этих требованиях”. Любой из таких подходов (а именно такие подходы превалировали в проектах, которые управлялись традиционными методами) ведет к отказу от ответственности. Когда программисты или тестеры провозглашают подобные подходы, это, по сути, означает, что они отказываются брать на себя ответственность за конечный успех соответствующего проекта. Они говорят: “Скажите, что я должен сделать, и я сделаю это”. Однако им следовало бы думать о продукте, задавать вопросы о каждой его функциональной возможности и думать о том, как та или иная функциональная возможность влияет на ценность продукта для его будущих пользователей.

См. также Переход от составления документов, в которых излагаются требования, к обсуждению требований подробно рассматривается в главе 13, “Журнал запросов на выполнение работ”.

Поскольку на этапе сбора информации о требованиях Scrum-команды переносят акцент с составления документов, в которых излагаются требования, на обсуждение требований, общение с владельцем продукта становится для тестера основным способом выяснения, что должна представлять собой та или иная функциональная возможность. Тестер должен обсудить с владельцем продукта, как должна проявляться эта функциональная возможность, каким должно быть ее быстродействие, каким приемочным критериям она должна удовлетворять и т.п. Задача тестера не ограничивается получением подобной информации лишь от владельца продукта. При необходимости тестер может поговорить с пользователями, заказчиками и другими заинтересованными сторонами.

Как и в случае с программистами, работа в такой интерактивной среде может оказаться некомфортной для тестеров, переходящих к Scrum. Многие тестеры, подобно своим коллегам, занялись разработкой программного обеспечения в расчете на то, что это позволит им целый день сидеть в своем отсеке, сведя к минимуму общение с внешним миром. При использовании методологии Scrum они уже не смогут позволить себе подобную роскошь. Тестерам в Scrum-команде придется привыкнуть к более частым деловым разговорам с товарищами по работе, а во многих случаях и с теми, кто не являются членами команды.

Наряду с отказом от мифа, будто загадка может быть написана некая идеальная спецификация, одна из крупнейших перемен, с которыми придется столкнуться

тестеру, заключается в том, что ему нужно будет научиться работать итеративно, т.е. методом последовательных приближений. Концептуально это не так уж сложно. Если каждый спринт рассматривать как самостоятельный проект, то тестирование для каждого такого проекта/спринта выполняется в рамках этого спринта. Однако это вовсе не означает, что нужно лишь объявить последнюю неделю каждого спринта неделей тестирования. Это не только не приведет к нужному результату, но, напротив, создаст своего рода миниатюрные “водопады” внутри каждого спринта. В течение нескольких первых спринтов тестеры столкнутся с колossalной проблемой. В это самое время программистам также придется учиться работать методом последовательных приближений, и, вероятно, у них также не все будет гладко. Скорее всего, команда не сможет правильно рассчитать свои силы (в смысле того объема работы, который она в состоянии выполнить в течение одного спринта), а программистам удастся полностью закодировать все запланированные возможности лишь к самому окончанию спринта. Поэтому они попытаются вручить готовый код тестерам на восемнадцатый день 20-дневного спринта. После того как люди, исполняющие эти роли, научатся работать в рамках гибкой методологии разработки, подобные авральные ситуации исчезнут сами по себе.

См. также Советы о том, как организовать совместную работу тестеров, программистов и других членов команды, можно найти в главе 11, “Организация коллективного труда”.

См. также Разъяснение того, почему команда может одновременно пользоваться несколькими инструментами автоматизации тестирования, можно найти в главе 16, “Качество”.

Особый акцент на автоматизации тестов становится отличительным признаком Scrum-команд. Даже команды, которые годами нехотя осваивали искусство автоматизации тестов, приходят к выводу, что короткие спринты Scrum делают автоматизацию тестов жизненно необходимой. Со временем команды перестают полагаться на тестирование вручную и на тестеров, которые читают сценарий, нажимают определенную кнопку и фиксируют результаты. Таким тестерам часто предлагают освоить один или более инструментов автоматизации тестирования, используемых командой. Хотя при создании тестов некоторые из инструментов автоматизации тестирования используют то, что можно было бы назвать программированием, это относится далеко не ко всем инструментам автоматизации тестирования. Насколько я могу судить по собственному опыту, лишь очень немногие из тех, кто занимается ручным тестированием, не смогли оказать своим командам существенной помощи в деле автоматизации тестирования. С другой стороны, я встречал многих тестеров, занимавшихся ручным тестированием, которых страшила перспектива подобных перемен в их жизни. Наилучшими лекарствами от таких страхов являются время, практика, обучение и работа в паре с кем-либо из коллег (в том числе с кем-либо из программистов).

Лайза Криспин (Lisa Crispin), написавшая в соавторстве с Джанет Грегори (Janet Gregory) книгу *Agile Testing*, вспоминает, что, когда она начала работать в составе Scrum-команды, первое, на что она обратила внимание, была необходимость действовать в режиме упреждения.

Ни в коем случае не следует сидеть и ждать, пока к вам кто-то обратится. Действуйте в режиме упреждения! Мы, тестеры, не имеем права ждать, пока кто-то попросит нас протестировать то или иное приложение. Мы сами должны искать для себя работу. Сотрудничество с программистами покажется необычным для большинства тестеров. (Хотя я не могу сказать этого о себе: какой бы процесс мы ни использовали, я всегда проявляла инициативу с самого начала выполнения проекта.) Необходимость сотрудничества с клиентами также окажется новостью для многих тестеров. Все эти новшества вызывают у большинства тестеров ощущение дискомфорта. Программисты — занятые люди, кажущиеся подчас замкнутыми и малообщительными. Когда я была единственным тестером в команде из восьми программистов, мне было нелегко обратиться к ним за помощью, даже несмотря на то, что с большинством из них я работала ранее в другой компании.

ВОЗРАЖЕНИЕ

“Если я буду работать в чересчур тесном контакте с другими членами команды, то у меня постепенно выработается «программистский» взгляд на вещи. Это заставит меня смотреть на все глазами программиста, а не глазами тестера”.

Честно говоря, мне трудно понять, как работа в более тесном контакте с программистами может заставить тестера утратить собственную точку зрения настолько, что он окажется не в состоянии качественно тестировать программное обеспечение. Специалисты по базам данных, годами работавшие в тесном контакте с программистами, отнюдь не утратили свою квалификацию. Не один десяток лет тестеры отставали оттестирование как по принципу “белого ящика” (когда они могли видеть внутреннее содержимое соответствующей системы), так и по принципу “черного ящика” (когда они не могли видеть этого содержимого). Если работа в тесном контакте с программистами может выработать у тестера “программистский” взгляд на вещи, то почему бы нам не считать, что тестер, который долгое время выполнял тестирование по принципу “белого ящика”, точно так же утратил собственный взгляд на вещи и не сможет выполнять тестирование по принципу “черного ящика”? К счастью, однако, на практике ничего подобного не происходит.

Хотя поначалу многие перемены, которые влечет за собой Scrum, могут вызвать у людей дискомфорт, многим тестерам наверняка понравятся новые методы работы, после того как они к ним привыкнут. Джайри Партанен (Juuri Partanen) занимает должность начальника службы технического контроля в компании Sulake, которая разработала Habbo — виртуальное пространство, в котором ежемесячно обретается в среднем свыше восьми миллионов посетителей. Вот как описывает Джайри Партанен перемены, с которыми приходится сталкиваться тестерам.

Тестирование — это профессия, в которой очень сильны старые привычки. При переходе к гибкой методологии разработки приверженность старым способам работы может привести к половинчатому воплощению принципов этой методологии разработки. Обычно дискомфорт, который испытывают специалисты по тестированию, вызван опасениями потерять работу и переменами, которые предстоящий переход к гибкой методологии разработки может вызвать в их повседневной

деятельности. Однако такие опасения беспочвенны. Основываясь на своем опыте, а также на опыте других людей, завершивших переход к гибкой методологии разработки в службе технического контроля, могу с уверенностью сказать, что этот переход, несомненно, оказался правильным, разумным решением. Специалисты по тестированию в составе Scrum-команд способны оказывать большое влияние на процесс разработки и, что еще важнее, на конечный продукт.

Проектировщики пользовательского интерфейса

У проектировщиков пользовательского интерфейса (User Experience Designers — UED) зачастую имеются законные основания беспокоиться в связи с последствиями внедрения Scrum. Хотя они привыкли действовать методом последовательных приближений, они предпочитают выполнять свои итерации с некоторым опережением относительно прочих участников проекта. Однако в случае Scrum-проекта было бы нежелательным, чтобы вся работа UED была выполнена до начала другой деятельности по реализации проекта.

Больше всего мне нравится описание работы проектировщиков, пользующихся гибкой методологией разработки, составленное компанией Autodesk из Торонто. Линн Миллер (Lynn Miller) (2005) и Дезире Сай (Desirée Sy) (2007) написали о подходе, которым они пользовались для интеграции проекта в процесс гибкой методологии разработки. Я принимал участие в реализации не одного десятка проектов, исполнители которых пользовались советами упомянутых авторов.

По утверждению Миллер и Сай, в проекте должны присутствовать две параллельные линии работы: одна относится к собственно разработке, а другая — к проектированию взаимодействия. Эти две линии работы и взаимодействие между ними показаны на рис. 8.2. Важно здесь то, что активность проектировщиков всегда опережает действия разработчиков по меньшей мере на один спринт. Проектировщикам предоставляется право первоначального запуска проекта за счет сочетания исходного “нулевого” спринта и акцента в первом спринте на реализации таких функциональных возможностей, в которых пользовательский интерфейс либо вообще отсутствует, либо присутствует лишь в минимальном объеме.

Показанный здесь подход вполне работоспособен, однако риск заключается в том, что проектировщики начинают рассматривать себя как отдельную команду. Линн Миллер, предложившая первую версию этой схемы, согласна с тем, что ее не следует интерпретировать таким образом, будто она предполагает существование каких-то отдельных команд.

Излагая слушателям эту концепцию, я всегда подчеркиваю, что проектировщикам пользовательского интерфейса не следует рассматривать себя как отдельную команду; чтобы эта концепция доказала свою высокую эффективность, необходимо тесное и частое общение. Глядя на эту схему, люди нередко ошибочно полагают, что речь здесь идет о каких-то отдельных командах, тогда как я имела в виду совсем не это.

Важно, чтобы проектировщики рассматривали себя в качестве членов единой команды. Идея многофункциональных команд является фундаментальной для Scrum: в составе единой команды должны быть все специалисты, которые могут обеспечить

переход от идеи к практическому воплощению. Что мешает группе тестирования представить собственную версию рис. 8.2, на которой были бы показаны параллельные пути программирования и тестирования в ходе выполнения спринтов?

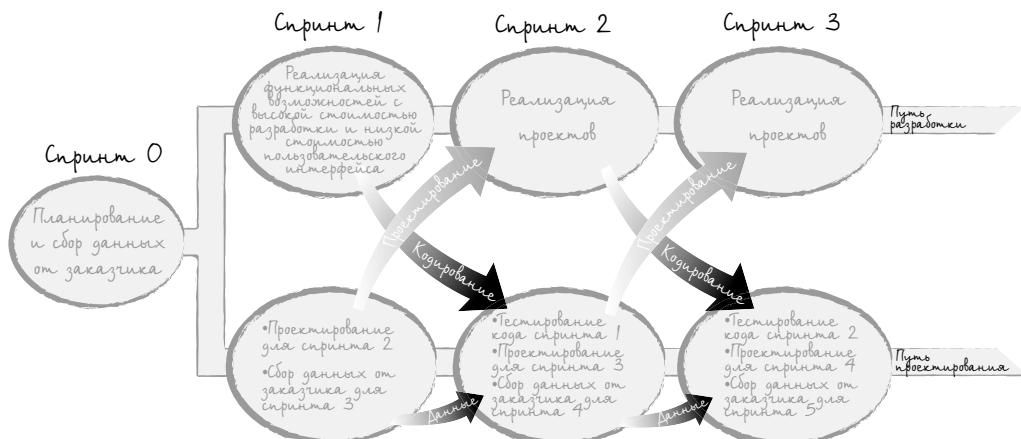


Рис. 8.2. Проектирование и разработка могут двигаться параллельными курсами (публикуется с любезного разрешения Линн Миллер)

Если бы я встретил кого-либо из проектировщиков в коридорах вашей компании и спросил у него: “Чем вы занимаетесь?”, то мог бы услышать, например, следующее: “Я являюсь проектировщиком пользовательского интерфейса. Я опережаю разработчиков на один спринт. Моя задача заключается в том, чтобы к тому моменту, когда они начнут очередной спринт, я мог вручить им проект, в соответствии с которым они будут осуществлять разработку в ходе этого спринта”. Хотя этот ответ и соответствует тому, что мы видим на рис. 8.2, он мне не нравится. Я предпочел бы услышать, например, такой ответ: “Я являюсь проектировщиком пользовательского интерфейса. Я являюсь членом команды разработчиков, а моя основная задача заключается в том, чтобы мы успешно завершили работу, запланированную для текущего спринта. У меня остается свободное время, значительную часть которого я затрачиваю на обдумывание того, что мы создадим в течение одного-двух последующих спринтов. Затем я собираю данные, создаю что-то наподобие макета проекта, а также делаю все, что в моих силах, чтобы к тому моменту, когда мы приступим в будущем спринте к реализации какой-либо функциональной возможности, мы смогли успешно реализовать ее к окончанию этого спринта”.

Обе эти придуманные мною цитаты описывают в точности одну и ту же работу. В обоих случаях проектировщики в ходе спринта работают с командой над разрешением проблем, касающихся этого спринта; наряду с этим они думают о будущем. Однако эти разные ответы отражают разные представления о деятельности. Главное, чего я хочу, — чтобы проектировщики ощущали себя частью единой команды и чтобы важнейшим приоритетом их деятельности было успешное завершение работы, запланированной на текущий спринт. Кроме того, они обязаны думать о будущем точно так же, как и владелец продукта, от которого мы ожидаем прогнозирования действий конкурентов, будущих пожеланий пользователей и т.п.

См. также В главе 14, "Сprintы", мы рассмотрим гораздо подробнее, как проектировщики пользовательского интерфейса могут обеспечивать эффективное перекрытие своей работы с работой других членов команды.

Не я один полагаю, что мышление в духе гибкой методологии разработки необычайно важно для проектировщиков при переходе к использованию Scrum. Это мнение разделяет, в частности, такой общепризнанный эксперт по эргономичности, как Джейкоб Нильсен (Jakob Nielsen).

Для специалистов по пользовательскому интерфейсу, работающих в составе Scrum-команд, самой важной переменой является изменение образа мышления. Наличие обширных познаний в области пользовательского интерфейса поможет вам [специалистам по пользовательскому интерфейсу] понять, как изменить традиционное проектирование и методы оценки, чтобы соответствовать перемене акцентов, которая произошла в вашей Scrum-команде. Однако в конечном счете, если вы хотите добиться успеха, вы должны верить в себя и использовать на практике концепции гибкой методологии разработки. Если вы готовы изменить методы своей работы и взять на себя ответственность, то у вас появляются великолепные возможности повысить свою эффективность и усилить свое влияние на команды, в которых вы работаете (2008).

Три общие темы

В этой главе мы рассмотрели изменившиеся роли аналитиков, руководителей проектов, архитекторов, функциональных менеджеров, программистов, администраторов базы данных, тестеров и проектировщиков пользовательского интерфейса. Во время этого рассмотрения красной нитью проходили три основные темы.

- **Необходимость инкрементности.** Следует стремиться к концу каждого спринта создавать промежуточный продукт, потенциально готовый к поставке заказчику, — что возможно лишь путем последовательного наращивания функциональных возможностей продукта.
- **Необходимость итеративности.** Функциональные возможности продукта могут корректироваться и совершенствоваться от спринта к спринту.
- **Выход за пределы специализации.** Чтобы можно было к окончанию каждого спринта создавать промежуточный продукт, потенциально готовый к поставке заказчику, люди должны быть готовы работать, не ограничиваясь лишь своей узкой специализацией, если того требуют интересы команды.

Было бы неплохо, если бы по мере чтения последующего материала вы помнили замечания о том, как должны работать члены Scrum-команды.

Дополнительная литература

Ambler, Scott. Agile Data Home Page. <http://www.agiledata.org>.

Полезный веб-сайт, на котором собран ряд статей о применении гибкой методологии разработки в средах, характеризующихся огромными объемами данных. Эти статьи принадлежат перу весьма плодовитого автора — Скотта Эмблера.

Crispin, Lisa, and Janet Gregory. 2009. *Agile Testing: A practical guide for testers and agile teams*. Addison-Wesley Professional.

Ичерпывающее руководство по тестированию при выполнении проекта с применением гибкой методологии разработки. Книга начинается с изложения десяти принципов тестирования при использовании гибкой методологии разработки. Затем приводится описание организационных перемен, которые почтует на себе группа тестирования или группа контроля качества. «Изюминкой» этой книги является графическое представление тестирования в виде четырех квадрантов. Многие команды находят такой способ представления тестирования весьма полезным.

Highsmith, Jim. 2009. *Agile project management: Creating innovative products*. 2nd ed. Addison-Wesley Professional.

Самая популярная книга по управлению проектами, выполняемыми с применением гибкой методологии разработки. Во втором издании добавлены главы по планированию выпусков, изменению масштаба, управлению и измерениям, хотя содержание первого издания уже казалось исчерпывающим.

Jeffries, Ron. 2004. *Extreme programming adventures in C#*. Microsoft Press.

Увлекательная книга, в которой рассказывается о том, как Рон Джессифрис изучает C# путем парного программирования с Четом Хендриксоном (хотя автор и не собирался писать учебник по программированию на C#). Эта книга является пре-восходным введением в тот тип программирования с быстрой обратной связью, который практикуется Scrum-командами.

Johnston, Andrew. 2009. The role of agile architect, June 20. Материал с веб-сайта Agile Architect. <http://www.agilearchitect.org/agile/role.htm>.

На этом веб-сайте содержится информация исключительно для архитекторов, пользующихся гибкой методологией разработки. Из всех материалов, размещенных на веб-сайте, данная статья показалась мне самой полезной. В ней содержится описание пяти ключевых целей и семи «золотых правил» архитектора, пользующегося гибкой методологией разработки.

Krug, Steve. 2005. *Don't make me think: A common sense approach to web usability*. 2nd ed. New Riders Press.

Это — наилучшая из известных мне книг, в которых излагается дискоинтный подход к эргономичности. Опытным проектировщикам пользовательского интерфейса такой подход может показаться чрезмерно упрощенным. Им может также показаться, что он игнорирует результаты очень многих исследований. Между тем такой подход кажется многим командам весьма полезным.

Marick, Brian. 2007. *Everyday scripting with Ruby: For teams, testers, and you*. Pragmatic Bookshelf.

Эта превосходная книга предназначена для тестеров, которые желают освоить базовые приемы, используемые при написании сценариев с помощью языка Ruby (возможно, это нужно им для успешной работы в составе Scrum-команды). Однако эта книга может пригодиться каждому, кому необходимо хорошее введение в Ruby.

Sliger, Michele, and Stacia Broderick. 2008. *The software project manager's bridge to agility*. Addison-Wesley Professional.

Слайджер и Бродерик являются преподавателями Scrum, а также специалистами по управлению проектами (Project Management Professional — PMP). Их книга предназначена для таких же PMP, как они сами, которые переходят к использованию Scrum или любой другой гибкой методологии разработки.

Subramaniam, Venkat, and Andy Hunt. 2006. *Practices of an agile developer: Working in the real world*. Pragmatic Bookshelf.

В этой небольшой книге содержится примерно 50 рекомендаций, предназначенных для программистов, которые участвуют в реализации любого проекта, выполненного с помощью гибкой методологии разработки. Каждая из этих рекомендаций (например, “Управлять проектом — не значит диктовать”) включает описание соответствующего приема и наиболее вероятных результатов его успешного применения.

Глава 9

Технические приемы

Новые названия должностей, появление новых ролей и обязанностей — не единственные перемены, необходимые для успешного функционирования Scrum-команд. Чтобы функционирование Scrum-команды было по-настоящему успешным, она не должна ограничиваться внедрением лишь базовых, лежащих на поверхности элементов Scrum. Она должна быть готова к подлинным переменам, связанным со способом выполнения практической работы по созданию продукта. Мне приходилось видеть команды, которые работали спринтами, правильно организовывали совещания по планированию и обзору спринтов, не пропускали ни одного из ежедневных совещаний разработчиков, а в конце каждого спринта обязательно проводили ретроспективный обзор. Такие команды добивались ощутимых улучшений, а их производительность в отдельных случаях повышалась в два раза по сравнению с тем, что было до внедрения Scrum. Однако они могли бы добиться значительно больших результатов.

Главное, что упускали из виду такие команды (и что мешало им достичь гораздо большего), — это перемены в используемых ими технических приемах. Scrum не предписывает применения каких-то особых технических приемов — это было бы отходом от базовой философии Scrum (“позвольте команде самой решать поставленную перед ней задачу”). Например, нигде напрямую не сказано, что Scrum требует тестирования продукта. Нигде не говорится и о том, что писать код нужноарами, в духе разработки программного обеспечения на основе составления тестов. Главное, что требует от команд Scrum, — это составление высококачественного и потенциально готового к отправке заказчику кода к концу каждого спринта. Если команды могут делать это, не прибегая к каким-то новым для себя техническим приемам, то никто их за это не осудит. Однако большинство команд находит, внедряет и применяет новые технические приемы, поскольку они облегчают достижение целей, поставленных перед командой.

В этой главе мы рассмотрим пять типичных приемов, которые стали популярны благодаря экстремальному программированию и начали применяться многими Scrum-командами, отличающимися высочайшей производительностью. Мы рассмотрим, как эти приемы появились в результате упорной борьбы за техническое совершенство.

Наконец мы рассмотрим, как технические приемы, которые использует Scrum-команда, целенаправленно управляют проектированием программного обеспечения.

Стремление к техническому совершенству

Каждый раз, когда моим дочерям, как и большинству других детей, удается нарисовать какой-нибудь особенно выдающийся шедевр, они выставляют его на самом видном месте, а именно — на нашем холодильнике. Однажды, написав на C++ особенно удачную, на мой взгляд, программу, я также прикрепил ее листинг к дверце нашего холодильника — в знак особой гордости этим плодом своего труда. Вообще говоря, было бы просто замечательно, если бы мы всякий раз могли настолько гордиться качеством своей работы, что имели бы все основания прикрепить ее к дверце холодильника рядом с выдающимися художественными произведениями наших детей! Хотя вы, возможно, и не заходите так далеко, чтобы прикреплять листинги своих лучших программ и тестов или схемы базы данных к дверце своего холодильника, выполнение работы на самом высоком уровне является целью, общей для многих Scrum-команд.

В этом разделе мы рассмотрим типичные технические приемы, используемые Scrum-командами для повышения качества работы: разработку программного обеспечения на основе тестов, рефакторинг, коллективное владение, непрерывную интеграцию и парное программирование. Несмотря на то что я назвал эти технические приемы типичными, следует отметить, что говорить об их повсеместном применении не приходится. Эти приемы действительно пользуются всеобщим признанием и ведут к повышению качества продукта, но поскольку их практическое внедрение связано с определенными трудностями, они используются не так часто, как хотелось бы. Между тем каждая из Scrum-команд должна хотя бы опробовать их на практике. Поскольку рассмотрению каждого из указанных приемов посвящено множество великолепных книг и статей, я лишь вкратце опишу каждый из них, весь же остальной материал этой главы будет посвящен способам внедрения каждого из них в вашей организации и опровержению типичных возражений против этих приемов.

См. также Конкретные рекомендации относительно дополнительных материалов для чтения, которые помогут вам узнать больше о самих этих технических приемах, приводятся в разделе “Дополнительная литература” в конце данной главы.

Проектирование на основе тестирования

Если вы поинтересуетесь, как работают программисты в составе команды, придерживающейся традиционного метода разработки, то окажется, что, как правило, они выбирают для написания определенную часть программы, кодируют ее, пытаются скомпилировать, исправляют все ошибки компиляции, прогоняют написанный код через отладчик и повторяют цикл. Этот процесс в графическом виде представлен на рис. 9.1. Он весьма отличается от разработки программного обеспечения на основе тестов, который также представлен на этом рисунке. Программист, участвующий в разработке программного обеспечения на основе тестов, работает очень короткими циклами определения и автоматизации тестов работоспособности, написания

“сырого” кода, способного пройти такой тест, и последующей “очистки” этого кода, прежде чем приступить к очередному циклу.

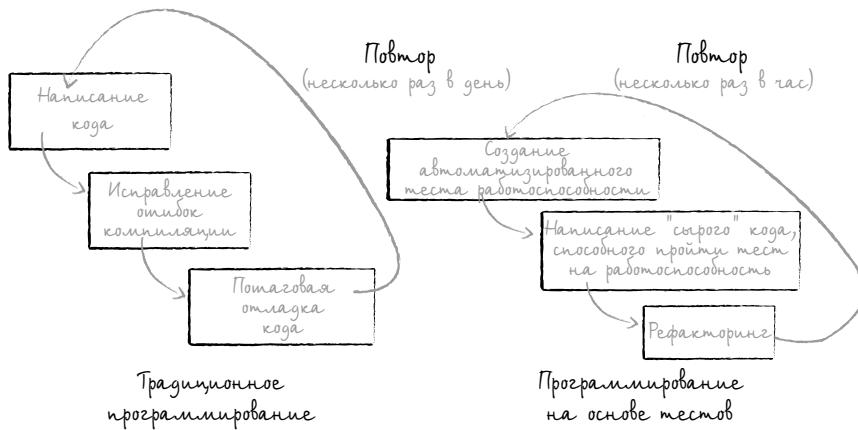


Рис. 9.1. Микроциклическая природа традиционного метода разработки и разработки программного обеспечения на основе тестов

См. также Такого рода “очистка” кода называется рефакторингом. Рефакторинг подробно обсуждается в следующем разделе данной главы.

Я прихожу к выводу, что разработка программного обеспечения на основе тестов (Test-Driven Development — TDD) поистине бесцenna. Один из самых убедительных доводов в пользу TDD заключается в гарантии, что в готовом продукте не будет непротестированного кода. Если в качестве реакции на тест работоспособности должен быть написан весь код, то даже если мы не сделаем ничего другого, с помощью TDD мы по крайней мере добьемся полного охвата кода. Можно решить, что такого же результата можно достичь и с помощью метода тестирования непосредственно после написания кода. Однако я пришел к выводу, что когда программисты обязуются составлять свои тесты “непосредственно после” того, как завершат реализацию соответствующей функциональной возможности, они зачастую просто забывают о необходимости тестирования. Необходимость как можно скорее приступить к программированию очередной функциональной возможности может перевесить все остальные доводы разума. Вот почему программисты очень часто пишут тесты лишь для части новых функциональных возможностей или откладывают тестирование “на потом” (а потом оказывается, что это “потом” не наступает никогда).

Имеет смысл рассматривать TDD не только как метод программирования, но и как метод проектирования. В конце концов, тесты, которые пишет программист, а также последовательность, в которой они пишутся, направляют проектирование и разработку той или иной функциональной возможности. Программист не составляет список из 50 небольших тестов и не выбирает затем произвольно, какой из них реализовать первым. Напротив, выбор каждого теста, а также его места в последовательности тестов осуществляется таким образом, чтобы все вопросы, касающиеся соответствующей функциональной возможности, разрешались как можно раньше.

В этом смысле выбор и реализация тестов действительно направляют процесс разработки. В результате дизайн программного обеспечения, по крайней мере частично, вытекает из потребностей системы.

См. также Идея разработки программного обеспечения на основе тестов привела на следующем уровне масштаба к идеи разработки на основе приемочных тестов. Эта тема освещается в главе 16, “Качество”.

Время от времени возникают дискуссии по поводу того, приводит ли использование TDD к появлению более надежных (или лучших в каких-то иных отношениях) программ.¹ Однако не вызывает сомнения тот факт, что TDD как метод, который помогает программистам продумывать свои проектные решения, является весьма полезным. Например, проектное решение, которое трудно протестировать, может указывать на плохо структурированный код. Я рекомендую пользоваться TDD из-за преимуществ, связанных с тестированием; любые потенциальные улучшения проектного решения, обеспечиваемые TDD, оказываются дополнительной премией за применение этого метода.

ВОЗРАЖЕНИЕ

“Я работаю над сложной системой. Мне нужно сначала выполнить определенную работу архитектурного характера”.

Да, если вы работаете над большой и сложной системой, то это, вероятно, так. Нечего и говорить о том, что TDD как метод, применяемый на микроуровне, невозможно эффективно сочетать с небольшим объемом предварительной архитектурной проработки. Ниже в этой главе я познакомлю читателей с идеей, суть которой в том, что придерживаться гибкой методологии разработки означает уметь находить правильный баланс между предвидением и адаптацией. Вопрос лишь в том, какой именно объем предварительной архитектурной проработки необходим для достижения указанного баланса (если, конечно, такая проработка вообще требуется).

ВОЗРАЖЕНИЕ

“Предварительное составление теста связано с большими затратами времени. У меня не так много времени, чтобы транжирить его подобным образом”.

Факты указывают на то, что использование TDD влечет за собой примерно 15-процентное увеличение затрат времени на разработку программного обеспечения (George and Williams, 2003). С другой стороны, те же факты указывают, что использование TDD ведет к сокращению числа ошибок. Результаты двух исследований, проведенных в Microsoft, показывают, что при использовании TDD количество обнаруженных ошибок сократилось на 24 и 38% (Sanchez, Williams, and Maximilien, 2007, 6). Итак, поначалу использование TDD

¹ См., например, блог Эбби Фichtнера (Abby Fichtner) по адресу <http://haxrchick.blogspot.com/2008/07/tdd-smackdown.html>, где содержится ссылка на видеозапись дискуссии между Джимом Коплином (Jim Coplien) и Робертом Мартином (Robert Martin).

действительно влечет за собой увеличение затрат времени на разработку программного обеспечения, но эти дополнительные затраты времени команда компенсирует в форме сокращения времени на исправление ошибок и сопровождение системы.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Потратьте на следующей неделе по меньшей мере один полный рабочий день на разработку программного обеспечения на основе тестов. Если раньше вам не приходилось пользоваться TDD, вам, вероятно, лучше работать в паре с другим программистом, который будет вам помогать. Даже если у вашего партнера также отсутствует опыт использования TDD, вам будет легче учиться вместе.
- ❑ Научиться писать тесты работоспособности до написания кода, который должен тестируться, не так-то легко. Это принципиально иной способ работы по сравнению с традиционным способом разработки программного обеспечения. Чтобы лучше понять этот новый способ работы, можно попробовать такой метод программирования, как групповое программирование (*gang programming*). Соберите в конференц-зале группу из четырех-восьми программистов, предоставив в их распоряжение ноутбук и проектор. Кто-то из этих программистов должен приступить к кодированию, в то время как все остальные будут смотреть на этот исходный код, спроектированный на экран. Выберите один тест на работоспособность, который вы можете написать, и попросите этого программиста написать код, который пройдет данный тест. Примерно через 15 минут передайте ноутбук другому программисту. Продолжайте цикл написания кода и передачи ноутбука до тех пор, пока поставленная задача не будет выполнена.
- ❑ Если испытание TDD на вашем приложении в целом поначалу покажется слишком трудным делом, выберите какой-нибудь вспомогательный проект, на котором вы сможете потренироваться. Как насчет программы преобразования данных, до которой пока еще ни у кого не дошли руки? А как насчет небольшой утилиты, которую системные администраторы просили вас написать еще в прошлом месяце?

Рефакторинг

Рассмотрим классическое описание того, что обычно происходит, когда спустя некоторое время выполняется модификация любой программной системы, предложенное Фредом Бруксом (Fred Brooks) в книге *The Mythical Man Month* (“Мифический человеко-месяц”).

Любые модификации, как правило, ведут к разрушению структуры, повышению энтропии и хаоса в соответствующей системе. Все меньше и меньше усилий тратится на исправление недочетов в исходном проекте; все больше и больше усилий тратится на решение проблем, возникших в системе в результате предыдущих модификаций. Со временем система становится все менее упорядоченной. Раньше или позже эти модификации утрачивают какой-либо смысл, поскольку каждый шаг вперед сопровождается шагом назад. Оставаясь, в принципе, пригодной к использованию, такая система перестает быть основой для прогресса, для поступательного движения вперед (1995).

К счастью, с тех пор, как в 1975 году Брукс написал эту книгу, удалось найти такие способы модификации систем, которые не приводят к дальнейшему ухудшению после каждой очередной модификации. Способность осуществлять модификации систем без их ухудшения очень важна для Scrum, поскольку Scrum-команды создают свои продукты инкрементным способом. Вот что говорит по этому поводу Рон Джейфрис (Ron Jeffries): “При использовании гибкой методологии разработки мы идем от простого к сложному, постепенно повышая сложность проекта. Это достигается при помощи рефакторинга”.

Рефакторинг означает изменение структуры, но не поведения кода. Позвольте привести вам пример. Допустим, программист располагает двумя методами, каждый из которых содержит три идентичных оператора. Эти три общих оператора можно извлечь из обоих методов и вставить в один новый метод, который вызывается из обоих старых мест. Такой рефакторинг (формально известный как *метод извлечения*) несколько улучшает удобочитаемость и сопровождение данной программы, поскольку теперь становится очевидным, что речь идет о повторно используемом коде, причем этот дублирующийся код перенесен в единственное место. Структура кода изменилась, а поведение — нет.

Рефакторинг не является единственным рецептом успеха TDD, но помогает избежать деградации кода. Деградация кода типична для продукта, выпускающегося долгое время, и часто после нескольких лет выпуска новых версий необходимо полностью переписывать код. Путем постоянного рефакторинга и исправления мелких дефектов до того, как они перерастут в крупные проблемы, можно избежать деградации приложений. Роберт Мартин (Robert C. Martin) называет это “правилом бойскаута”.

У американских бойскаутов есть простое правило, которое мы можем применить к своей профессии: покидая место привала, нужно оставить его более чистым, чем оно было до вашего появления. Если после наших манипуляций код окажется несколько “чище”, чем до них, то код просто не сможет деградировать (2008, 14).

ВОЗРАЖЕНИЕ

“Если бы код был написан правильно с самого начала, не пришлось бы прибегать к рефакторингу”.

С таким же успехом можно было бы сказать: “Если бы Toyota производила более качественные автомобили, то можно было бы не заботиться о замене масла, покупке новых протекторов, да и вообще о каком-либо техобслуживании”. Приложения требуют сопровождения. Рефакторинг означает “сопровождение понемножку”; в этом случае сопровождение обходится значительно дешевле. Большинство менеджеров и владельцев продукта, с которыми мне приходилось общаться, запрещали своим командам заниматься рефакторингом только потому, что в прошлом их команды злоупотребляли этой возможностью. Типичным примером является команда, которая резервирует для рефакторинга последние три дня каждого десятидневного спринта. Еще одним примером является команда, которая говорит владельцу продукта: “Нет, в этом спринте мы не можем реализовать эту важную функциональную возможность, поскольку нам нужно выполнить рефакторинг того, что мы написали в прошлом спринте”. Если команде требуется три дня в каждом спринте для выполнения рефакторинга, то это является признаком других проблем. Если же команда запланировала рефакторинги, которые

настолько велики, что ей приходится отказываться от реализации функциональных возможностей, запланированных владельцем продукта, то такой рефакторинг следовало бы включить в журнал запросов на выполнение работ.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Занесите в журнал рефакторинга информацию обо всем коде, который вы хотите очистить от дефектов. Если ваша команда сосредоточена в одном месте, просто выпишите всю необходимую информацию на большом плакате и повесьте его на видном месте. В противном случае воспользуйтесь каким-либо электронным эквивалентом такого плаката. Желательно, чтобы этот перечень носил как можно более неформальный характер. Цель заключается в том, чтобы исправить все дефекты, а затем удалить этот перечень. Увековечивание такого списка рефакторинга в некоторой специализированной базе данных посредством какого-либо специализированного веб-клиента, RSS-технологии или иным способом приведет к тому, что этот список сохранится у вас навсегда...
- Изучите рефакторинги, которые можно выполнять в вашей интегрированной среде разработки автоматически.
- При выявлении очередной возможности для рефакторинга предложите членам команды зафиксировать ее в учетной карточке. Разместите такие карточки на небольшом отведенном для этого участке стены в комнате, где работает ваша команда. По мере заполнения этого участка члены команды будут испытывать все большую потребность выполнить один или несколько рефакторингов.
- В конце вашего очередного двухчасового сеанса программирования потратьте 20–30 минут на устранение дефектов, замеченных вами, когда вы просматривали уже имеющийся код.
- В ходе очередной ретроспективы проведите обсуждение рефакторинга вместе с другими членами команды, включая владельца продукта. Каков порог, когда рефакторинг переходит из разряда личного решения в разряд коллективного решения всей команды? Разумеется, я могу самостоятельно переименовать какую-нибудь неудачно названную мною переменную — решение команды для этого не требуется. Но что если разработчикам необходимо выполнить модификацию кода, которая может занять два дня? Могут ли они просто выполнить ее или для этого требуется санкция со стороны владельца продукта?

Коллективное владение

Под коллективным владением мы подразумеваем чувство личной ответственности всех разработчиков за все артефакты процесса разработки и особенно за код и автоматизированные тесты. Из-за высоких темпов выполнения Scrum-проекта команде нужно избегать заявлений наподобие “Это код Теда. Мы не будем его трогать”. Коллективное владение побуждает каждого из членов команды ощущать личную ответственность за все части программы, а это значит, что любой программист может работать над любым модулем данной программы. Таким образом, внося изменения в тот или иной модуль, программист разделяет ответственность за качество этого модуля с человеком, который писал изначальный код этого модуля.

Коллективное владение не должно порождать коллективной безответственности: программисты по-прежнему специализируются на определенных областях, в которых они предпочитают работать, но при этом каждый член команды должен добиваться следующего.

- Чтобы никто из разработчиков не становился настолько узким специалистом, что мог бы работать только в своей узкоспециализированной области.
- Чтобы ни одна из специализированных областей не становилась настолько сложной, что в ней мог бы работать только один разработчик.

Естественное преимущество выработки у членов команды чувства коллективного владения заключается в том, что это чувство побуждает разработчиков изучать все новые и новые части системы. При этом они, как правило, осваивают и новые способы выполнения своей работы. Плодотворные идеи, использующиеся в одной части приложения, быстрее находят применение в других областях в результате того, что программисты, переключаясь с одной части приложения на другую, исполняют роль переносчиков идей.

ВОЗРАЖЕНИЕ

“Это мой код, и только я буду им заниматься. Я не хочу выискивать ошибки в чужом коде”.

Я не обвиняю вас, однако не забывайте, что в это же время другие программисты находят ошибки в вашем коде. Вообще говоря, мой личный опыт свидетельствует о том, что команда, практикующая коллективное владение, пишет более “чистый” код (а следовательно, код, содержащий меньшее количество ошибок). В конце концов, никто из нас не желает ударить в грязь лицом перед своими товарищами по работе. Если же какой-то код является “моим личным” и никто другой не будет “совать в него свой нос”, то при его написании я могу позволить себе некоторую неряшливость. Однако мне придется относиться к себе гораздо строже, если я буду знать, что мой код может оказаться в руках любого из моих коллег. Чтобы убедиться в правильности такого хода мыслей, вспомните, что в ожидании гостей вы вынуждены делать в доме хотя бы поверхность уборку. Напротив, знание того, что в ближайшем будущем в вашем доме не появится никто из посторонних, расхолаживает вас и позволяет перенести уборку на отдаленный срок.

“Я не хочу, чтобы посторонние люди совали нос в мой код и рассуждали об уровне моей квалификации, опыта и т.п.”

Вполне естественные опасения. Наилучший способ избавиться от них — писать более качественный код. Если вы хорошо кодируете, оценки посторонних людей будут положительны. Если же вы не уверены в своей способности писать высококачественный код, объединяйтесь как можно чаще в пары с другими программистами. Это наверняка поможет вам улучшить качество своего кода.

“Процесс разработки ускоряется, если каждый разработчик отвечает за свою часть системы”.

Это полностью зависит от временных рамок, в которых осуществляется измерение. Если нам с вами предстоит создать в течение следующих двух недель некую систему для временного использования, то наверняка будет быстрее, если каждый из нас

будет отвечать только за свою часть данного приложения. Если же мы с вами ведем разработку в рамках гораздо более масштабной системы, эксплуатация которой рассчитана на достаточно длительный срок, то повышение квалификации, перекрестное обучение и прочие преимущества коллективного владения убедительно свидетельствуют в его пользу.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Воспользуйтесь тем, что владелец какого-либо критического фрагмента системы находится в отпуске и у вас практически нет возможности с ним посоветоваться. Примите на пару спринтов волевое решение, согласно которому выполнение задач, связанных с некоторой узкоспециализированной областью, "очевидному кандидату" брать на себя не разрешается. Если возникает необходимость в консультации специалиста в этой области, то с ним можно связаться только по телефону (даже если этот человек сидит в соседней комнате, с ним можно общаться только по телефону).
- Когда в следующий раз вам придется работать с особенно трудным для вас кодом, исправляйте его, даже если этот код был написан кем-то другим. Если вам кажется, что это выходит за рамки ваших возможностей, попросите программиста, написавшего этот код, поработать вместе с вами, чтобы сделать код более простым в применении.

Непрерывная интеграция

Создание официального ежедневного варианта того или иного программного продукта считается оптимальной отраслевой практикой по крайней мере с начала 1990-х годов. Но если создание ежедневных вариантов программного продукта считается хорошей идеей, то уж непрерывное создание программного продукта следует признать идеей просто замечательной. Под непрерывной интеграцией подразумевается как можно более оперативное включение в приложение нового или измененного кода и последующее тестирование этого приложения с целью проверки его работоспособности. Вместо того чтобы проверять код каждые несколько дней или даже каждые несколько недель, каждый программист в составе Scrum-команды, осуществляющей непрерывную интеграцию, должен проверять код по нескольку раз в день и выполнять определенный комплекс регрессионных тестов по отношению ко всему приложению.

Непрерывная интеграция обычно осуществляется с помощью некоего инструмента или сценария, который фиксирует момент регистрации кода в системе контроля версий. Cruise Control стал первым продуктом автоматизации непрерывной интеграции, который приобрел высокую популярность. Он мог построить продукт, выполнить над ним столько тестов, сколько требуется, а затем автоматически отправить уведомление разработчику, который нарушил работоспособность программы (или всей команды). Cruise Control мог также отправлять результаты на различные дополнительные устройства обратной связи.

Некоторые команды предпочитают работать вручную. В этом случае разработчики самостоятельно выполняют построение и тестирование для каждой регистрации.

Я выступаю категорически против такого подхода. Несмотря на то что он может оказаться успешным в случае непрерывной интеграции, мой собственный опыт свидетельствует о том, что разработчики время от времени пропускают циклы построения и тестирования. Время от времени разработчику приходит в голову мысль: “Я изменил только две строки кода, и на моей машине это сработало”. Также соблазнительно пропустить цикл построения и тестирования при регистрации кода в самом конце рабочего дня: “Ух ты, уже почти шесть часов вечера! Пора собираться домой”. Разработчик рассуждает примерно так: “Я уверен, что это работает, а ожидать еще 15 минут, пока завершатся тесты, что-то не хочется...” Учитывая простоту настройки инструмента непрерывной интеграции, это почти всегда оказывается первым, чему я учю свои команды.

Для большинства разработчиков первое знакомство с автоматизированной непрерывной интеграцией является настоящим откровением. Я тоже не был исключением в этом смысле. В свое время я привык к преимуществам создания ежедневных вариантов и мне как-то даже не приходило в голову, что “несколько раз в день” может быть лучше, чем “один раз в день”. Поработав всего лишь день в среде непрерывной интеграции, я почувствовал, что в этом что-то есть. Мы не только можем устраниТЬ весь риск возникновения проблем, связанных с крупномасштабной интеграцией в конце проекта, но и получаем еще одно важное преимущество: вся команда разработчиков принимает практически в режиме реального времени информацию обратной связи о состоянии продукта.

ВОЗРАЖЕНИЕ

“Поддержка сервера сборки и всех этих тестов отнимает время, необходимое для другой, более важной работы”.

Scrum-командам требуется среда автоматизированного тестирования независимо от того, предусмотрена ли в ней непрерывная интеграция. Таким образом, единственной дополнительной нагрузкой является настройка и поддержка сервера сборки. Для большинства приложений эта инвестиция окупится в течение первого месяца за счет времени, сэкономленного на решении задач интеграции.

“Наша система чересчур сложна: чтобы выполнить полный ее тест, требуется не один час. Из этого следует, что мы просто не в состоянии заниматься сборкой непрерывно”.

В наше время довольно часто встречаются Scrum-команды с тестовым комплектом, выполнение которого длится несколько часов. Разумное решение в этом случае заключается в расчленении тестового комплекта, а не в отказе от самой идеи непрерывной интеграции. Именно так поступили Стефан Марш (Stephen Marsh) и Стелиос Пантазопулос (Stelios Pantazopoulos), которые работали над проектом Трансканадского трубопровода.

Уже через несколько месяцев после начала реализации проекта стало очевидно, что выполнить весь регрессионный тест за время, не превышающее пятнадцати минут, невозможно. В результате этот регрессионный тест пришлось разделить на две части: испытание герметичности трубопровода с помощью дыма и полный тест. Первый из этих тестов выполнялся после каждой регистрации и включал

все тестовые сценарии из текущей поставки [спрингта], а также определенное подмножество сценариев из прошлых поставок. Второй из этих тестов выполнялся ежечасно и включал все тестовые сценарии всех поставок. В подавляющем большинстве случаев было вполне достаточно первого теста. Второй тест завершался неудачно лишь в очень редких случаях (2008, 241).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Создание официальной ежедневной сборки является обязательным для любой Scrum-команды. Обеспечение как минимум такой частоты создания сборок должно быть одной из ваших главных задач (если, конечно, вы еще не решили эту задачу). Время и усилия, затраченные на выполнение этой задачи, окупятся максимум через месяц, поэтому у вашего нежелания решить ее не может быть оправданий. Запланируйте решение этой задачи на свой следующий спринт.
- ❑ Если задача получения ежедневной сборки вами уже решена, сделайте следующий шаг и приступайте к непрерывному формированию сборок.
- ❑ Если вы уже занимаетесь непрерывной сборкой, но не выполняете в ее рамках тестирования, то его необходимо предусмотреть. В главе 16, “Качество”, рассказывается о пирамиде автоматизации тестирования. Включение первого теста каждого типа из этой пирамиды связано с определенными трудностями. Но после того как первый тест будет интегрирован, остальное станет гораздо более простым делом.

Парное программирование

Парное программирование — это совместная работа двух разработчиков над написанием кода. Парное программирование возникло в результате следующего простого соображения: если полезны взаимные проверки кода, проводящиеся время от времени, то постоянные взаимные проверки кода должны быть еще более полезными. Многие из описанных выше методов на практике реализуются гораздо проще при использовании парного программирования. Осваивать разработку программного обеспечения на основе составления тестов гораздо легче, работая вместе. Чувство коллективного владения возникает у людей, когда они пишут код совместно (*парами*). Когда рядом с вами сидит другой разработчик, это дисциплинирует вас и заставляет внимательнее и серьезнее относиться к процессу кодирования; в результате код содержит меньше ошибок.

У парного программирования есть ряд несомненных достоинств. Именно поэтому я его изобрел. Ну, ладно, изобрел его, по правде говоря, не я, но мне хотелось бы думать, что это сделал я. Случилось так, что в силу обстоятельств мне пришлось прибегнуть к парному программированию. (Вообще говоря, изобретения очень часто являются результатом возникновения тех или иных обстоятельств.) В 1986 году я поступил на работу в лос-анджелесское представительство компании Andersen Consulting. В первый же день я составил на себя квалификационную характеристику. Я написал, в частности, что считаю себя “опытным программистом”, кодирующим на языке программирования C, хотя, справедливости ради нужно сказать, что в то время я лишь *начинал* писать программы на этом языке. Но, рассуждал я, каждый вечер после работы я занимаюсь изучением этого языка, и к тому времени, когда кто-нибудь

в Andersen Consulting удосужится прочитать мою характеристику, я наверняка буду неплохо знать этот язык. К сожалению, мою характеристику прочитали уже на следующий день после того, как я ее написал. А еще через день я летел из Лос-Анджелеса в нью-йоркское представительство Andersen Consulting, чтобы подключиться к выполнению проекта, которому срочно требовались программисты, владеющие языком программирования С.

Прибыв в Нью-Йорк, я встретился с другим программистом, которого также откомандировали в нью-йоркское представительство Andersen Consulting, поскольку он знал С. Я понимал, что мне не удастся его обмануть и поэтому решил покаяться перед ним в “преувеличениях”, допущенных мною при составлении квалификационной характеристики. “Честно говоря, — вздохнул он в ответ, — я тоже приврал, когда составлял собственную квалификационную характеристику”. После такого взаимного признания мы решили, что будем работать вместе. Вот так и возникла идея парного программирования (хотя в то время мы, конечно, не знали такого термина). Мы решили, что из двух начинающих программистов вполне может получиться один опытный программист на С. К тому же, полагали мы, если мы будем всегда работать вместе, то начальство не будет знать, кого из нас двоих следует уволить.

Идея оказалась блестящей! Мы работали вместе большую часть последующих восьми лет в трех разных компаниях, стараясь как можно больше работать в паре, особенно когда речь шла о написании трудных программ. Мы создали ряд замечательных и невероятно сложных продуктов — и каждый раз с очень небольшим количеством ошибок. Мы чувствовали, что такой способ работы резко повышает нашу производительность.

Мой первый положительный опыт применения парного программирования навсегда привил мне любовь к этому методу. Я понял, что это весьма эффективный способ написания кода. С другой стороны, многие из программистов (и я в том числе) выбрали эту профессию только потому, что она дает возможность уединиться в своем отсеке, включить плеер Sony Walkman (кое-кто, наверное, даже не помнит, что это такое), нацепить наушники и отключиться на целый день от внешнего мира. Даже теперь у меня бывают дни, когда я с удовольствием провожу время подобным образом: уединяюсь в своем отсеке, включаю MP3-плеер и начинаю кодировать в таком темпе, что пальцы едва успевают перескакивать с одной клавиши на другую. Поскольку окружающие видят, что такой способ работы доставляет мне истинное наслаждение, мне бывает не очень-то легко убедить свою команду в полезности и даже необходимости парного программирования.

К счастью, большинство команд пришло к выводу, что многие преимущества парного программирования могут быть реализованы даже в том случае, если этот метод применяется не ежедневно и ежечасно. Поэтому, пребывая в роли наставника команд, я предлагаю им применять метод парного программирования хотя бы время от времени (например, при создании самых “рискованных” частей приложения). Я предлагаю командам самим выбирать “критерий достаточности” применения парного программирования, подчеркивая при этом, что такой “критерий достаточности” должен быть все-таки несколько выше, чем 0%, и в то же время соглашаясь с тем, что могут быть причины, в силу которых он должен быть ниже 100%.

У парного программирования есть много достоинств. Это касается даже команд, которые пользуются методом парного программирования не все свое рабочее время.

Несмотря на то что результаты большинства исследований указывают на небольшое увеличение общего количества человеко-часов, затрачиваемых при использовании парного программирования, это увеличение компенсируется сокращением суммарной продолжительности самого процесса программирования. Иными словами, несмотря на то что использование парного программирования приводит к увеличению общего количества человеко-часов, само по себе время программирования соответствующей задачи сокращается (Dybå et. al, 2007). Несмотря на то что проекты всегда находятся под финансовым прессингом, решающим фактором является все же не суммарное количество человеко-часов, затраченных на реализацию проекта, а время от появления идеи нового продукта до вывода этого продукта на рынок. Парное программирование, как свидетельствуют результаты исследований, ведет также к повышению качества. В своем обзоре ряда исследований Дыба (Dybå) и его коллеги пришли к выводу, что во всех случаях парное программирование приводило к повышению качества. Кроме того, оно способствует передаче знаний и является идеальным способом введения в курс дела разработчиков, подключающихся к выполнению нового для них проекта. Парное программирование является также эффективным способом работы на “неизведанных территориях” или решения трудных проблем в известных частях системы.

ВОЗРАЖЕНИЕ

“Парное программирование стоит дороже. Я не хочу платить двум программистам за выполнение работы, с которой вполне может справиться один человек”.

Парное программирование обойдется дороже в краткосрочной перспективе. Однако эти дополнительные начальные издержки наверняка окупятся за счет сокращения сроков разработки и повышения качества, что в конечном итоге приведет к снижению эксплуатационных затрат. Вместо того чтобы ориентироваться на результаты отраслевых исследований, проверьте это на собственном опыте. Предложите кодировать самые трудные модули парам программистов и посмотрите, не сократилось ли количество ошибок и не облегчилось ли впоследствии сопровождение этих модулей (возможно, в сравнении с аналогичными модулями из других программ, созданных без использования парного программирования).

“Мы работаем в условиях жесткого дефицита времени. Мы не можем позволить себе работу двух программистов над одной задачей”.

Совершенно ошибочное мнение! Как раз наоборот: если вы работаете в условиях жесткого дефицита времени, вам следует прибегнуть к парному программированию. Я уже упоминал, что парное программирование позволяет сократить время реализации проекта (увеличивая, однако, суммарное количество человеко-часов). Более того, есть факты (Williams, Shukla, and Anton, 2004), указывающие на то, что парное программирование является эффективным способом противодействия так называемому закону Брукса (“подключение дополнительных исполнителей к программному проекту, не укладывающемуся в сроки, ведет к еще большему затягиванию этого проекта”). Иными словами, если вас поджимают сроки или вам хотелось бы подключить дополнительных исполнителей к проекту, не укладывающемуся в сроки, это тот самый случай, когда следует прибегнуть к парному программированию.

“Когда приходится решать какую-либо сложную проблему, мне нужно побить какое-то время одному и в спокойной обстановке обдумать сложившуюся ситуацию”.

Поговорите со своим партнером и договоритесь с ним расстаться на час-другой, чтобы в спокойной обстановке обдумать сложившуюся ситуацию. Когда вы возобновите работу в паре, прежде всего обменяйтесь с ним мыслями, которые пришли вам в голову, когда вы самостоятельно размышляли над возникшей проблемой.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- На своем следующем совещании, посвященном планированию спринта, предусмотрите решение какой-либо задачи посредством парного программирования — пусть даже в относительно небольшом объеме. Это обязательство желательно было бы зафиксировать в журнале спринта (например, “Майк и Боб занимаются парным программированием в течение двух часов”, “после обеда и до конца рабочего дня Майк и Сэм работают в паре”, и т.п.). Это хороший способ избавить людей от ощущения необязательности, появляющегося у некоторых из них в связи с парным программированием. Очень легко отградить людей от парного программирования, предложив им заняться парным программированием “в ближайшем будущем”, когда представится подходящий случай. Включение соответствующего обязательства в журнал спринта действует, как назойливое напоминание, что существенно повышает вероятность того, что те, к кому это относится, не смогут легко его проигнорировать.

Проектирование: целенаправленное и спонтанное

Scrum-проекты не предусматривают обязательной фазы предварительного анализа или фазы проектирования: вся работа выполняется повторяющимися циклами спринтов. Это, однако, не означает, что в таком случае проектирование не выполняется целенаправленно. Процесс целенаправленного проектирования — это управление проектированием посредством принятия целенаправленных, осознанных решений. Особенность Scrum-проекта заключается не в том, что целенаправленное проектирование отвергается, а в том, что оно выполняется (подобно всему остальному в Scrum-проекте) инкрементно, методом последовательного наращивания. Scrum-команды исходят из того, что несмотря на несомненную привлекательность заблаговременного принятия всех проектных решений, такой вариант невозможен при выполнении Scrum-проектов. Это означает, что при выполнении Scrum-проектов проектирование носит как целенаправленный, так и спонтанный характер.

Для организации, осваивающей гибкую методологию разработки, очень важно найти оптимальный баланс между предвидением и приспособлением (Highsmith, 2002). Этот баланс — а также действия и артефакты, которые влияют на него — представлены на рис. 9.2. При выполнении предварительного (заблаговременного) проектирования мы пытаемся *предвидеть* потребности пользователей. Поскольку мы не в состоянии идеально предсказать эти потребности, мы обязательно будем допускать

какие-то ошибки, какую-то работу придется переделывать. Когда же мы полностью отказываемся от анализа и проектирования и переходим сразу же к кодированию и тестированию без каких-либо попыток представить, как будет выглядеть наш конечный продукт, мы просто пытаемся *приспособиться* к текущим потребностям пользователей. Вообще говоря, все проекты располагаются где-то между этими двумя полюсами — между предвидением в чистом виде и приспособлением в чистом виде — исходя из их собственных уникальных характеристик: ни одно приложение не может быть построено только на предвидении или только на приспособлении. Критичное для жизни человека приложение, обеспечивающее безопасность медицинского вмешательства, может располагаться достаточно близко к полюсу предвидения. Начинающая компания из трех человек, создающая веб-сайт с информацией о гонках на байдарках, может позиционироваться достаточно близко к полюсу приспособления.

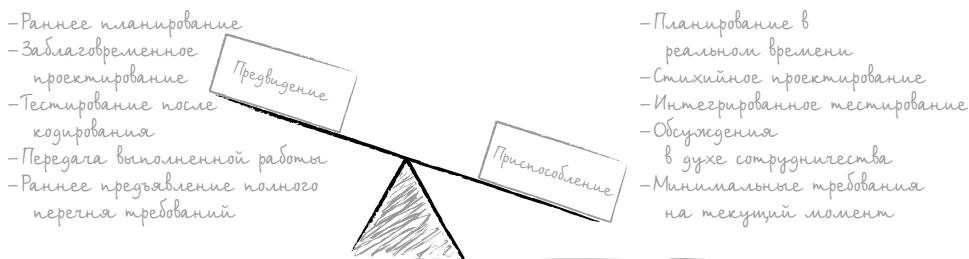


Рис. 9.2. Достижение баланса между предвидением и приспособлением предполагает учет влияния действий и артефактов с обеих сторон

Лектор и автор многочисленных работ “Ду-Вайл” Джонс (“Do-While” Jones) еще в 1990 году предсказывал склонность гибкой методологии разработки к простоте.

Я не имею ничего против планирования будущего. Всегда следует заранее подумать о будущем расширении возможностей. Но когда весь процесс проектирования увязает в попытках удовлетворить будущие требования, которые могут никогда не материализоваться, нужно остановиться и посмотреть, нет ли более простого способа решения проблем, непосредственно стоящих перед нами.²

Scrum-команды избегают этого “увязания”, понимая, что не все будущие потребности стоят того, чтобы беспокоиться о них уже сегодня. Многие будущие потребности можно удовлетворить, просто приспосабливаясь к ним по мере их возникновения.

Привыкайте к жизни без крупномасштабного проектирования

Когда Scrum-команды постепенно становятся приверженцами технических приемов, описанных в этой главе, они, естественно, начинают уходить от предвидения потребностей пользователей и сдвигаться в сторону приспособления к ним. Это ведет к ряду изменений в работе архитектора или проектировщика, начинающего действовать

² Статья Джонса “The Breakfast Food Cooker” (1990) остается хрестоматийным примером того, что может случиться, если разработчики программного обеспечения переусердствуют с проектированием какого-либо решения. Я настоятельно рекомендую ознакомиться с этой статьей, находящейся по адресу http://www.ridgecrest.ca.us/~do_while/toaster.htm.

в среде гибкой методологии разработки, и ему придется привыкать к этим переменам. Ниже перечислены новые реалии, обусловленные этим сдвигом.

- **Планировать становится труднее.** Оценивание, планирование и соблюдение взятых обязательств сложны сами по себе. Но эти задачи становятся еще более сложными при отсутствии заблаговременного проектирования. На создание предварительного проекта уходит много времени и сил, которые позволяют оценить продолжительность выполнения тех или иных действий и составить на основе этих оценок соответствующие планы. Положительная сторона отказа от заблаговременного проектирования заключается в том, что работа, которую нужно оценить, зачастую оказывается проще, что позволяет быстрее оценить отдельные функциональные возможности.
- **Труднее разделить работу между командами или отдельными исполнителями.** Наличие заблаговременно разработанного проекта позволяет быстрее определить, какие функциональные возможности следует создавать одновременно, а какие — последовательно. Это упрощает задачу распределения работы между командами или отдельными исполнителями.
- **Если нет заблаговременно разработанного проекта, люди испытывают определенный дискомфорт.** Несмотря на понимание того, что никакой заблаговременно разработанный проект не бывает полностью идеальным, мы чувствуем себя комфортнее, когда такое проектирование выполнено. “Разумеется, — рассуждаем мы, — по большому счету мы уже все продумали. Если и потребуются какие-то изменения, то они коснутся лишь мелочей”.
- **Неизбежна переделка работы.** При отсутствии заблаговременно выполненного проекта команда рано или поздно столкнется с необходимостью переделки тех или иных частей проекта. Этот аспект итеративной разработки (“два шага вперед, шаг назад”) может вызывать дискомфорт у специалистов, привыкших заранее выявлять все потребности и заранее принимать все проектные решения. К счастью, рефакторинг и автоматизированные тесты, созданные в процессе разработки программного обеспечения на основе тестов, могут существенно сократить объем таких переделок.

Разработка укрупненного предварительного проекта приобрела столь большую популярность из-за уверенности людей в том, что такое проектирование позволяет сэкономить деньги и время. Считалось, что стоимость создания предварительного проекта плюс стоимость внесения изменений в этот проект будут меньше, чем суммарная стоимость множества мелких изменений, необходимых в случае стихийного проектирования. В графическом виде эта ситуация представлена на рис. 9.3. Вопрос лишь в том, какая из чаш этих весов перевесит другую.

В прошлом при выполнении укрупненного предварительного проекта зачастую удавалось сэкономить деньги и время. В конце концов Барри Боэм продемонстрировал в *Software Engineering Economics* (1981), что устранение дефектов обходится тем дороже, чем позже в процессе разработки их удается выявить. Но технические приемы, используемые эффективными Scrum-командами, способны радикально изменить это “уравнение”. Если команда пользуется эффективными техническими приемами (такими, как разработка программного обеспечения на основе составления тестов, интенсивное использование автоматизированного тестирования исходного кода,

рефакторинг и парное программирование), она может оказаться в ситуации, когда дешевле приспосабливаться к потребностям пользователей путем более частой переработки соответствующего приложения, чем пытаться предвидеть эти потребности и лишь время от времени переделывать уже выполненную работу.

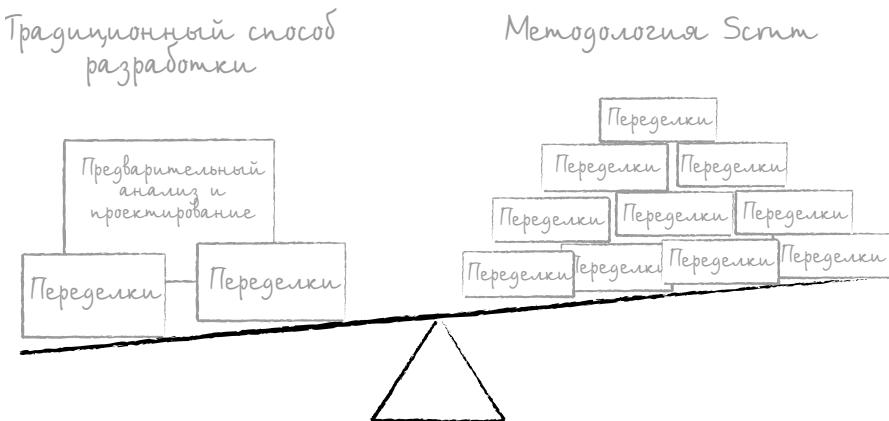


Рис. 9.3. На одной чаше весов находятся затраты на создание внушительного предварительного проекта и анализ, а также на внесение время от времени дорогостоящих изменений; на другой — стоимость частых, но незначительных изменений в Scrum-проекте

Из рис. 9.3 следует, что при использовании традиционного способа разработки существенно повышается стоимость предварительного анализа и проектирования. Эти инвестиции сокращают объем последующих изменений. Но когда требуется внести то или иное изменение, оно оказывается довольно дорогостоящим, поскольку нарушает исходное предположение о мизерной вероятности внесения изменений. В отличие от традиционного способа разработки применение методологии Scrum изначально предполагает необходимость внесения гораздо большего числа изменений, но величина каждого такого изменения оказывается значительно меньшей. Это результат предвидения того, что изменения обязательно потребуются, только неизвестно, в каких именно местах. Поэтому Scrum-команда стремится к техническому совершенству, обеспечивая высокую факторизацию кода, как можно больше упрощая конструкцию программы и применяя комплект автоматизированных тестов для как можно более раннего обнаружения регрессионных проблем. И потому, несмотря на наличие большего числа переделок уже выполненной работы, объем каждой из них сравнительно невелик.

Управление проектированием

Нападки на Scrum-проекты в связи с отсутствием укрупненного предварительного проектирования обычно начинаются с того, что технические члены команды не оказывают никакого влияния на последовательность добавления в систему функциональных возможностей. Это ошибочное предположение. В действительности едва ли не самое лучшее, что может сделать Scrum-команда для того, чтобы чаши весов, показанных на рис. 9.3, склонились в требуемую сторону, — повлиять на последовательность добавления в систему функциональных возможностей.

Однако в главе 7, “Новые роли”, было сказано, что расстановка приоритетов в журнале запросов на выполнение работ является обязанностью владельца продукта. Это верное утверждение. Однако в той же главе указывалось и на то, что опытный владелец продукта обязательно прислушается к советам членов своей команды. Стандартное правило Scrum гласит, что владелец продукта выполняет расстановку приоритетов, основываясь на некой расплывчатой концепции “экономической выгоды”. Несмотря на то что это звучит довольно убедительно, здесь чувствуется некоторое упрощение. Подлинная задача владельца продукта заключается в том, чтобы максимизировать реализацию функциональных возможностей за определенный период времени. Это может означать некоторое снижение “экономической выгоды” сейчас во имя ее повышения в будущем. Иными словами, опытный владелец продукта по-прежнему сосредоточивается на том, чтобы готовый продукт представлял собой как можно большую экономическую ценность, но при этом представляет команде возможность внесения своего вклада в технические аспекты этого продукта.

Приступая к выполнению нового проекта, команда должна предложить владельцу продукта выбрать те элементы журнала запросов на выполнение работ, которые максимизируют возможности обучения и исключают как можно больше технической неопределенности или риска. Именно это я имел в виду, когда говорил ранее о том, что проектирование при выполнении Scrum-проекта носит как целенаправленный, так и спонтанный характер. Мы говорим о спонтанном проектировании (которое фактически происходит во время каждого из спринтов), поскольку самостоятельная фаза предварительного проектирования формально отсутствует. Мы говорим о целенаправленном проектировании, поскольку элементы журнала запросов на выполнение работ выбираются с вполне определенной целью: направлять ход проектирования в то или иное русло в те или иные моменты времени.

Пример

В качестве примера того, как можно упорядочить элементы журнала запросов на выполнение работ, чтобы повлиять на архитектуру системы, рассмотрим систему документооборота, над созданием которой мне пришлось однажды работать. Эта система предназначалась для одной компании по сбору денег на благотворительные нужды. Компания производила специализированные тенниски и другую подобную продукцию, собирая таким образом деньги, которые использовались на всевозможные благотворительные нужды. Продажей этой продукции должны были заниматься дети школьного возраста. Переходя от дома к дому, они предлагали жильцам купить свой товар. Доход от продажи предполагалось разделить между этой компанией и организацией, которую представляли дети (школа, спортивная команда или какая-то другая группа). Каждый факт продажи регистрируется следующим образом: школьник заполняет соответствующую форму и отправляет ее в компанию, где она сканируется, проходит процесс оптического распознавания и преобразуется в заказ. С целью снижения транспортных издержек заказы из одной и той же организации группируются и отправляются обратно в соответствующую организацию, после чего дети вручают товар покупателям.

Разработанное нами программное обеспечение контролировало весь процесс — с момента получения документа компанией и до вручения товара покупателю. Порядок, которым дети заполняют формы, а также грамотность детей, как известно, оставляют

желать лучшего, поэтому нашей системе приходилось делать больше, чем просто сканировать формы и составлять упаковочные реестры. Были предусмотрены разные уровни проверки данных в зависимости от того, насколько точно, по нашему мнению, была прочитана каждая форма заказа. Некоторые формы направлялись клеркам, которые осуществляли визуальный контроль входных данных: на одной половине экрана отображалась отсканированная форма, а на другой — ее интерпретация системой; кроме того, было зарезервировано место для внесения поправок.

Поскольку в самые напряженные дни обрабатывались данные по тысячам теннисок, этот процесс нужно было максимально автоматизировать. Вместе с владельцем продукта, Стиви, я составил журнал запросов на выполнение работ. После этого я встретился с командой разработчиков, чтобы обсудить, какие части системы связаны с наибольшим риском; в противном случае нам было бы очень трудно решить, как разрабатывать эту систему. Мы решили, что наш первый спринт будет посвящен “прогону” высококачественного документа через всю систему. Нужно было отсканировать этот документ и пропустить его через систему оптического распознавания символов, а также сгенерировать упаковочный реестр. Мы решили “обойти” необязательные этапы, такие как выравнивание помявшихся страниц, очистка изображения от “мусора” и т.п. Главное для нас было — убедиться, что вся эта технологическая цепочка может быть выполнена от начала до конца. Возможно, ценность этой проверки была не так уж велика, но это обязательно нужно было сделать; к тому же это позволяло разработчикам протестировать архитектуру в целом. После того как эта работа была выполнена, мы получили исходную базу данных и смогли перемещать документы из одного состояния в другое, инициируя требуемые этапы документооборота.

Затем разработчики спросили у владельца продукта, могут ли они работать над той частью системы, которая отображает отсканированный документ для человека, со-поставляющего отсканированные и интерпретированные системой данные и принимающего окончательное решение относительно их достоверности. Это было выбрано в качестве второй архитектурной цели проекта в силу трех причин.

- Это был “ручной” этап, что обусловливало его отличие от ранее выполненных этапов документооборота.
- Очень важным моментом была правильная организация пользовательского интерфейса. Учитывая значительное количество документов, проходящих через эту систему, очень важно было экономить буквально каждую секунду. Мы хотели, чтобы пользователи как можно раньше оценили качество нашей системы. В таком случае у нас оставалось бы больше времени на повышение ее эргономичности.
- После добавления этой функциональной возможности пользователи могли бы сразу приступить к обработке заказов на тенниски.

Реализация проекта продолжалась в том же духе в течение нескольких месяцев и завершилась огромным успехом. Нам удалось достичь всех целей, касающихся надежности и производительности этой системы. Ключом к успеху стало то обстоятельство, что владелец продукта и технический персонал совместно выбирали оптимальную последовательность работ. Команда в плотную подошла к фазе проектирования, когда мы, собравшись в конференц-зале, определили области наибольшего риска и так называемые “темные углы” и решили, за какие из них мы возьмемся в первую очередь.

Начиная с этого момента, спринт за спринтом, пошел процесс стихийного проектирования, который, тем не менее, целенаправленно управлялся выбором тех элементов журнала запросов на выполнение работ, которые были призваны освещать “темные углы” и риски данного проекта.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Организуйте дискуссию между командой и владельцем продукта о том, в какой мере технические факторы должны влиять на приоритизацию владельцем продукта журнала запросов на выполнение работ.
- ❑ До начала очередного совещания по планированию спринга выявите пять важнейших технических неопределенностей данного проекта и риск, связанный с каждой из них. Посмотрите, есть ли в журнале запросов на выполнение работ такие элементы, приоритет которых можно было бы несколько повысить, искусственно создав учебную необходимость в устраниении выявленных вами неопределенностей.

Обязательное совершенствование технических приемов

Технические приемы, описанные в этой главе, как мне кажется, должны составлять инструментарий любой высокопроизводительной команды. Разумеется, можно утверждать, что этими приемами необязательно пользоваться все время, пока вы работаете над своим приложением. Однако все они должны быть в арсенале эффективной Scrum-команды. Непрерывная интеграция — это лишь естественное расширение ежедневной сборки, что является минимальным требованием к команде, использующей гибкую методологию разработки. Умение выполнять рефакторинг, или тип мышления, ориентированный на коллективное владение, со временем может сформироваться у любой команды. Такие приемы, как парное программирование и разработка программного обеспечения на основе тестов, позволяют писать более качественный код, что является целью любой Scrum-команды.

Используемые совместно, эти технические приемы позволяют создавать высококачественные продукты с пониженным процентом дефектов. В главе 1, “Гибкая методология разработки: нелегко, но перспективно”, рассказывалось о повышении качества и сокращении процента дефектов в продуктах как о целях, к которым должны стремиться команды, использующие гибкую методологию разработки. Эти улучшения являются результатом повышения Scrum-командами своей квалификации и целенаправленного применения ими всего арсенала технических приемов, описанных в этой главе.

В результате этих улучшений эффективные Scrum-команды получают возможность еще больше сместить баланс между предвидением и приспособлением в сторону приспособления. Минимизация, а в отдельных случаях и полный отказ от предварительного анализа и укрупненного проектирования, позволяет экономить как время, так и деньги. В статье с очень удачным заголовком (“Проектирование, предусматривающее изменения”) Дэйв Томас (Dave Thomas), основатель компании Object Technology International, которая одной из первых занялась разработкой Eclipse, показывает, как достижение этого баланса упрощает задачу внесения изменений.

Гибкая методология разработки позволяет проектировать программное обеспечение с учетом последующего внесения изменений... Ее цель заключается в том, чтобы разрабатывать программы, восприимчивые к изменениям, и даже более того — рассчитанные на их внесение. В идеальном случае программирование в рамках гибкой методологии разработки позволяет вносить изменения простым, локализованным способом, что предоставляет возможность избежать значительных работ по рефакторингу, повторному тестированию и сборкам системы или хотя бы существенно сократить объем таких работ (2005, 14).

Дополнительная литература

Ambler, Scott W., and Pramod J. Sadalage. 2006. *Refactoring databases: Evolutionary database design*. Addison-Wesley.

В первых пяти главах этой книги объясняется роль специалиста по данным в организации, использующей гибкую методологию разработки. В последующих главах излагаются хорошо продуманные способы совершенствования проекта базы данных. Каждый рефакторинг включает описание того, что заставило вносить соответствующее изменение, компромиссы, которые следует принять во внимание, прежде чем вносить данное изменение, как обновлять схему данных, как переносить данные и какие изменения следует внести в приложения, которые обращаются к этим данным.

Bain, Scott L. 2008. *Emergent design: The evolutionary nature of professional software development*. Addison-Wesley Professional.

Я все ждал, когда же кто-нибудь напишет книгу, в которой будет показано, как осуществляется эффективное проектирование без предварительного продумывания всего проекта. Увидев название этой книги, я решил, что моим надеждам наконец-то суждено сбыться. Оказалось, что это не совсем так. Тем не менее книга Бейна является превосходным описанием того, как следует разрабатывать коды при выполнении проекта с помощью гибкой методологии разработки. В книге содержатся превосходно написанные главы с описанием многих технических приемов, о которых рассказывалось в данной главе.

Beck, Kent. 2002. *Test-driven development: By example*. Addison-Wesley Professional.

Эта небольшая книжка не научит вас, конечно, всему, что нужно знать о разработке программного обеспечения на основе составления тестов. (Для этого вам нужно обратиться к книге Lasse Koskela, *Test Driven: TDD and Acceptance TDD for Java Developers*.) Ценность книги Бека в том, что в ней описан принцип работы TDD и показано, почему имеет смысл применить TDD на практике.

Duvall, Paul, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: Improving software quality and reducing risk*. Addison-Wesley Professional.

В этой книге рассказывается обо всем, что может понадобиться знать о непрерывной интеграции. В ней рассказывается, с чего следует начинать непрерывную интеграцию, как включить в нее тесты, как использовать инструменты анализа кода и даже как оценивать сами инструменты непрерывной интеграции.

Elssamadisy, Amr. 2007. *Patterns of agile practice adoption: The technical cluster.* C4Media.

В этой книге охвачены все технические приемы, рекомендованные в данной главе (а также ряд дополнительных). Она является превосходным источником информации для тех, кто хочет пользоваться единственной книгой, в которой подробно рассматриваются все технические приемы. Изобилующая полезными советами, эта книга выдержана в едином стиле: каждый технический прием описывается строго определенным образом. Я пришел к заключению, что такой стиль изложения, несмотря на кажущуюся его логичность и привлекательность, приводит к тому, что спустя какое-то время внимание читателя рассеивается.

Feathers, Michael. 2004. *Working effectively with legacy code.* Prentice Hall PTR.

Освоение новых технических приемов и постоянное стремление к техническому совершенству при реализации нового проекта создает достаточно сложные проблемы; ситуация становится еще более сложной, когда речь заходит об уже существующем приложении, которое нужно модернизировать. В превосходной книге Майкла Физерса содержится немало полезных советов на этот случай.

Fowler, Martin. 1999. *Refactoring: Improving the design of existing code.* При участии Kent Beck, John Brant, William Opdyke, и Don Roberts. Addison-Wesley Professional.

Эту книгу следовало бы назвать “Библия рефакторинга”. Современные интегрированные среды разработки способны выполнять вместо нас огромные объемы рефакторинга, но вернуться к первоисточнику и ознакомиться с каталогом рефакторингов, представленным в этой книге, — очень полезно. Одной из моих любимых глав является глава “Big Refactorings” (“Крупномасштабные рефакторинги”). Именно крупномасштабные рефакторинги являются наиболее сложными.

Koskela, Lasse. 2007. *Test driven: TDD and acceptance TDD for Java developers.* Manning.

Это самая подробная книга по разработке программного обеспечения на основе составления тестов. Она подходит как для тех, кто не знаком с TDD, так и для тех, кто уже имеет опыт ее использования. Коскела не уходит от трудных тем и приводит советы по таким часто игнорируемым темам, как TDD для многопоточных кодов и пользовательские интерфейсы. Автор применяет глобальный подход к TDD, включив в свою книгу даже около 150 страниц, посвященных разработке программного обеспечения на основе составления приемочных тестов.

См. также Разработка программного обеспечения на основе составления приемочных тестов рассматривается в главе 16, “Качество”.

Martin, Robert C. 2008. *Clean code: A handbook of agile software craftsmanship.* Prentice Hall.

На титульной странице этой книги приведено изречение “Нет разумного оправдания недобросовестному выполнению работы”. Далее в книге рассматривается совокупность методов написания “чистого” кода. В книге излагается широкий

спектр тем, начиная с самых тривиальных (присвоение осмысленных названий) и заканчивая новаторскими (разработка архитектуры на основе составления тестов и стихийное проектирование). Я рекомендовал бы прочитать эту книгу всем программистам.

Meszaros, Gerard. 2007. *xUnit test patterns: Refactoring test code*. Addison-Wesley.

Эта поистине энциклопедическая книга охватывает все, что может понадобиться знать программисту о популярном семействе инструментов тестирования xUnit. Книга начинается с изложения основ, но достаточно быстро переходит к подробному рассмотрению более сложных вопросов.

Wake, William C. 2003. *Refactoring workbook*. Addison-Wesley Professional.

Хорошо продуманное и легкодоступное введение в рефакторинг. В этой книге приводится множество примеров Java-кода, рефакторингом которого вы можете заняться. Книга представляет собой сочетание базового учебника по рефакторингу и сборника примеров, которые служат для закрепления теоретического материала. Последняя треть книги содержит четыре программы, рефакторинг которых вы можете выполнить самостоятельно.

ЧАСТЬ III

Команды

Большинство команд вовсе не являются таковыми. Они представляют собой лишь совокупности индивидуальных отношений с начальником. Каждый из членов такой “команды” борется со своими коллегами за расширение своих полномочий, за престиж и место под солнцем.

— Дуглас Макгрегор

Глава 10

Структура команды

Возможно, это и выдумка, что люди имеют много общего со своими домашними любимцами — кошками, собаками и другими, — но нужно признать, что это очень похоже на правду. То же самое можно сказать о продуктах и командах, которые их создают.

Очень часто создаваемая система имеет структуру, являющуюся зеркальным отражением структуры группы, которая создает эту систему, независимо от того, намеревалась ли группа добиться такого сходства. Следовало бы воспользоваться этим фактом и целенаправленно формировать структуру группы таким образом, чтобы добиться желаемой структуры системы. (Conway, 1968; часто на это высказывание ссылаются как на “Закон Конвея”.)

Если верно, что продукт отражает структуру команды, которая его создавала, то вопрос формирования команды исполнителей того или иного Scrum-проекта приобретает немаловажное значение. Факторами, которые влияют на принятие соответствующего решения, являются численность команды, знание членами команды соответствующей предметной области, каналы коммуникации, эскизный проект разрабатываемой системы, индивидуальные уровни опыта членов команды, применяемые технологии, новизна этих технологий, географическое размещение членов команды, влияния со стороны конкурентов и рынка, ожидания, касающиеся календарного плана проекта, и многое другое.

В этой главе мы обсудим важность двух критических факторов, которые следует учитывать, принимая решение о структуре Scrum-команды: обеспечение компактности команды и ориентация каждой команды на обеспечение функциональности, видимой пользователям. Мы также обсудим важность комплектования каждой команды нужными ей специалистами и предотвращения чрезмерной загруженности этих людей (которая может быть вызвана тем, что им придется участвовать одновременно в нескольких командах). Завершается эта глава рассмотрением девяти вопросов, на которые нужно ответить, начиная проект, в реализации которого участвует несколько команд.

Накормите команду двумя пиццами

Однажды, когда я работал над проектом по заказу одной компании, специализирующейся на биоинформатике, главный исполнительный директор этой компании попросила меня оценить продолжительность выполнения данного проекта. Речь шла о создании весьма внушительного приложения, предметная область представлялась мне достаточно сложной, а в команде работали малоизвестные мне люди. Поскольку предметная область была сложной, в состав нашей команды входило несколько человек с учеными степенями, считавшихся авторитетными специалистами в области биологии и генетики (правда, в программировании они были новичками), а также несколько очень толковых программистов, большинство из которых имели весьма поверхностные знания в области биологии и генетики. Ни один из членов этой команды не мог похвастаться глубокими познаниями одновременно и в биологии, и в программировании.

Проведя небольшое исследование и поработав с командой, я вернулся к главному исполнительному директору с интересовавшей ее оценкой — около ста человеко-лет. Иными словами, если бы мы действовали все сорок человек команды, то могли бы завершить этот проект примерно через два с половиной года. Я не думаю, что такая оценка слишком шокировала главного исполнительного директора, тем не менее два с половиной года — срок довольно значительный, поэтому она спросила у меня: “Какой способ реализации данного проекта является, по вашему мнению, самым дешевым?” Я ответил так: “Предложите Стиви — ученому-биологу, который лучше остальных разбирается в программировании и к тому же имеет склонность к этому роду деятельности — провести десять лет в солидной фирме, занимающейся разработкой программного обеспечения. При этом от него потребуется только одно: стать великолепным программистом. Затем Стиви должен будет вернуться в вашу компанию и потратить тридцать лет на написание нужного вам приложения. Все вместе займет сорок лет, но это — самый дешевый вариант”. Мой ответ должен был устроить ее: действительно, исходя из первоначальной оценки (сто человеко-лет), я предложил ей способ сократить трудозатраты более чем наполовину. Увы, компания не могла позволить себе ждать нужного результата целых сорок лет.

Как следует из этой истории, огромное преимущество команды заключается в том, что она может реализовать проект гораздо быстрее, чем один человек. Правда, это преимущество нивелируется необходимостью постоянных контактов, интенсивного общения между членами команды. Учитывая это обстоятельство, можно ли ответить на вопрос “Какая численность команды является оптимальной при выполнении Scrum-проектов?” Принято считать, что оптимальная численность команды составляет от пяти до девяти человек. В принципе, я согласен с такой оценкой. Правда, меня несколько смущает, что приходится называть точные величины. Если вы возглавляете в настоящий момент команду из десяти человек, то у вас может возникнуть непреодолимое желание отнести эту книгу в магазин, где вы купили ее, и попросить продавца возместить вам ее полную стоимость; заодно вы можете вообще отказаться от использования Scrum.

Пожалуйста, не делайте скоропалительных выводов!

Вместо того чтобы воспринимать рекомендацию “от пяти до девяти человек” слишком буквально, я посоветовал бы вам вспомнить, что говорит о своих командах

компания Amazon.com. Она говорит о своих командах как о “командах, которые можно накормить двумя пиццами” (Deutschman, 2007). К этому определению нужно, конечно, относиться с известной долей юмора, хотя, как известно, в каждой шутке есть доля шутки. Если пропитание группы людей превращается для вас в проблему, то это может свидетельствовать о том, что такая компания чересчур велика для вас.

Самая большая из Scrum-команд, с которыми мне приходилось работать, состояла из 14 человек. Члены этой команды, ее Scrum-мастер и я рассмотрели все возможные способы ее разделения на две самостоятельные команды, но в конце концов были вынуждены прийти к заключению, что будет лучше, если мы оставим все как есть. Мне пришлось также работать с командой из 25 человек. Они также настаивали на сохранении этой команды в неизменном виде. Это оказалось ошибкой: накладные расходы, вызванные необходимостью общения между столь большим числом членов команды, оказались слишком большими.

См. также Крупномасштабные Scrum-проекты реализуются с помощью “коллектива команд”. В главе 17, “Изменение масштаба Scrum”, приводится информация о крупномасштабных Scrum-проектах.

Почему достаточно двух пицц

По правде говоря, у больших команд есть определенные преимущества. Чем больше численность команды, тем более широкий круг специалистов (с разным опытом, специализацией и подходами) можно в нее включить. Чем больше численность команды, тем меньше риск провалить проект из-за потери какого-нибудь ее ключевого члена. Большие команды предоставляют своим членам больше возможностей специализироваться в использовании какой-то определенной технологии или какого-то разрабатываемого приложения.

С другой стороны, у малых команд также есть свои преимущества. Они перечислены ниже.

- **Уменьшается вероятность возникновения такого явления, как социальное нахлебничество.** Социальное нахлебничество — это склонность людей расслабляться и меньше “вкалывать”, когда они видят рядом с собой тех, кто готов взять их “на буксир”. Члены небольших команд в меньшей степени склонны к социальному нахлебничеству. Данное явление впервые было продемонстрировано психологом Максом Рингельманном (Max Ringelmann) еще в 20-х годах XX столетия, когда он измерял силу, прикладываемую командами (и отдельными их членами), которые соревновались в перетягивании каната. Команды из трех человек прикладывали усилие, составляющее 2,5 средних усилий одного человека. А команды из восьми человек прикладывали усилие, меньшее даже четырехкратного среднего усилия одного человека. Исследование Рингельманна, а также другие исследования подобного типа наглядно показывают, что величина усилий, прикладываемых отдельным человеком, связана с величиной команды обратно пропорциональной зависимостью (Stangor, 2004, 220).
- **Конструктивное взаимодействие в большей степени характерно для малых команд.** Стефан Роббинс (Stephen Robbins), автор книги *Essentials of Organizational*

Behavior, популярного учебника по организационному поведению, пришел к выводу, что у команд, численность которых превышает 10–12 человек, возникают трудности с установлением атмосферы доверия, взаимной ответственности и сплоченности. При отсутствии такой атмосферы конструктивное взаимодействие оказывается весьма проблематичным (2005).

- **Меньше времени затрачивается на координацию.** Малые команды тратят меньше времени на координацию действий отдельных ее членов. Это справедливо как в отношении совокупного времени на выполнение проекта, так и в отношении процента от общего времени проекта. Простой пример: все мы знаем, что иногда даже спланировать простое собрание многочисленной команды очень нелегко.
- **Никто не может спрятаться в тень.** В больших командах снижается степень участия отдельных членов команды в групповой деятельности и дискуссиях. Аналогично повышается дисбаланс степени участия членов команды. Эти проблемы могут помешать группе лиц выкристаллизоваться в сплоченную, высоко-производительную команду.
- **Работа в небольших командах доставляет их членам большее удовлетворение.** В небольшой команде вклад одного человека оказывается более значимым и заметным для окружающих. Возможно, это является одной из причин того (и на это указывают результаты многих исследований), что участие в большой команде приносит ее членам меньшее удовлетворение.
- **Работа в небольшой команде снижает вероятность чрезмерной специализации отдельных ее членов, отрицательно сказывающейся на их профессиональной карьере.** При реализации крупномасштабного проекта повышается вероятность узкой специализации его участников (Shaw, 1960). Например, кто-то из исполнителей работает только над пользовательским интерфейсом. Это приводит к излишним передачам работы из рук в руки между отдельными членами команды и не предоставляет возможности людям выйти за пределы своей узкой специализации и научиться чему-то новому.

См. также Проблемы, порождаемые передачей работы из рук в руки, будут рассматриваться в главе 11, “Организация коллективного труда”.

В ходе одного интересного исследования, посвященного выбору оптимальной численности команды, изучалась работа 109 команд. В малых командах работало от 4 до 9 человек, а в больших — от 14 до 18. По результатам этого исследования было сделано несколько выводов.

Члены небольших команд принимали более активное участие в деятельности своей команды, они были в большей степени преданныы своей команде, были лучше осведомлены о ее целях, лучше знали характер друг друга, рабочие роли и стили общения, демонстрировали большую степень взаимопонимания. Результаты исследования показывают также, что крупные команды более сознательно и ответственно относятся к подготовке повестки дня предстоящих собраний в сравнении с небольшими командами (Bradner, Mark, and Hertel, 2003, 7).

Словом, небольшая команда обеспечивает ряд несомненных преимуществ за единственным исключением: большая команда более сознательно и ответственно относится к подготовке повестки дня предстоящих собраний.

Производительность небольшой команды

Учитывая огромные преимущества малых команд, можно ожидать, что малые команды окажутся более производительными, чем большие. Именно к такому выводу пришел Дуг Путнам (Doug Putnam) из компании QSM после изучения 491 проекта, которые выполняли команды численностью от 1 до 20 человек. Начиная с 1978 года QSM собирает данные о производительности команд, занимающихся разработкой программного обеспечения и создала самую полную базу данных в отрасли разработки программного обеспечения. В этой базе данных содержится, в частности, информация о величине приложений, трудозатратах, отраслях, для которых создавались те или иные приложения, и о многом другом. Как таковая база данных QSM представляет огромную ценность для сравнения разных типов проектов.

Из базы данных QSM, включающей свыше 7 тысяч проектов, Дуг Путнам выбрал данные по 491 проекту, выполненному в период с 2003 по 2005 год и содержащему от 35 до 95 тысяч новых или модифицированных строк кода.¹ Величины проектов были равномерно распределены на шкале от 1 до 20 членов команды. Как видно из рис. 10.1, Путнам выяснил, что, чем меньше численность команды, тем выше производительность каждого отдельного члена команды. Однако разница в производительности между командами численностью от 1,5 до 7 человек была очень незначительной.

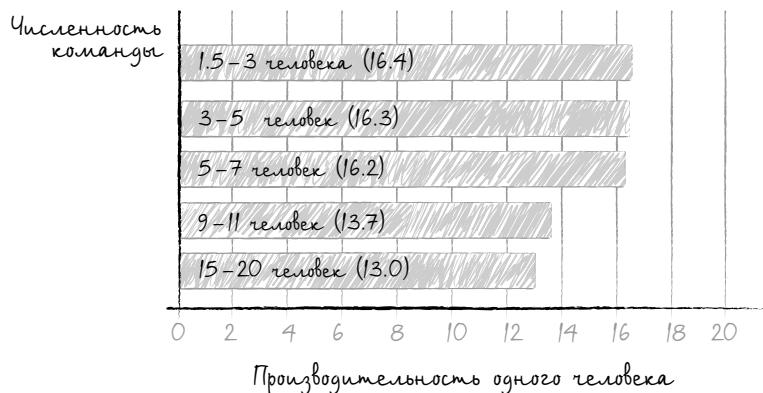


Рис. 10.1. Средняя производительность одного члена команды для команд разных размеров (перепечатано с разрешения QSM, Inc.; все права защищены)

Путнам оценил также совокупные трудозатраты на разработку рассмотренных им проектов. Здесь он пришел к следующему выводу: совокупные трудозатраты на выполнение проектов у малых команд меньше, чем у больших команд. Путнам пришел к заключению, что “использование больших команд приводит к большим трудозатратам

¹ Количество строк кода — показатель, в значительной степени скомпрометировавший себя (во многих случаях совершенно заслуженно). Однако я полагаю, что при использовании базы данных такого размера количество строк кода может быть вполне приемлемым показателем величины проекта и, таким образом, может использоваться при оценке производительности.

и большим издержкам, причем эта тенденция носит экспоненциальный характер. Наиболее экономной стратегией является использование малых команд, однако ярко выраженный нелинейный характер нарастания издержек начинает проявляться лишь после того, как величина команды достигает девяти и более человек". Эти результаты представлены на рис. 10.2.

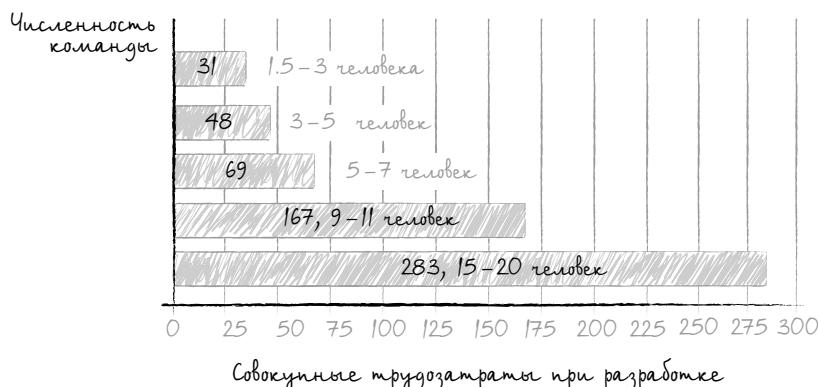


Рис. 10.2. При реализации проектов одинакового масштаба использование команд меньшего размера приводит к меньшим совокупным трудозатратам (перепечатано с разрешения QSM, Inc.; все права защищены)

Однако в большинстве случаев главной заботой является не столько минимизация совокупных трудозатрат, сколько минимизация продолжительности выполнения проекта. В конце концов, можем ли мы позволить себе ждать 40 лет, пока один-единственный разработчик выполнит проект, который должен быть готов к следующей весне? Влияние величины команды на продолжительность выполнения проекта показано на рис. 10.3. Из этого рисунка следует, что команда из 5–7 человек способна выполнить проект определенного объема за кратчайшее время. Командам меньшего размера придется затратить несколько больше времени. Еще раз обратите внимание на резкий рост продолжительности выполнения проекта, если величина команды достигает 9–11 человек.

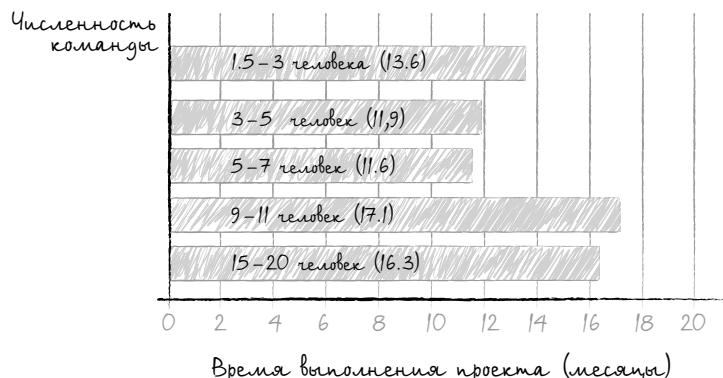


Рис. 10.3. При реализации проектов одинакового масштаба использование команд из пяти-семи человек позволяло минимизировать продолжительность выполнения проекта (перепечатано с разрешения QSM, Inc.; все права защищены)

См. также Существуют, конечно, большие проекты, которые невозможно выполнить силами одной команды (команды, “которую можно накормить двумя пиццами”). Такие проекты приходится выполнять силами нескольких команд (“команды команд”), а не одной огромной команды. Подробнее о выполнении крупномасштабных проектов рассказывается в главе 17, “Изменение масштаба Scrum”.

В ходе другого исследования, описанного в журнале *Communications of the ACM*, сравнивалась производительность больших и малых команд. Вот что пишет об этом исследовании ветеран отрасли Филлип Армор (Phillip Armour).

Большие команды (двадцать девять человек) создают примерно в шесть раз больше дефектов, чем малые команды (три человека), расходуя при этом гораздо больше денег. Вместе с тем большая команда выполняет примерно такой же объем работ, что и малая, затрачивая в среднем на двенадцать дней меньше, чем малая. Это по-разительный результат при том, что он полностью отвечает моему личному опыту работы над проектами, а этот опыт превышает тридцать пять лет (2006, 16).

Учитывая перечисленные выше убедительные доводы в пользу малых команд, я не думаю, что буду готов в обозримом будущем кормить свои команды тремя пиццами.

ВОЗРАЖЕНИЕ

“Особенность выполняемого нами проекта такова, что в его реализации должно участвовать много специалистов узкого профиля (аналитики, программисты, разработчики баз данных, программисты с клиентской стороны, программисты среднего уровня, тестеры, специалисты по автоматизации тестов и т.д.). Из этого следует, что мы не можем использовать малочисленную команду (например, из пяти-девяти человек)”.

Несмотря на то что проект может требовать ряда узкопрофильных работ, это наверняка не потребует наличия в команде узкого специалиста в каждой из перечисленных вами областей. В команде из девяти человек будет трудно (а то и вообще невозможно) равномерно распределить нагрузку между всеми членами команды. Структура команды, часть членов которой может работать только в рамках своей узкой специализации, а остальные могут работать в двух или более областях, существенно облегчает задачу балансировки нагрузки, которая ложится на плечи разных специалистов. Если в команде по крайней мере часть людей может работать (и работает) в нескольких областях, это вызывает у людей ощущение ответственности за продукт в целом и отучает их мыслить по принципу “я свое дело сделал, а там хоть трава не расти”.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если численность вашей команды составляет девять человек или более, попытайтесь разделить ее на две команды после завершения текущего спринта. Затем, по прошествии не менее двух спринтов, обсудите с людьми, привело ли такое разделение к позитивному результату.
- Для каждой команды, численность которой составляет от пяти до девяти человек, рассмотрите возможность ее разделения на две команды.

Команды, специализирующиеся на определенных функциональных возможностях

Когда я впервые начал консультировать одну калифорнийскую студию, занимающуюся разработкой видеоигр, ее команды специализировались на определенных элементах и объектах, присутствующих в разрабатываемой видеоигре. Для каждого персонажа видеоигры была предусмотрена отдельная команда. Были команды, разрабатывающие оружие, команды, разрабатывающие транспортные средства, и т.п. Такой способ организации команд порождал определенные проблемы (например, вооружение было слишком слабым, чтобы уничтожить монстров, цвета были слишком темными, чтобы пользователь мог увидеть тайные проходы, а препятствия были такими, что в отчаяние приходили даже самые терпеливые пользователи).

В случае более традиционных, корпоративных, проектов мы наблюдаем проблемы того же порядка, когда команды организуются в соответствии с уровнями приложения. Например, типичная ошибка, характерная для ранних стадий выполнения проекта, архитектура которого представлена на рис. 10.4, могла бы заключаться в создании четырех команд: команда богатого клиента, команда веб-клиента, команда среднего уровня и команда базы данных. Создание *компонентных команд*, подобных такой, порождает множество проблем, в том числе следующие.

- Ухудшение взаимодействия между уровнями
- Ощущение достаточности проектного решения, предусмотренного контрактом
- Завершение спринтов без наращивания продукта, которое можно было бы предъявить заказчику



Рис. 10.4. Типичная трехуровневая архитектура

Если структуризация команд в соответствии с уровнями архитектуры является неправильным подходом, то какой подход является правильным? Вместо того чтобы организовывать команды на основе компонентов, каждая из команд, участвующих в проекте, должна в идеальном случае отвечать за сквозную (от начала до конца) реализацию полностью работоспособных (протестированных) функциональных возможностей (“функциональная команда”). Такая команда, работающая над приложением, показанным на рис. 10.4, должна охватывать все уровни архитектуры. Она могла бы разрабатывать одну функциональную возможность, которая включает как уровень базы данных, так и сервисный уровень и интерфейс пользователя богатого клиента. В том же или в следующем спринте она могла бы разрабатывать функциональную возможность, охватывающую веб-клиента, сервисный уровень и уровень базы данных.

См. также Важность обеспечения сквозной функциональности подробно обсуждается в главе 14, “Сprintы”.

Реализация многокомандных проектов с помощью команд, специализирующихся на определенных функциональных возможностях, обладает рядом преимуществ.

- **Такие команды способны лучше оценить влияние проектных решений.** В конце каждого очередного спринга такая команда сможет предъявить заказчику сквозную функциональную возможность, охватывающую все уровни технологического “многослойного пирога” приложения. Это позволяет максимизировать обучение членов команды принятым ими проектным решениям по соответствующему продукту (устраивает ли пользователей предложенный вариант функциональной возможности?) и принятым ими проектным решениям технического характера (в какой мере этот подход к реализации данной функциональной возможности устраивает нас самих?).
- **Экономия на излишних передачах из рук в руки.** Передача работы из рук в руки (от одной группы к другой или от одного работника к другому) слишком расточительна. В случае компонентной команды возникает риск, что будет разработано слишком мало или, наоборот, слишком много функциональности, что будет разработана неправильная функциональность, что какая-то часть функциональности уже не понадобится, и т.д.

См. также О других проблемах, вызванных передачей работы из рук в руки, рассказывается в главе 11, “Организация коллективного труда”.

- **Гарантия общения между собой людей, которым это необходимо.** Поскольку команды, специализирующиеся на определенных функциональных возможностях, включают всех специалистов, требующихся для прохождения всего пути, от идеи до ее практического воплощения, т.е. до появления реально работающей, протестированной функциональной возможности, то использование такой команды гарантирует, что люди, обладающие требуемой квалификацией, смогут ежедневно общаться между собой.
- **В случае компонентной команды возникает риск для исполнения календарного плана.** Работа компонентной команды приобретает реальную ценность лишь после ее интеграции в соответствующий продукт командой, специализирующейся на определенных функциональных возможностях. Трудоемкость интеграции работы компонентной команды в соответствующий продукт должна оцениваться командой, специализирующейся на определенных функциональных возможностях независимо от того, произойдет ли это в том же спринге, в котором она выполнялась (в лучшем случае), или в каком-то из последующих. Оценить такую работу иногда нелегко, поскольку для этого нужно, чтобы команда, специализирующаяся на определенных функциональных возможностях, оценила работу по интеграции, ничего не зная о качестве соответствующего компонента.
- **Возможность сосредоточиться на реализации определенных функциональных возможностей.** У команды может возникнуть соблазн вернуться к привычкам,

которые сформировались у нее до перехода к Scrum. Создание команд, призванных реализовывать определенные функциональные возможности, а не на основе тех или иных архитектурных элементов или технологий служит постоянным напоминанием о сфокусированности Scrum на реализации в ходе каждого спринта определенных функциональных возможностей.

ВОЗРАЖЕНИЕ

“Мое приложение слишком сложное. Я не в состоянии обеспечить сквозную функциональность в рамках одного спринта”.

Научиться определять небольшие порции функциональности — вот одно из первых серьезных препятствий, которые приходится преодолевать новой Scrum-команде. Я помню свой первый Scrum-проект. Вначале нам было очень нелегко найти что-то такое, что можно было бы реализовать менее чем за шесть недель. Вспоминая ту систему много лет спустя, я вижу немало способов, с помощью которых мы могли бы разделить эту работу. Более того, я вижу достаточно способов разделить работу даже в том случае, если бы нам сейчас пришлось выполнять спринты продолжительностью в один день.

Приобретая опыт, члены команды будут находить все новые и новые способы разделить функциональные возможности и реализовывать таким образом сквозную функциональность в рамках каждого спринта. Если подчас это кажется невозможным, то лишь потому, что команды не структурированы надлежащим образом. Прежде чем расписываться в своей неспособности решить данную проблему, проанализируйте еще раз внимательно состав своей команды и квалификацию каждого из ее членов.

Используйте компонентные команды экономно

Несмотря на то что предпочтение следует по возможности отдавать командам, призванным реализовывать определенные функциональные возможности, иногда целесообразно использовать компонентную команду. Компонентная команда (в том смысле, в каком я употребляю этот термин в своей книге) — это команда, которая разрабатывает программное обеспечение, подлежащее передаче какой-то другой команде, а не непосредственно пользователям. Примерами компонентной команды могут служить команда, разрабатывающая уровень объектно-реляционного отображения между приложением и базой данных, или команда, разрабатывающая некий повторно используемый элемент пользовательского интерфейса.

Важно, чтобы к концу каждого спринта компонентная команда по-прежнему вырабатывала высококачественный, протестированный, потенциально готовый к использованию код. Однако новые возможности, создаваемые компонентной командой, обычно бесполезны сами по себе. Вернемся на минуту к приведенным выше примерам. Уровень объектно-реляционного отображения, разработанный одной из компонентных команд, представляет интерес для конечных пользователей только в том контексте, в каком он используется командами, призванными реализовывать определенные функциональные возможности. Но что можно сказать о команде, разрабатывающей такие повторно используемые элементы пользовательского интерфейса, как специализированные выпадающие списки, сетки для ввода данных и т.п.? Уж они-то, несомненно, представляют интерес для конечных пользователей, не так ли?

Да, но, опять-таки, лишь в контексте других функциональных возможностей. Конечного пользователя не интересует новая сетка для ввода данных до тех пор, пока она не будет встроена в какую-либо страницу или экран.

Создавайте компонентные команды только тогда, когда они требуются командам, реализующим функциональные возможности

Поскольку результаты работы компонентной команды передаются какой-то другой команде, именно эта команда выступает в качестве владельца продукта по отношению к компонентной команде. Если вашей команде требуются результаты работы моей команды, то вы выступаете в качестве владельца продукта по отношению к моей команде. В таком случае вы исполняете все обязанности эффективного владельца продукта. В начале очередного спринта вы должны помочь расставить приоритеты работ, которые мне предстоит выполнять. В конце спринта вы примете или отвергнете эту работу, сообщив мне свою оценку выполненной мною работы.

Вам будет нелегко расставить приоритеты работ и сообщить свою оценку выполненной мною работы, если моя команда работает с большим опережением по сравнению с вашей командой. Именно поэтому компонентной команде не следует разрабатывать новые возможности до тех пор, пока одна или несколько команд, призванных реализовывать функциональные возможности, не будут готовы к работе. Когда компонентная команда сильно опережает требования команд, реализующих некоторые функциональные возможности, начинаются догадки о том, какие возможности могут потребоваться в ближайшем будущем. В результате очень часто появляются компоненты или инфраструктуры, которые не используются реализующими функциональные возможности командами. Все, в том числе и создаваемое компонентными командами, должно разрабатываться в контексте функциональности, видимой конечному пользователю.

Роб был старшим разработчиком в компонентной команде, разрабатывавшей уровень объектно-реляционного отображения, которым должны были пользоваться многие из 15 команд, реализующих различные функциональные возможности данного проекта. Поначалу команде Роба предстояло сделать выбор между разработкой этой технологии собственными силами и разработкой с использованием какого-либо коммерческого продукта или продукта с открытым исходным кодом. Члены команды приняли решение (впрочем, весьма спорное) вести разработку собственными силами. Не имея возможности доказать окружающим правильность этого решения, Род и его команда начали изо всех сил работать на опережение потребностей команд, призванных реализовывать конкретные функциональные возможности. Вместо того чтобы тесно сотрудничать с одной или несколькими такими командами, компонентная команда Роба начала строить догадки о проектном решении разрабатываемой системы в целом. В течение двух месяцев (двух спринтов) команда Роба не выдала ни одного готового результата своей работы командам, призванным реализовывать функциональные возможности. Когда закончился третий месяц и когда они, наконец, представили первоначальную версию, оказалось, что она не отвечает потребностям и ожиданиям других команд.

Команде Роба следовало бы вместо этого работать в очень тесном контакте с командами, реализующими функциональные возможности, и добавлять в свою работу новые

возможности только в контексте функциональных возможностей, реализуемых этими командами. Это обусловило бы гораздо более тесное сотрудничество между компонентной командой и командами реализации, что повлекло бы за собой создание компонентной командой именно тех возможностей, которые нужны. Команда Роба могла бы, например, реализовать во время первого спринта только возможность записи в базу текстовых данных фиксированной длины. Команды реализации, которые получили бы в свое распоряжение эту возможность, не могли бы записывать в базу числовые данные, даты и т.п. Кроме того, они не могли бы читать из базы данных. Все, что они могли бы делать, — это записывать в базу текстовые данные фиксированной длины и сообщать Робу и его команде об удобстве пользования этим компонентом.

Возможно, наилучший способ обеспечить обратную связь с компонентной командой — временно укомплектовать ее людьми из команд, реализующих функциональные возможности. Разработчик, прикомандированный к компонентной команде и знающий, что вскоре ему предстоит вернуться в свою прежнюю команду, наверняка позаботится о том, чтобы работа такой компонентной команды принесла как можно большую пользу его “родной” команде.

Принятие решения о целесообразности использования компонентной команды

По возможности следует создавать команды, реализующие функциональные возможности, а не компонентные команды. Я предпочитаю исходить из того, что все команды, которые будут работать над многокомандным проектом, будут командами, реализующими функциональные возможности. Я готов отказаться от этого предложения, но только при условии, что формирование одной или более компонентных команд реально поспособствует созданию соответствующего продукта. Я считаю, что рассматривать возможность формирования компонентной команды следует лишь при выполнении большинства перечисленных ниже условий.

- **Компонентная команда будет создавать нечто такое, чем смогут воспользоваться несколько команд, реализующих функциональные возможности.** Если какой-либо компонент будет использоваться только одной из команд, реализующих функциональные возможности, то пусть эта команда сама создает данный компонент. Это гарантирует, что новая возможность будет реализована в контексте потребностей и ожиданий команды, что, в свою очередь, повышает вероятность ее эффективного использования. Даже когда компонентная команда создает что-то полезное для нескольких команд, более эффективная стратегия зачастую заключается в том, чтобы команда, реализующая функциональные возможности, создала необходимую ей функциональность, а другие команды впоследствии выполнили рефакторинг и обобщили эту функциональность по мере расширения своих потребностей.

См. также Подробнее о недостатках “многозадачности” рассказывается далее в этой главе, в разделе “Один человек — один проект”.

- **Компонентная команда сужает возможности, связанные с совместным использованием специалистов.** В случае некоторых многокомандных проектов некоторые

узкоспециализированные исполнители используются одновременно несколькими командами. Несмотря на то что совместное использование командами одних и тех же специалистов в некоторых случаях действительно необходимо, злоупотребление им лишь вредит делу, поскольку специалист не может уделить должного внимания ни одной из команд. Можно рассмотреть возможность создания компонентной команды в случае, если это позволит эффективнее контролировать степень совместного использования командами одних и тех же специалистов.

- **Риск применения несовместимых подходов перевешивает недостатки компонентной команды.** Если мы решили создавать какой-либо совместно используемый компонент или сервис силами нескольких команд, реализующих функциональные возможности, то нужно помнить о двух взаимосвязанных рисках, которые возникают в этом случае. Во-первых, появляется риск того, что каждая из команд реализации создаст собственный вариант решения одной и той же проблемы. Во-вторых, появляется риск того, что каждая из этих команд создаст надстройку над созданным предыдущими командами такого рода, но сделает это, руководствуясь собственными представлениями. Указанные риски могут быть как велики, так и не очень — в зависимости от того, какая именно совместно используемая функциональность создается. Когда риск применения несовместимых подходов оказывается достаточно большим, разумной альтернативой является создание компонентной команды.
- **Это может заставить говорить тех, кто не мог этого делать при иных обстоятельствах.** Люди, как правило, разговаривают больше с членами своей команды, а не с “посторонними”. Это верно даже в случае Scrum-проекта. Более того, это касается именно Scrum-проектов, поскольку члены команд, выполняющих Scrum-проекты, склонны совершенно однозначно идентифицировать себя со своими командами. Этим обстоятельством можно воспользоваться, формируя команды из людей, которые должны работать вместе, но иначе не могут свободно общаться друг с другом. Если прошлый опыт показывает, что, скажем, программисты искусственного интеллекта, работающие над определенным проектом, не общались между собой достаточно часто, это может оправдывать краткосрочное использование компонентной команды (если, кроме того, есть и иные причины для создания такой команды).
- **Вы уверены во временном характере использования данной компонентной команды.** Компонентная команда не должна оставаться “навсегда”, как теща, приехавшая к вам погостить “буквально на пару дней”, но не намеренная уезжать и через месяц... Команда должна разработать функциональность, ради реализации которой она была создана, и распасться после этого как можно скорее. При создании компонентной команды необязательно знать, когда именно она будет расформирована, однако вы должны представлять себе примерный срок ее существования или какого результата она должна добиться к моменту, когда ее миссию можно будет считать завершенной. Поскольку компонентная команда является своего рода отклонением от идеала (когда все команды реализуют функциональные возможности), не следует создавать компонентную команду, которая может выглядеть так, будто ее создали навечно.

Признавая определенные выгоды от использования компонентной команды, я хочу подчеркнуть еще раз, что подавляющее большинство команд, участвующих в реализации крупного проекта, должны быть командами, реализующими функциональные возможности. Вес Уильямс (Wes Williams) и Майк Старт (Mike Stout) описали, что произошло в Sabre Airline Solutions, когда началось создание компонентных команд.

С точки зрения пользователя, работа не была доведена до конца. Команды работали в разное время над реализацией разных функциональных возможностей с разными критериями приемки. Впоследствии многое пришлось переделывать. Команды обвиняли друг друга в неполной реализации функциональных возможностей, в неудачных сборках, тестах и т.д. Оглядываясь назад, можно сказать, что следовало создавать команды только для реализации функциональных возможностей (2008, 359).

Кто принимает эти решения

В идеальном случае команда сама принимает решения о том, как ей следует структурироваться. Если команде доверяют решать вопросы, связанные с созданием продукта, то почему бы не доверить ей и принятие решения о собственной структуре? Но несмотря на то что команды привыкли принимать технические решения, обычно им не хватает опыта для принятия организационных решений. Поэтому поначалу команда может оказаться не по плечу принятие решений о собственной структуре.

Мне довелось знакомить со Scrum не одну сотню команд. Я заметил, в частности, что очень часто сразу после знакомства со Scrum люди говорят примерно следующее: “Scrum — очень удачная находка для нашей компании. Правда, эта методология подойдет для всех групп, кроме нашей”. Архитекторы добавляют: “После того как мы разработаем предварительный вариант архитектуры, станет понятно, какую пользу это принесет программистам и тестерам”. Проектировщики пользовательского интерфейса говорят: “После того как мы выполним предварительное исследование, станет по-настоящему понятно, какую пользу это принесет архитекторам, программистам и тестерам”. Тестеры думают так: “Было бы просто замечательно, если бы все работали в столь тесном контакте друг с другом, после чего передавали бы нам результаты своей работы для большого цикла интегрированного тестирования”.

Если бы мы попросили членов команд с этими общими первоначальными представлениями разработать структуру своего многокомандного проекта, то не было бы ничего удивительного, если бы они вернулись к нам с планами для команды архитекторов, команды программистов, команды проектировщиков пользовательского интерфейса и команды тестеров. Разумеется, я немного утрирую, но склонность именно к такому образу мышления преобладает настолько, что у людей неминуемо появляется соблазн организовать свою работу в соответствии с ним.

Таким образом, решения, касающиеся организации команд, вероятнее всего, вначале должны будут принимать функциональные менеджеры, руководители проектов, Scrum-мастера или те, на кого возложена ответственность за переход к Scrum. Все лица, принимающие участие в принятии этих решений, должны попросить свои команды предоставить информацию, которая может понадобиться для принятия таких решений (особенно это касается членов команд, уже имеющих опыт работы со Scrum или другими гибкими методологиями разработки).

То, что правильно сегодня, может оказаться неправильным завтра

Выбирая подходящую структуру команды, важно помнить, что никакая структура не может быть вечной. Если текущая структура ухудшает способность команды (или проекта) использовать Scrum, то этот вопрос нужно поднять в ходе ретроспективы, проводимой в конце спринта. Разумеется, постоянно изменять структуру команды нежелательно, поскольку, чтобы привыкнуть к той или иной структуре, членам команды требуется некоторое время; но если текущая структура команды заведомо неэффективна, то ее нужно менять безо всяких сомнений.

Когда члены команды приобретут достаточный опыт использования Scrum, целесообразно привлечь их к более активному участию в принятии решений о структуре команды, в том числе о том, какие именно команды требуются для реализации соответствующего проекта, какой должна быть та или иная команда (компонентной или предназначеннной для реализации функциональных возможностей) и каким должен быть персональный состав той или иной команды.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Составьте полный перечень команд, принимающих участие в реализации вашего текущего проекта. Определите тип каждой из них (компонентная или предназначена для реализации функциональных возможностей). Для каждой компонентной команды рассмотрите условия, перечисленные в разделе "Принятие решения о целесообразности использования компонентной команды". Рассмотрите возможность реструктуризации команды, если выполняются не все эти условия.

"Самоорганизующаяся" не означает "случайно сформированная"

Способность команды самоорганизовываться на основе поставленных перед ней целей абсолютно необходима для всех гибких методологий разработки, включая Scrum. Вообще говоря, Agile Manifesto указывает на самоорганизующиеся команды как на один из ключевых принципов, утверждая, что "источником наилучших архитектур, требований и проектных решений являются самоорганизующиеся команды" (Beck et al., 2007). Принимая решение о том, как лучше всего достигать поставленных целей, некоторые команды приходят к выводу, что все ключевые технические решения должны приниматься одним членом команды. Другие команды предпочитают разделять ответственность за технические решения по "техническим водоразделам": наш специалист по базам данных принимает решения по базам данных, а наш самый опытный программист на C# принимает решения по C#. Некоторые команды полагают, что тот, кто работает над реализацией определенной функциональной возможности, должен принимать все решения, касающиеся этой функциональной возможности; в то же время он обязан сообщить команде о результатах своего решения.

Здесь есть два ключевых момента. Во-первых, каждая команда выбирает собственный, наиболее подходящий для себя способ организации (и это правильно). Во-вторых, используя коллективный разум команды, как правило, можно изыскать более подходящий способ организации на основе работы, выполняемой данной командой. Это лучше, чем полагаться исключительно на мудрость руководителя отдела кадров. Однако преимущество предоставления команде возможности самоорганизации заключается не в том, что она находит некий оптимальный способ организации своей работы, упущеный из виду руководителем. Скорее преимущество заключается в том, что, предоставив команде возможность самоорганизации, вы подталкиваете ее членов к тому, чтобы они почувствовали полную ответственность за проблему, которую им предстоит решить.

Типичная критика в адрес самоорганизующихся команд звучит примерно так: “Мы не можем просто собрать вместе восемь случайно выбранных нами людей, сказать им “самоорганизуйтесь!” и ждать, пока из этого получится что-нибудь хорошее”. Да, я тоже не знаю, насколько правилен такой подход, но, составляя Scrum-команду, которую можно “накормить двумя пиццами”, мы наверняка не собираем для этого восемь случайно выбранных нами людей. Вообще говоря, те, кто отвечают за инициирование Scrum-проекта, должны потратить немало сил, чтобы отобрать исполнителей, из которых получится настоящая команда.

См. также В главе 12, “Управление самоорганизующимся коллективом”, рассказывается о том, как лидеры оказывают ненавязчивое, малозаметное, но позитивное влияние.

В оригинальной статье, содержащей описание Scrum, Такучи (Takeuchi) и Нонака (Nonaka) сформулировали принцип “мягкого контроля” (“subtle control”), как одного из шести принципов Scrum. Они утверждают, что кадровые решения являются одной из ключевых обязанностей руководства.

Правильный подбор исполнителей для команды, которой предстоит реализовать проект, с одновременным отслеживанием изменений в групповой динамике и подключением или увольнением членов, когда это необходимо, [является одной из ключевых обязанностей руководства]. “В случае чрезмерного сдвига баланса в сторону радикализма мы включали в команду более старшего по возрасту и консервативного члена, — сказал один из руководителей компании Honda. — Мы подбираем исполнителей проекта весьма скрупулезно, тщательно взвесив все «за» и «против». Мы анализируем разные кандидатуры, чтобы убедиться в том, что они действительно подходят данной команде” (1986, 144).

Как подобрать подходящих членов команды

Если вы руководитель по кадровым вопросам или можете каким-то иным способом влиять на состав команд в своей организации, то вам нужно знать, как правильно подобрать членов команды.

- **Включайте в состав команды всех необходимых специалистов.** Поскольку речь идет о формировании межфункциональной команды, важно, чтобы в команде

были представлены все специалисты, необходимые для того, чтобы команда могла пройти весь путь от идеи до практической реализации соответствующей функциональной возможности. Поначалу это может означать, что численность команды окажется несколько большей, чем хотелось бы. Но со временем каждый из членов Scrum-команды освоит какие-то из навыков, которыми владеют его товарищи по работе. Это является естественным результатом работы в Scrum-команде. После того как некоторые из членов Scrum-команды освоят более широкий круг навыков, остальных членов можно будет перевести в другие команды.

- **Сбалансируйте уровни технического мастерства.** Принимая во внимание соображения, касающиеся величины команды, нужно стремиться сбалансировать уровни технического мастерства в команде. Например, если в команде есть три старших программиста, но отсутствуют программисты менее опытные, то старшим программистам придется заниматься кодированием не очень интересных и важных программ, что может показаться им довольно скучным делом. В то же время для менее опытного и квалифицированного программиста выполнение такой работы может оказаться вполне интересным и полезным занятием. К тому же он может научиться чему-нибудь полезному для себя у старших программистов.
- **Сбалансируйте знание членами команды той или иной предметной области.** Точно так же, как мы стремимся сбалансировать уровни технического мастерства в команде, нужно стремиться к обеспечению баланса между специалистами, обладающими глубокими познаниями в той предметной области, в которой мы работаем, или в той проблеме, которую мы пытаемся решить. Тем самым я вовсе не хочу сказать, что если у нас есть возможность составить команду полностью из экспертов в определенной предметной области, то нам следует пренебречь такой возможностью. Скорее, нам следует принять во внимание долгосрочные цели нашей организации. Одной из этих целей, вероятно, является формирование знаний в предметной области во всей организации. Вам будет очень непросто добиться этого, если все ваши эксперты в определенной предметной области будут собраны в одной команде.
- **Стремитесь к разнообразию.** Разнообразие может относиться ко многому, в том числе к половой, расовой и культурной принадлежности человека. Наверное, не менее важным может быть восприятие людьми тех или иных проблем, способ принятия ими решений, объем информации, необходимой им для принятия решений, и т.д. Однородные команды достигают консенсуса быстрее, чем неоднородные, однако при этом учитывают далеко не все возможные варианты (Mello and Ruckes, 2006).
- **Учитывайте фактор привычки.** Чтобы члены команды научились работать вместе, требуется определенное время. Нужно стремиться подбирать людей таким образом, чтобы в команде оказалось как можно больше людей, которым уже приходилось работать вместе и этот опыт совместной работы оказался для них успешным. Формируя новую команду, учитывайте, как долго ее члены смогут работать вместе до того, как кто-то из них или все они будут переведены в другие команды.

ВОЗРАЖЕНИЕ

“Мы не можем самоорганизоваться, поскольку в составе нашей команды есть бывший технический руководитель, который самостоятельно принимает все решения, не давая нам ни малейшего шанса обсудить проблему”.

Если возможно, отойдите с этим бывшим техническим руководителем в сторонку и расскажите ему об этой проблеме. Скажите ему, что даже в ситуациях, когда ему заранее известно “правильное” решение, ему иногда следует воздерживаться от высказывания собственного мнения до тех пор, пока это не сделают остальные члены команды. Спросите, не кажется ли ему, что команда смогла бы принять правильное решение, если бы он представил ей свои соображения лишь как мнение, а не как окончательное решение. Добейтесь от него обещания, что он будет относиться к членам команды, как наставник, который не только контролирует, правильные ли принимаются решения, но еще и делает все, чтобы квалификация членов команды со временем повысилась настолько, что при выполнении последующих проектов, в которых он, возможно, не будет участвовать, они смогут принимать правильные решения.

“Моя команда не способна к самоорганизации: ее члены слишком пассивны и ждут моих указаний”.

Если они ожидают, что вы будете ими руководить, то вам не остается ничего другого, как ждать, пока они не научатся действовать самостоятельно. Как Scrum-мастер команды внушиите им, что ваша задача — оказывать им поддержку, а не принимать решения вместо них. Если вы — один из членов команды, то вам совсем не обязательно всегда держать свое мнение при себе и помалкивать. В то же время вам следует находить способы подключения других членов команды к процессу принятия решений и воздерживаться от самостоятельного принятия решений во всех без исключения случаях. Например, прежде чем высказать собственное мнение, попробуйтесь ответить на вопросы членов команды.

“Моя команда слишком молода, и ей не хватает опыта для самоорганизации”.

Если им хватает опыта, чтобы разработать программный продукт, то им наверняка хватит опыта, чтобы понять, как самоорганизоваться. Если это не так, помогите им дальним советом, научите их. Нередко за возражением такого рода скрывается кое-что другое, например: “Мне кажется, что команда не сможет самоорганизоваться так, как мне того хотелось бы”. Очень плохо! Необходимо ненавязчиво и незаметно контролировать команду в том, что касается ее состава и реализации поставленной перед ней цели, но не в том, что касается ее повседневной работы.

Один человек — один проект

Люди, которым приходится работать одновременно в нескольких проектах, неизбежно начинают халтурить, недорабатывать. Многозадачность — попытка работать сразу над несколькими проектами или выполнять несколько дел одновременно — является одним из самых серьезных факторов, подрывающих производительность команды. Тем не менее многозадачность, к сожалению, стала одним из наиболее часто используемых инструментов руководителя, постоянно перегруженного работой. Причина этого, как мне кажется, — создание иллюзии прогресса; у руководителя

появляется ощущение, что все проблемы как-то решаются. В действительности же во многих случаях проблемы не решаются, а напротив, лишь усугубляются.

Рассмотрим случай Джона, директора разработки баз данных, в непосредственном подчинении которого находилась группа администраторов баз данных (АБД). АБД было недопустимо мало в сравнении с количеством программистов, тестеров и сотрудников других категорий в компании Джона. Перед Джоном стояла непростая задача: распределить своих подчиненных (включая самого себя) между проектами, количество которых заметно превосходило численность АБД. Джон решил создать электронную таблицу, подобную той, которая показана на рис. 10.5. Электронная таблица Джона позволяла ему распределить АБД по разным проектам, что он и выполнил — (вплоть до пятипроцентного уровня). Пять процентов от восьмичасового рабочего дня равняются 24 минутам. Основываясь на этой электронной таблице, Джон сказал, что Билл может тратить по 24 минуты на проекты Napa и PMT, Ахмед может тратить также по 24 минуты на проекты PMT и Spinwheel, и т.д.

	Napa	Cronos	SpongeBob	Dodge City	DB2 Migration	Europa	PMT	Spin Wheel
Bill	5%	15%	50%	25%	5%			
Ahmed		90%				5%	5%	
Sir	25%			25%	5%	25%		
Tor	25%		50%		10%		15%	
Robert		20%					5%	75%
Jon	5%	10%	0%	10%		5%	10%	

Рис. 10.5. Часть электронной таблицы Джона, предназначенной для распределения АБД по разным проектам

Действительно ли Джон полагал, что каждый день Билл будет прекращать работать над проектом Napa через 24 минуты? Разумеется, нет. Он, наверное, считал, что Билл в состоянии контролировать собственный рабочий график, исходя из того, что каждую неделю в его распоряжении есть $24 \times 5 = 120$ минут, в течение которых он может работать над проектом Napa. По сути, в сложившейся ситуации проблему распределения наличных ресурсов (которую он не был в состоянии решить), Джон просто переадресовал членам своей команды. На самом же деле Джону следовало переадресовать эту проблему своему начальнику.

Переадресация проблем команде зачастую оказывается замечательной стратегией. Более того, переадресация проблем команде является основой методологии Scrum. Однако, когда какая-либо проблема переадресовывается команде, ей нужно предоставить полномочия для решения данной проблемы. В случае Джона и подчиненных ему АБД очевидно, что следует рассмотреть такой вариант, как сокращение числа параллельно выполняемых проектов. Не имея полномочий принять такое решение, они оказались в совершенно безвыходной ситуации.

Разумеется, им удалось решить эту проблему ничуть не лучше, чем Джону. Они прибегли к проверенному временем методу: работать каждый раз над тем проектом, который не терпит отлагательства.

Когда задач слишком много, время на их выполнение сокращается

Ким Кларк (Kim Clark) и Стивен Уилрайт (Steven Wheelwright) изучали влияние многозадачности на производительность. Результаты их исследования, представленные на рис. 10.6, показывают, что совокупное количество времени, затрачиваемого на выполнение задачи, увеличивается, когда человеку приходится работать над двумя задачами. Однако после этого, согласно результатам исследования, проведенного Кларком и Уилрайтом, количество времени, затрачиваемого на выполнение задачи, сокращается. Фактически при параллельном выполнении трех задач количество времени, затрачиваемого на выполнение задач, сокращается настолько, что оказывается меньшим, чем в случае, когда человеку приходится выполнять только одну задачу (1992, 242).

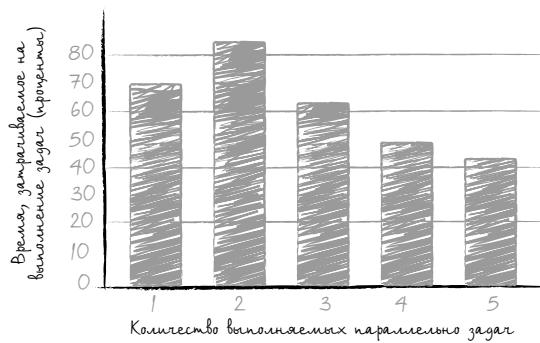


Рис. 10.6. Количество времени, затрачиваемого на выполнение задач, сокращается при параллельном выполнении трех задач и более задач

Если вам приходится выполнять только одну задачу, то почти наверняка время от времени будут возникать ситуации, когда вам придется приостанавливать работу над ней. Возможно, вам придется ждать ответного звонка от кого-либо из ваших коллег, ответа по электронной почте, утверждения предложенного вами проектного решения и т.п. Наверное, именно этим объясняется результат, полученный Кларком и Уилрайтом (человек, которому приходится работать параллельно над двумя задачами, тратит на выполнение задач больше времени, чем тот, кто работает только над одной задачей). Однако нужно принять во внимание, что Кларк и Уилрайт проводили свое исследование в начале 1990-х годов.

Что изменилось с тех пор? Вспомним для начала об электронной почте, мгновенном обмене сообщениями, широком распространении мобильных телефонов и множестве других средств общения. Я полагаю, что столбцы на рис. 10.6 нужно сместить на один интервал влево, отразив таким образом нынешнее ускорение темпа жизни. Я отчетливо помню работу, которой занимался в 1992 году, когда Кларк и Уилрайт опубликовали результаты своего исследования. Временами я сидел за рабочим столом, скучая от безделья и не зная, чем заняться. Конечно, с тех пор многое изменилось.

Темп современной жизни резко ускорился. Чтобы вас просто считали хорошим работником, нужно работать гораздо больше, чем в 1992 году. Нужно гораздо больше читать, перерабатывать гораздо большие объемы информации, гораздо усерднее трудиться. “Просто быть работником” нужно рассматривать сегодня как первоочередную задачу для каждого из нас. Первый проект, в котором мы участвуем, можно рассматривать как вторую задачу; в этом случае мы уже достигаем оптимальной производительности. Все остальные проекты, к выполнению которых нас подключают, лишь снижают нашу производительность.

Одной из главных причин высокой неэффективности многозадачности являются связанные с ней издержки переключения с одной задачи на другую. Приступая к выполнению одной задачи, переключаясь на другую, а затем возвращаясь к первой, мы несем колоссальные накладные расходы. Чем больше задач (или проектов), в которых мы участвуем, тем вероятнее, что нам придется прерывать свою работу над той или иной задачей в самые неподходящие моменты. Исследование работы членов одной из команд, занимавшихся разработкой программного обеспечения, показало, что им приходилось прерывать работу каждые 11 минут (Gonzales and Mark, 2004). Если вы читаете эту главу у себя в офисе, то вам, наверное, пришлось хотя бы раз прервать чтение.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если вы — руководитель, составьте список своих непосредственных подчиненных и проектов, в выполнении которых принимает участие каждый из них. Если кто-то из них участвует в выполнении более чем двух проектов, немедленно найдите способ исправить эту ситуацию. Если вам это уже удалось, попытайтесь добиться, чтобы каждый из ваших подчиненных участвовал в выполнении только одного проекта. Оцените ситуацию по истечении двух спринтов.

Когда многозадачность является подходящим вариантом

Все это было сказано вовсе не для того, чтобы создать у читателей впечатление, будто при выполнении проектов следует полностью исключить многозадачность. Иногда многозадачность бывает полезна. Главное — помнить о том, что человек, участвующий параллельно в нескольких проектах, скорее всего, выполнит меньший объем совокупной работы, чем если бы он полностью сосредоточился на чем-то одном.

Давайте вернемся к примеру с Джоном и его АБД. Допустим, что каждый из АБД может выполнять за день 20 заданий, связанных с базой данных (пусть все эти задания имеют примерно одинаковые объемы). АБД, которому посчастливилось работать только над одним проектом, способен достичь этого уровня производительности. Однако АБД, которому придется работать над двумя проектами, сможет выполнять за день лишь 16 заданий, связанных с базой данных. А АБД, которому придется работать над тремя проектами, сможет выполнять за день лишь 14 таких заданий. Допустим, что один из наших АБД параллельно участвует в двух проектах и должен делить свое время поровну между этими проектами. Таким образом, он сможет выполнять за день по 8 заданий по каждому проекту. Это может оказаться оптимальным вариантом использования его рабочего времени, если ни один, ни другой проекты не требуют от него ежедневного выполнения 20 заданий. Если ни один из этих проектов не

требует от АБД выполнять за день более 8 заданий, то будет лучше, если этот АБД по-делит свое время между двумя проектами, чем полностью посвятит себя выполнению какого-то одного из них. Из этого примера следуют несколько важных правил.

- Вообще говоря, для большинства членов команды, многозадачность — нежелательное явление.
- Многозадачность может оказаться приемлемым вариантом, если кто-то из исполнителей не может быть полностью или практически полностью задействован в каком-то одном проекте. Если вернуться к рис. 10.5 и АБД Джона, то мы увидим, что в проекте Connect задействовано три человека, причем сумма долей времени, которые уделяют этому проекту Ахмед, Роберт и Джон, превышает 100%. Более удачным решением было бы, наверное, использование в проекте Connect лишь одного человека, но на все 100% его рабочего времени.
- Работу необходимо распределять таким образом, чтобы каждый занимался многозадачной работой понемногу, а не чтобы многозадачной работой занималось помногу лишь некоторые из членов команды. Из рис. 10.6 видно, что наибольшее сокращение количества времени, затрачиваемого на задачу, происходит после того, как кто-то из исполнителей берется за первую задачу слишком часто. В случае Джона более удачным решением было бы сделать все возможное, чтобы двое или трое его АБД участвовали только в каком-то одном проекте, даже если это будет означать, что многозадачная нагрузка на других станет еще большей.

Корпоративная форма многозадачности

Люди чувствуют себя обязанными работать параллельно над несколькими задачами, потому что организации, в которых они работают, также пытаются выполнять одновременно большое количество проектов. Корпоративной формой многозадачности является параллельное выполнение множества проектов. Когда организация берется за одновременное выполнение слишком большого числа проектов, ее сотрудникам приходится участвовать параллельно в нескольких из них, что ведет к высокой степени индивидуальной многозадачности. Ее пагубное влияние выражается в увеличении времени выполнения проектов, что приводит к нарастанию многозадачности ближе к концу проекта, когда уже нужно приступить к выполнению следующего проекта.

Результаты восьмилетнего исследования проектов, выполнявшихся в дюжине компаний, были опубликованы в *Harvard Business Review*. Авторы этого исследования пришли к выводу, что “проекты выполняются быстрее, если организация принимает меры к сокращению количества одновременно выполняемых проектов” (Adler et al., 1996). Корпоративная многозадачность (попытка организации выполнять одновременно как можно больше проектов) — вот что создало ситуацию, в которой оказался Джон из приведенного выше примера и которая заставила его распределить своих АБД по разным проектам вплоть до пятипроцентного уровня.

Мэри (Mary) и Том Попpendик (Tom Poppendieck) призывают организации ограничить объем одновременно выполняемой ими работы, исходя из реальных возможностей, которыми эти организации располагают. Организация, которая старается одновременно выполнять больше проектов, чем в состоянии “потянуть” имеющийся у нее контингент исполнителей, пытается выйти за пределы своих реальных возможностей.

“Если хотите, чтобы ваши команды укладывались в жесткие сроки, *ограничьте объем выполняемой работы реальными возможностями* (2006, 134, курсив авторов).

Исключите однообразный механический труд

Одним из счастливейших дней в моей жизни был день, когда я, работая консультантом, смог объяснить генеральному менеджеру крупного подразделения одной большой компании возможные последствия личной и корпоративной многозадачности. Мне показалось, что мое объяснение нашло горячий отклик в душе этого руководителя. Он попросил меня пройти вместе с ним в конференц-зал, находившийся неподалеку от его кабинета. Указав на огромное число бумажек, приkleенных к самой широкой стене конференц-зала, он сказал: “Мы только что составили план на следующий год. Вот этот план. Не кажется ли вам, что мы несколько перестарались?”

В его подразделении работало свыше ста разработчиков, но стена была буквально испещрена наклейками. Мы поговорили с ним об их плане, количестве одновременно выполняемых проектов, а также о последствиях, которые могли бы иметь место, если бы сроки завершения какого-либо из этих проектов были существенно нарушены. Генеральный менеджер понимал, что они запланировали слишком много, и я был полностью с ним согласен. На следующий день он провел собрание вице-президентов и директоров, принимавших участие в составлении плана, и призвал их сократить количество проектов. Услышав эту неожиданную новость, собравшиеся вздохнули с видимым облегчением. Каждый из них понимал, что составленный ими неделю назад план был чрезмерно амбициозным и вряд ли мог быть выполнен. Однако никто из них не решался сказать об этом вслух.

Через год я встретился с этим генеральным менеджером и обрадовался (хотя и не удивился) тому, что его подразделение буквально на днях завершило самый успешный год в своей истории. Частично это объяснялось переходом к использованию Scrum и улучшениями, которые этот переход принес данному подразделению. Но в не меньшей мере этот успех обусловливался сокращением числа проектов, одновременно выполняемых в этом подразделении, и связанной с этим возможностью уделять больше внимания каждому из оставшихся проектов.

Как следует из этой истории, наилучшим способом положить конец многозадачности зачастую является трезвая оценка собственных возможностей. Однако история с генеральным менеджером произвела на меня впечатление именно потому, что этот руководитель оказался одним из очень немногих, кто имеет смелость выступить вслух. Если вы не можете немедленно положить конец многозадачности или если ваша должность не позволяет принимать решения со столь далеко идущими последствиями, есть кое-что еще, что можно было бы попробовать сделать.

Не начинайте новый проект до тех пор, пока для него не будет полностью укомплектована команда. Избегайте соблазна начать новый проект, если в вашем распоряжении имеется лишь пара-тройка аналитиков и, возможно, один программист. Постарайтесь убедить всех и каждого в том, что новые проекты можно начинать только после того, как они будут полностью укомплектованы специалистами, необходимыми для их успешного выполнения. Это вовсе не значит, что, прежде чем приступить к выполнению крупного проекта, нужно ждать, пока в вашем распоряжении окажутся все 50 разработчиков. Начиная каждый раз новый проект только после того, как будет полностью и надлежащим образом укомплектована по крайней мере одна команда,

мы существенно облегчаем себе задачу приведения во взаимное соответствие темпа инициирования новых проектов и темпа их выполнения.

Не забывайте включать в план предприятия время развертывания и свертывания. Если, подобно генеральному менеджеру в приведенной выше истории, вы составляете большой годовой план, не забудьте предусмотреть в нем время, которое требуется, чтобы начать (развернуть) и завершить (свернуть) любой проект. Слишком часто по оценке команды для выполнения проекта требуется, например, шесть месяцев, и именно эти шесть месяцев предусматриваются в плане предприятия. Однако даже при выполнении Scrum-проекта (особенно если его выполняет новая Scrum-команда) может потребоваться месяц-другой для свертывания проекта. В это время по меньшей мере часть команды может заниматься вылавливанием крупных ошибок или реализацией замечательных новых идей, которые возникли уже после выпуска соответствующего продукта. Если для выполнения подобных действий не будет предусмотрено определенное время, то практически неизбежно возникнут периоды взаимно перекрывающихся проектов.

Вводите простые правила. Введение простых правил помогает добиться надлежащего организационного поведения. Например, такое простое правило, как “никто не может принимать участие более чем в двух проектах одновременно”, способно творить чудеса. Йоханнес Бродуолл (Johannes Brodwall), главный научный сотрудник компаний Steria из Норвегии, предлагает следующее простое правило.

Каждый член команды должен уделять своей команде по меньшей мере 60% своего рабочего времени. Эти 60% производят впечатление некоего “магического числа”, они как бы говорят людям: “Это самое важное, о чем вам следует помнить”. Такая структура заставляет людей относиться с большей ответственностью к своей основной команде.

Продвигайтесь вперед не спеша, но продвигайтесь. Я считаю, что требуется немалая смелость мысли, чтобы быть уверенными в том, что одновременное выполнение меньшего числа проектов позволит в конечном счете выполнить больше проектов. Даже если кто-то считает, что ускоренное выполнение проектов в конце концов приведет к повышению производительности, он чувствует дискомфорт, когда приходится переносить дату завершения крупномасштабного проекта на более поздние сроки или вообще отказываться от его завершения. Именно поэтому нужно начинать с малого: убрать один проект из плана первого квартала и посмотреть, чем все это закончится.

Рекомендации по выбору структуры команды

В этом разделе представлены рекомендации, которых следует придерживаться при выборе структуры команды. Каждая из них представлена в форме вопроса, касающегося существующей или предполагаемой команды. Такие вопросы должны задаваться итеративно. Задавайте каждый из вопросов, касающийся существующей или предполагаемой команды, изменения ее структуру соответственно полученному ответу. По мере изменения структуры задавайте эти же вопросы повторно до тех пор, пока не получите утвердительные ответы на каждый из них.

Сделан ли в данной структуре упор на сильные стороны ее членов, и удалось ли свести на нет их недостатки и подкрепить их мотивации? Люди не будут испытывать

удовлетворения, оказавшись в команде, в которой им не удастся проявить свои сильные стороны и придется заниматься тем, в чем они не очень-то сильны. Добросовестные члены команды готовы делать все необходимое для успешного выполнения проекта, однако это не избавляет нас от поиска такой структуры команды, в которой бы в максимальной степени раскрывались сильные стороны как можно большего числа членов команды.

Минимизирует ли данная структура количество людей, работающих параллельно в двух командах (и позволяет ли она полностью избежать работы своих членов в трех командах)? Хорошо продуманная структура команды для организации, которая не пытается выполнять одновременно слишком много проектов, позволит сократить многозадачность до вполне приемлемого уровня. Если организация не пытается выполнять одновременно слишком много проектов и при этом более 10–20% всех членов команды работают в двух или более командах, нужно придумать другую структуру команды или отказаться от каких-то проектов.

Максимизирует ли данная структура продолжительность времени работы команды в неизменном составе? При прочих равных условиях предпочтение нужно отдать структуре, которая максимизирует продолжительность времени работы команды в неизменном составе. Чтобы “притереться” друг к другу, научиться работать вместе, людям нужно время. Амортизируйте стоимость такого обучения на как можно большем отрезке времени, пытаясь как можно дольше сохранить неизменный состав команды (будет просто замечательно, если вам удастся найти такую структуру команды, которая подойдет для выполнения *нескольких* проектов).

Используются ли компонентные команды только в исключительных и полностью оправданных ситуациях? Большинство команд следует создавать на основе “сквозной” разработки работоспособных функциональных возможностей. Однако в некоторых случаях можно создавать и компонентные команды, разрабатывающие повторно используемые компоненты пользовательского интерфейса, обеспечивающие доступ к базе данных и другие подобные функциональные возможности. Но создание компонентных команд должно быть не правилом, а исключением.

Сможете ли вы накормить большинство команд двумя пиццами? Учитывая высокие производительность и качество, обеспечиваемые небольшими по численности командами, большинство команд должно включать от пяти до девяти членов.

Минимизирует ли данная структура количество коммуникационных путей между командами? Неудачно выбранная структура команды может привести к возникновению огромного количества коммуникационных путей между командами. Чтобы выполнить любую работу, команде сначала придется скоординировать свои действия со многими другими командами. Разумеется, какая-то координация действий между командами требуется в любом случае. Но если команда, которая желает добавить в какую-то форму новое поле, приходится координировать свои действия с действиями трех других команд (а мне приходилось сталкиваться с подобными ситуациями), то коммуникационные издержки следует признать слишком высокими.

Побуждает ли данная структура к взаимодействию команды, которые в противном случае не общались бы между собой? Общение некоторых команд между собой происходит совершенно естественно. Удачно выбранная структура команд способствует взаимодействию между командами или людьми, которым необходимо общаться, но которым иногда нелегко наладить такое общение собственными силами. По сути,

одна из причин, заставляющих включить одного человека сразу в две команды, заключается в том, что такой подход помогает наладить общение между этими командами. Если недостаточное взаимодействие между двумя командами превращается в серьезную проблему, то включение одного человека в состав этих двух команд представляется вполне оправданным.

Не противоречит ли выбранный вариант структуры команды четкому пониманию людьми их обязанностей? Удачно выбранная структура команды подкрепляет концепцию совместной ответственности всех команд за успех проекта в целом, предоставляя в то же время каждой команде четкие рамки ее обязанностей.

Влияет ли каждый из членов команды на выбор ее структуры? На ранних стадиях перехода к использованию Scrum такое влияние может оказаться невозможным. Людям еще не хватает опыта в создании работоспособных, протестированных, готовых к использованию продуктов к концу каждого спринта. Кроме того, кое-кто из сотрудников поначалу может настолько противиться переходу к Scrum, что было бы просто наивно ожидать от них конструктивного участия в обсуждении структуры команды. В таких случаях выбрать первоначальную структуру команды мог бы кто-нибудь из менеджеров, не имеющих никакого отношения к данной команде. Однако при этом не следует забывать, что в конечном счете указанную проблему должна решать сама команда.

Что дальше

Рассматривая в этой главе вопрос о том, почему Scrum-команды должны быть немногочисленными, в качестве аналогии мы использовали способность накормить каждую команду двумя пиццами. Чтобы сделать еще больший акцент на способности команды разрабатывать программные продукты быстро, правильно и эффективно, мы рассмотрели также, должна ли выполняться структуризация команд на основе функциональных возможностей или компонентов. Мы пришли к выводу, что при структурировании нескольких команд предпочтение следует отдавать командам, разрабатывающим определенные функциональные возможности, и пытаться избегать использования компонентных команд, признавая в то же время, что в некоторых случаях использование компонентных команд может быть вполне целесообразным.

Затем мы развенчали миф о том, что самоорганизующаяся команда представляет собой случайную совокупность людей. Членов Scrum-команды, как и любой другой команды, нужно подбирать максимально тщательно и вдумчиво. Мы также пришли к заключению о том, что структурировать команды нужно таким образом, чтобы людям не приходилось участвовать одновременно в работе двух и более команд. Наконец мы изложили девять рекомендаций по структурированию команд.

В следующей главе мы рассмотрим вопрос организации коллективного труда. В частности, мы выясним, что нужно для того, чтобы в ходе спринта члены отдельно взятой команды (которую можно накормить двумя пиццами) могли эффективно работать вместе.

Дополнительная литература

DeMarco, Tom, and Timothy Lister. 1999. *Peopleware: Productive projects and teams.* 2nd ed. Dorset House.

Какие бы добрые слова мы ни пытались сказать в адрес этой книги, их все равно будет недостаточно. Я помню день в 1989 году, когда главный исполнительный директор моей компании сказал мне: “Прочитав в эти выходные книгу Peopleware: Productive projects and teams, я пришел к выводу, что нашу группу разработчиков нужно полностью реорганизовать”. Он выполнил эту реорганизацию — и группа добилась превосходных результатов. В этой книге приведено множество советов о том, как помочь команде сполна реализовать свой потенциал.

Goldberg, Adele, and Kenneth S. Rubin. *Succeeding with objects: Decision frameworks for project management.* Addison-Wesley Professional.

Эта книга появилась еще до того, как начался переход к гибкой методологии разработки. Тем не менее в ней содержится ряд чрезвычайно полезных советов относительно выбора структуры команд. В двух главах описываются разные варианты структуры команд, даются советы по выбору наиболее подходящей структуры команд, а также рассматриваются ситуации, в которых рассказывается о структуризации шести разных команд.

Hackman, J. Richard. 2002. *Leading Teams: Setting the stage for great performances.* Harvard Business School Press.

Исходное положение этой книги заключается в том, что задачей лидера является формирование и поддержка команд, способных к самоуправлению. Она включает превосходную главу (“Enabling Structure”) о том, как структурировать команды.

Глава 11

Организация коллективного труда

Основу любого процесса гибкой методологии разработки составляет коллективный труд. В Agile Manifesto сказано, что мы отдаляем предпочтение “людям и взаимодействиям, а не процессам и инструментам” (Beck et al.). Это означает, что “великолепное программное обеспечение создают великолепные команды”. Уже из самого названия методологии Scrum¹ следует, что команда разработчиков продукта должна действовать во многом так, как команда регбистов, т.е. как действующая как единое целое группа игроков, общая задача которых — доставить мяч в так называемое зачетное поле за воротами соперника (или забить его в Н-образные ворота). Учитывая центральную роль команд в успешном выполнении Scrum-проектов, следует считать вполне естественным появление в этой книге главы под названием “Организация коллективного труда”.

Успехи и неудачи члены Scrum-команд делят между собой поровну. В Scrum-команде не может быть “моей работы” и “вашей работы”; в Scrum-команде может быть только “наша работа”. Для большинства людей — особенно для тех, кто привык работать в конгломерате специалистов, или для тех, у кого выработалась привычка делать только то, что им сказано делать — это принципиально новый способ работы. Команды, которым удалось избавиться от подобных умонастроений, по праву испытывают чувство удовлетворенности своей работой. Однако очень многие из них прекращают самосовершенствоваться в тот самый момент, когда они начинают функционировать как единый организм и оказываются не способными воспользоваться многими преимуществами, которые заключает в себе методология Scrum. Чтобы команда стала по-настоящему высокопроизводительной Scrum-командой, ее члены должны объединить свои усилия в направлении непрерывного обучения и совершенствования.

В последующих разделах будут рассмотрены такие понятия, как коллективная ответственность команды, сотрудничество и роль специалистов в Scrum-команде. Вы также узнаете, как Scrum-команды могут работать эффективно в течение спринта, делая “все время всего понемногу”. Завершается эта глава советами о том, как выйти

¹ “Scrum” в переводе с английского означает “борьба за мяч в регби”. — Примеч. пер.

за базовый уровень функциональности путем воспитания у команды способности и желания к обучению, устранения источников напрасной траты знаний, а также выработки у команды желания постоянно совершенствоваться.

Воспитание чувства коллективной ответственности

Одним из важнейших условий, необходимых для того, чтобы Scrum-команда стала по-настоящему функциональной, является выработка у нее чувства коллективной ответственности. Когда я выступаю в роли преподавателя, мне нравятся вопросы моих слушателей, начинающиеся со слов “Кто отвечает за...?” Какими бы словами ни заканчивался этот вопрос, мой ответ на него всегда один и тот же: “Команда”.

Не желая быть неправильно понятым, я пытаюсь развивать эту тему. Допустим, вопрос звучал так: “Кто ответственен за ведение журнала запросов на выполнение работ?” Я отвечаю в том духе, что, хотя ответственность за это несет вся команда, обсуждать проблемы ведения журнала запросов на выполнение работ я буду с владельцем продукта. Вся команда должна чувствовать свою ответственность за все аспекты продукта. Команда в целом несет ответственность за качество продукта. Команда в целом несет ответственность за “чистоту” кода. Команда в целом несет ответственность за правильное ведение журнала запросов на выполнение работ. И т.д.

Вот что пишут по этому поводу Йон Катценбах (Jon Katzenbach) и Дуглас Смит (Douglas Smith), авторы широко известной книги *The Wisdom of Teams*: “Никакая группа людей не станет командой до тех пор, пока не почувствует своей ответственности как команда” (1993, 60). Да, всегда будут конкретные люди, которые будут чувствовать дополнительную ответственность за те или иные аспекты выполняемого проекта. Но это вовсе не освобождает команду от совместной, коллективной ответственности за продукт в целом и все аспекты его разработки.

Несмотря на то что наличие чистого, грамотно написанного кода может казаться чем-то таким, за что несут ответственность лишь программисты, это вовсе не так. Допустим, тестер замечает, что в одной части приложения появляется, исправляется, а затем вновь появляется несколько ошибок. Это может говорить тестеру о том, что этот конкретный код стало трудно сопровождать. Как именно поступит тестер с этой информацией, зависит от самого тестера и культуры данной команды. Тестер может, например, поделиться своими сомнениями с программистом, который писал этот код, со всей командой (во время ежедневного совещания разработчиков или ретроспективы) или с владельцем продукта. Не так уж важно, какой из вариантов выберет тестер. Важно лишь, чтобы он принял меры из чувства ответственности за обеспечение чистоты кода.

Лайза Криспин (Lisa Crispin) (многоопытный тестер, работающий с использованием гибкой методологии разработки), написавшая в соавторстве с Джанет Грегори (Janet Gregory) книгу *Agile Testing*, вспоминает, как она впервые поняла, что, работая в составе Scrum-команды, она отвечает не только за качество продукта.

Должность, которую я занимала до того, как впервые оказалась в составе команды, использующей гибкую методологию разработки, называлась “ответственный за качество”. Я полагала, что именно я отвечаю за качество. Ко мне поступал

большой объем информации, что позволяло мне принимать обоснованные решения. Таким образом, у меня — и только у меня — были ключи к принятию решений. Наша первая итерация в моей новой ХР-команде показала следующий результат: сервер зависает каждый раз после того, как к тестируемому приложению одновременно обращаются два пользователя. Я была потрясена таким результатом и считала его совершенно неприемлемым. Наши наставники поговорили со мной и объяснили, что за качество отвечаю не я, а наш заказчик. Этот заказчик работал в одной начинающей компании и хотел продемонстрировать своим потенциальным клиентам всевозможные “рюшечки и финтифлюшечки”. Он даже не рассматривал возможность одновременного обращения к приложению двух пользователей (эта проблема не имела для него никакого значения), поэтому программисты написали код, в котором такая возможность не была предусмотрена. Этот случай произвел настоящий переворот в моем сознании.

ВОЗРАЖЕНИЕ

“Коллективная ответственность — это синоним безответственности. За все, что должно быть сделано, нужно устанавливать персональную ответственность”.

С точки зрения руководителя, просто здорово, если всегда находится конкретный человек, указав на которого, можно сказать: “Вот с кого я спрошу в случае неудачи”. Но аргумент типа “за все, что должно быть сделано, нужно устанавливать персональную ответственность” звучит убедительно лишь на первый взгляд. Разумеется, всегда может быть назначен тот, кому придется отвечать за все неприятности, если таковые будут иметь место, но это вовсе не означает, что именно он явился причиной этих неприятностей. Возьмем, к примеру, спортивную команду. Кого мы назначим в начале сезона ответственным за выигрыш чемпионского звания? Тренера? Владельца клуба? Ведущего игрока команды? Команды, которые становятся чемпионами, находят способ побеждать своих соперников независимо от обстоятельств. Если план на игру не приносит нужного результата, тренер и игроки пытаются его откорректировать. Если ведущий игрок команды встал сегодня не с той ноги, кто-то из других игроков должен взять на себя его роль. В случае победы вся команда — так или иначе, в той или иной степени — чувствует себя причастной к этой победе. В случае поражения нередко возникает соблазн обвинить во всех неприятностях кого-то одного, но команда-то знает, что виноват в этом поражении каждый из них. Никогда не бывает так, чтобы виноват в поражении был лишь один человек. В действительности вина за поражение распределяется между всеми членами команды.

Рассмотрим аналогию из области воспитания детей. Если воспитанием ребенка занимались оба родителя (и если мы исходим из того, что они не склонны к насилиственным методам воспитания и не относятся к своим родительским обязанностям заведомо небрежно), то кого из них нужно назначить виноватым в случае, если их чадо впоследствии станет преступником? Именно поэтому нужно говорить о родительском коллективе. Воспитание ребенка — коллективный труд.

Единственный способ сформировать атмосферу коллективной ответственности (и коллективного владения) — это отказаться от принципа обязательной персональной ответственности. Однако этот отказ не означает, что мы стремимся к безответственности. Это лишь означает, что каждый из членов успешной команды должен исполнять свою роль (а если потребуют интересы команды, то и выходить за рамки этой роли), обеспечивая таким образом достижение командой поставленных перед ней целей.

“Но результаты моей ежегодной аттестации зависят от того, что сделано лично мною, а не командой в целом”.

Действительно, так оно, наверное, и есть, и если мы ничего не изменим в этом отношении, то это, безусловно, помешает успешному внедрению Scrum в вашей организации. Нам нет нужды полностью отказываться от оценки индивидуального вклада каждого работника, но периодические аттестации должны включать весомый компонент, отражающий достижение командных целей. Более подробно эта тема обсуждается в главе 20, “Кадры, техническое обеспечение и отдел управления проектами”.

Воспитание у команды чувства приверженности поставленной перед ней цели

Наряду с воспитанием чувства коллективной ответственности, у команды нужно вырабатывать чувство приверженности цели, поставленной перед ней. Едва ли не самое плохое, что мне пришлось услышать в конце спринта от одного программиста, когда владелец продукта выразил сожаление по поводу того, что некоторые из запросов на выполнение работ спринта остались нереализованными, выражалось примерно такими словами: “Лично я уже справился со своими задачами”. Возможно, этот программист действительно справился со своими задачами, но его задачи были лишь частью работы, которую обязалась выполнить его команда, и лишь малой частью работы, требующейся для завершения работы над продуктом.

См. также В результате смещения акцента с индивидуальной ответственности на коллективную членам команды все чаще приходится выполнять работу, не связанную напрямую с их специализацией. О переменах в повседневной работе разных исполнителей рассказывается в главе 8, “Изменившиеся роли”.

В ходе планирования спринта команда планирует работу предстоящего спринта. Я не рекомендовал бы, чтобы это планирование предполагало выбор задач исполнителями (“я сделаю то, вы сделаете это...”). Подобное раннее распределение работ характерно для команд, лишь приступающих к освоению Scrum. В ходе этих ранних спринтов я напоминаю командам, наставником которых я являюсь, что к таким распределениям нужно относиться довольно осторожно. Цель команды заключается в том, чтобы завершить все работы, указанные в журнале запросов на выполнение работ для соответствующего спринта. К этому нужно относиться как к обязательству команды в целом, а не обязательству каждого из членов команды выполнить порученные ему задачи.

В некоторых организациях будет не так-то легко перейти от культуры, сущность которой можно сформулировать словами “Я отвечаю за порученные мне задачи и обязан их выполнить”, к культуре совместной, коллективной ответственности за достижение цели, поставленной перед командой. Однако до тех пор, пока такой переход не завершится, команде будет трудно реализовать все запросы на выполнение работ, запланированных на тот или иной спринт. В условиях, когда вся команда демонстрирует приверженность цели, поставленной перед ней, член команды, работающий с

опережением графика, приходит на помощь другому члену команды, который, в силу тех или иных причин, отстает от графика, в результате чего команда в целом точно укладывается в график выполнения работ, запланированных для данного спрингта. Если же такая приверженность общекомандной цели отсутствует, то можно почти не сомневаться, что к концу спрингта многие из запросов на выполнение работ, запланированных для данного спрингта, будут “выполнены на 90%”, поскольку те, кто справились со своей работой, будут просто сидеть и ждать, пока работник, в силу каких-то причин не укладывающийся в график, будет стараться наверстать отставание, но так и не выполнит свою работу до конца.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- В ходе очередного совещания, посвященного планированию спрингта, не требуйте от членов команды выбрать задачи, которые им предстоит выполнить. Вместо этого определите, кто из них будет, вероятнее всего, работать над теми или иными задачами. Главное — чтобы все члены команды пришли к согласию по поводу конкретного списка работ, включенных в журнал запросов на выполнение работ для планируемого спрингта. Иначе говоря, все члены команды должны быть согласны в том, что эти работы могут быть выполнены в ходе рассматриваемого спрингта. Только не нужно рядом с названиями задач указывать фамилии конкретных исполнителей: распределение задач должно произойти спонтанно в ходе спрингта. После спрингта можно обсудить, как именно все это произошло.
- Наваливайтесь всей командой на каждую из работ, запланированных для данного спрингта. Полностью завершив эту работу, все вместе переходите к выполнению следующей работы и так до тех пор, пока не будут выполнены все запланированные работы. Это предоставит членам команды великолепную возможность научиться работать вместе (однако я не могу рекомендовать такой вариант в качестве постоянно используемого метода работы; скорее его можно использовать лишь в исключительных случаях).

Полагайтесь на специалистов, не злоупотребляя ими

Типичным заблуждением является мнение о том, что каждый из членов команды должен быть не узким специалистом, а универсалом, одинаково хорошо разбирающимся во всех технологиях. Это вовсе не так. Самое удивительное для меня в этом заблуждении то, что каждая закусочная в мире уже давно научилась пользоваться услугами специалистов, тогда как мы, разработчики программного обеспечения, все еще продолжаем дискутировать по этому поводу.

Моей любимой закусочной является “Beach Hut Deli” в городе Фолсом, штат Флорида. Я заметил, что в этом заведении трудятся три категории работников: приемщики заказов, повара (которые готовят сэндвичи) и подсобные работники. Приемщики заказов работают у кассовых аппаратов, записывают заказ на листке бумаги и передают этот листок поварам. Повара готовят блюда согласно полученным заказам. Приемщики заказов и повара являются в закусочной “Beach Hut Deli” своего

рода специалистами, а подсобные работники — своего рода универсалами. Вообще говоря, они могут выступать и в роли приемщиков заказов, и в роли поваров, хотя, возможно, они не так искусны в этих профессиях, как приемщики заказов и повара. Нет, я не хочу сказать, что сэндвич, приготовленный подсобным работником, хуже сэндвича, приготовленного профессиональным поваром. Просто подсобный работник действует несколько медленнее профессионала. Я сам, будучи подростком, работал подсобным работником в одном из ресторанов фаст-фуд. Конечно, я не так ловко и быстро готовил хот-доги, как один из поваров этого ресторана, Марк. Каждый раз, когда нужно было заправить в кассовый аппарат новый рулон бумаги, я звал на помощь своего менеджера, Никки, поскольку никак не мог запомнить, как это делается. Но в отличие от Марка и Никки я мог и принимать заказы, и готовить хот-доги.

Я подозреваю, что практически в каждой закусочной мира работают специалисты, т.е. люди, которые либо готовят блюда, либо принимают заказы. Но владельцы этого бизнеса уже давно поняли важность и такой категории работников, как универсалы. Наличие среди работников закусочной какого-то количества универсалов (работающих, как правило, неполный рабочий день) помогает справляться с повышенной нагрузкой, возникающей в часы пик, и гибко реагировать на быстрые изменения потребности в приемщиках заказов и поварах.

Что это означает для Scrum-команд? Это означает, что в составе такой команды всегда должно быть несколько универсалов. Именно наличие универсалов позволяет специалистам выполнять свои узкоспециальные задачи. Всегда найдутся команды, которым требуется программист, умеющий разрабатывать драйверы для тех или иных устройств, программист на C++, хорошо разбирающийся в тонкостях Windows, программист искусственного интеллекта, инженер, умеющий тестировать быстродействие систем, специалист по биоинформатике, художник и т.д. Но включение в команду специалиста каждый раз следует рассматривать как некий эквивалент приема на работу в закусочную нового повара. Включая в состав команды слишком много специалистов, вы повышаете вероятность того, что кто-то из них будет тратить слишком много времени на ожидание получения работы от других членов команды (этую проблему мы рассмотрим подробнее в следующем разделе).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- В ходе очередного совещания, посвященного планированию спринта, примите решение, что в течение данного спринта кто-то из специалистов в вашей команде не будет работать по своей узкой специальности. Этот специалист может консультировать других членов команды, тех, кто будет выполнять эту специализированную работу, но не может выполнить ее самостоятельно. Цель эксперимента заключается не столько в том, чтобы расширить арсенал возможностей данного специалиста, сколько в том, чтобы выработать соответствующие навыки у других членов команды. Обсудите результаты этого эксперимента во время ретроспективы спринта. Рассмотрите возможность повторения этого эксперимента с тем же специалистом. Рассмотрите возможность повторения этого эксперимента с каким-либо другим специалистом.

Старайтесь понемногу выполнять разные работы

Команды, привыкшие к последовательному процессу разработки, привыкли и к тому, что выполненная работа передается от одного специалиста к другому. Аналитики передают выполненную ими работу проектировщикам, которые передают выполненную ими работу программистам, которые передают свою работу тестерам... Каждая из таких передач из рук в руки включает определенные накладные расходы в форме совещаний, представления документов на прочтение (и, возможно, на подпись) и т.д. Частично именно из-за этих накладных расходов передачам работы из рук в руки присущи, как правило, большие объемы функциональности. В соответствии с концепцией процесса разработки по методу “водопада”, все приложение передается от одной группы к другой.

Компании, являющиеся новичками в деле использования Scrum, зачастую прилагают не слишком большие усилия, чтобы избавиться от этих передач работы из рук в руки. Нередко они исходят из предположения, что программисты должны закончить программирование задачи, предусмотренной в журнале запросов на выполнение работ, и только после этого передать ее тестерам. Это приводит к продолжительным задержкам в самом начале спринта, когда тестерам приходится ожидать, пока первая работа, предусмотренная в журнале запросов на выполнение работ, будет передана им для тестирования. При выполнении Scrum-проекта порция работы, передаваемая между специалистами разного профиля, должна быть меньшей, чем отдельно взятый элемент журнала запросов на выполнение работ. Иначе говоря, несмотря на то, что какие-то передачи работы из рук в руки все же станутся (невозможно, чтобы каждый постоянно выполнял все виды работ), объем работы, передаваемой от одного исполнителя к другому, должен быть, вообще говоря, как можно меньшим.

Допустим, что некая команда разрабатывает какое-то новое приложение для электронной коммерции. Команда принимает решение работать над следующей пользовательской историей: *“Как покупатель я хотел бы выбирать способы доставки купленных мною товаров, основываясь на фактической стоимости доставки товаров на мой адрес, что дало бы мне возможность принять оптимальное решение”*. Результатом должно стать обсуждение между всеми заинтересованными лицами и будущими участниками разработки данной функциональной возможности. Допустим, что в этом обсуждении участвуют владелец продукта, бизнес-аналитик, тестер и программист. Предметом первоначальных обсуждений являются общие требования, касающиеся этой функциональной возможности, например какие транспортные компании мы предпочитаем (FedEx? DHL? Или другую?), Хотим ли мы обеспечить доставку в течение суток? Или в течение двух дней? В течение трех дней? И т.д.

В ходе этих обсуждений их участники, естественно, будут задумываться о том, с чего начать. В случае проекта, выполняемого традиционным способом, каждый может начать работать, как он сам пожелает (после того, как ему будет передана работа). Однако в случае Scrum-команды то, “как начать работу”, должны совместно обсуждать все, кто будут работать над реализацией данной функциональной возможности. Допустим, применительно к нашему примеру, что программист высказывает мнение, что, в силу каких-то причин будет легче начать с FedEx. Тестер соглашается с этим мнением. Аналитик высказывает намерение рассмотреть вариант с использованием

DHL и получить дополнительные сведения о параметрах, которые влияют на стоимость доставки посредством DHL. Аналитик намерен раздобыть эту информацию к тому моменту, когда программист и тестер закончат с FedEx.

Когда программист будет располагать информацией, достаточной для того, чтобы приступить к кодированию, он приступит к кодированию. Владелец продукта, аналитик и тестер обсуждают тесты верхнего уровня (будет ли наш сайт обеспечивать доставку каких-либо крупногабаритных товаров, таких как лыжи?). После этого обсуждения тестер перерабатывает список тестов верхнего уровня в конкретные тесты (ящики этого размера и веса, отправляемые в такой-то пункт назначения). Тестер создает тестовые данные и автоматизирует тесты. Определенная степень автоматизации может быть обеспечена без каких-либо промежуточных наработок со стороны программиста. Полная автоматизация может потребовать получения от программиста какой-либо предварительной версии. В то время как тестер размышляет над конкретными тестами, он должен проинформировать программиста о любых контрольных примерах, которые программист, возможно, не учитывал во время программирования. Когда программист и тестер завершат свою работу, они добавят в сборку способ вычисления стоимости доставки посредством FedEx, дополнив ее автоматизированными тестами. В графическом виде это можно представить так, как показано на рис. 11.1.

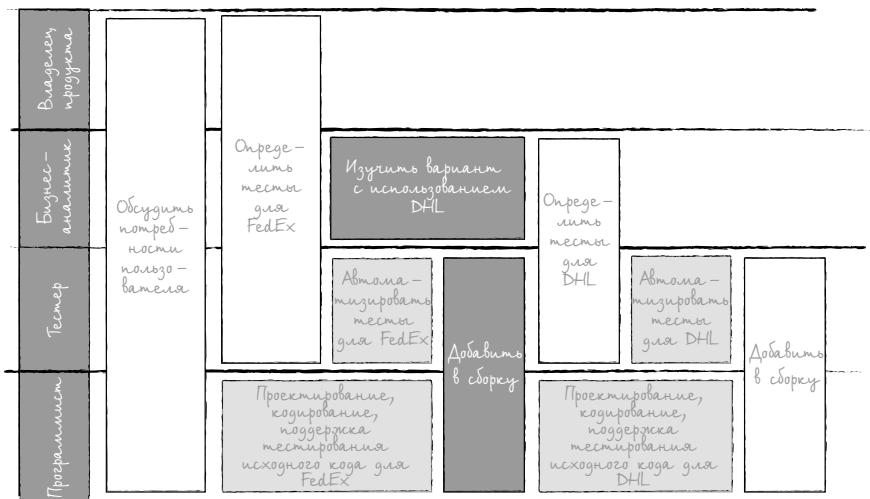


Рис. 11.1. Эти четыре сотрудника работают в тесном контакте друг с другом над одним и тем же элементом журнала запросов на выполнение работы, а не передают последовательно друг другу выполненную ими работу

Затем программист и тестер консультируются с бизнес-аналитиком, который, в принципе, должен располагать большими знаниями о вычислении стоимости доставки посредством DHL. Указанный процесс повторяется, и поддержка вычислений стоимости доставки посредством DHL добавляется в приложение, после того как будет завершено программирование и тестирование. Ключевым элементом на рис. 11.1 является то, что команда научилась работать, выполняя все время понемногу ту, то другую работу. Вместо того чтобы на смену фазе анализа (выполняемой без участия программиста и тестера) приходила фаза программирования, на смену которой,

в свою очередь, приходила фаза тестирования, постоянно выполняются небольшие порции каждого из этих видов профессиональной деятельности.

Чтобы завершить все работы, не нужно ожидать окончания спрингта

Симптомом продолжения передачи работы чрезмерно большими порциями будет тенденция к завершению всех работ, предусмотренных в журнале запросов на выполнение работ, лишь в последние дни спрингта. Тестеры в командах, склонных работать именно таким образом, нередко жалуются, что вся работа по тестированию наваливается на них за два-три дня до завершения спрингта и им приходится работать буквально круглосуточно, чтобы успеть протестировать эти огромные объемы исходных кодов к окончанию спрингта. Наилучшим способом продемонстрировать всем членам команды эту проблему является создание диаграммы, на которой отображалось бы количество элементов журнала запросов на выполнение работ, завершенных к окончанию каждого дня спрингта. Пример такой диаграммы показан на рис. 11.2, а.

Выступая в роли Scrum-мастера той или иной команды, я часто вывешиваю такую диаграмму в комнате, где работает команда, без каких-либо разъяснений или громких заявлений. Члены команды вскоре выявляют проблему, демонстрируемую такой диаграммой, и начинают изыскивать способы, позволяющие ускорить завершение тех или иных элементов журнала запросов на выполнение работ. Результат таких усилий зачастую похож на диаграмму, показанную на рис. 11.2, б и отображающую гораздо более рациональный график выполнения работ в течение спрингта.

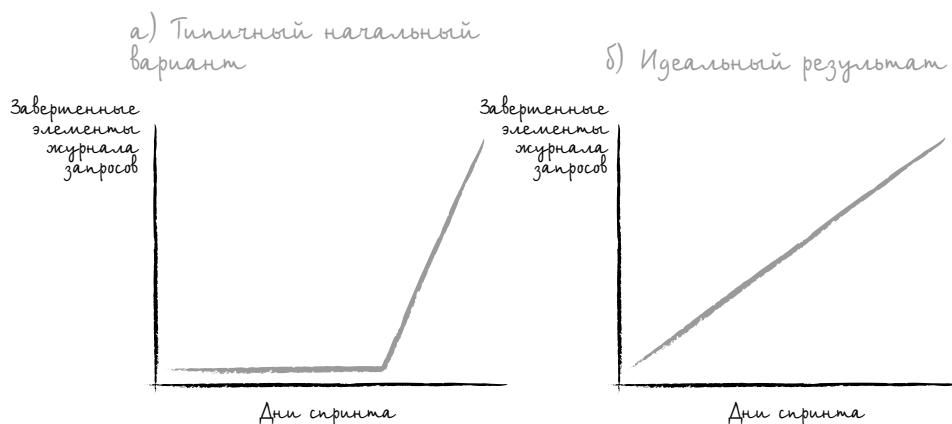


Рис. 11.2. Отображение количества элементов журнала запросов на выполнение работ, завершенных к окончанию каждого дня спрингта, может продемонстрировать проблему, заключающуюся в передаче исполнителям очень больших порций работы

Балансируйте размеры элементов журнала запросов

Планируя очередной спрингт, обращайте внимание на величину элементов журнала запросов на выполнение работ, которые вы обязуетесь завершить к окончанию спрингта. Некоторые из элементов журнала запросов на выполнение работ могут быть более сложными, чем пример FedEx/DHL, приведенный выше в этом разделе.

Для реализации каких-то элементов журнала запросов на выполнение работ программисту может потребоваться примерно неделя (только после этого он сможет передать тестеру нечто такое, что можно хотя бы в первом приближении рассматривать как программный продукт, подлежащий тестированию). Это нормально: далеко не все можно разделить на столь мелкие порции, работать с которыми нам было бы максимально комфортно.

Разумеется, нужно избегать появления в одном и том же спринте нескольких элементов, подобных этому, — в противном случае основная доля работы, связанной с тестированием, придется на конец спринта. Вместо того чтобы включать в спринт, например, три очень крупных элемента, которые невозможно реализовать частично, включите в спринт один или два крупных элемента, а также два или три более мелких элемента. Некоторые из программистов могут работать с крупными элементами, передавая их тестерам по мере готовности. Остальные программисты могут работать с меньшими элементами, загружая работой тестеров уже в начале спринта и обеспечивая таким образом более равномерный поток работы, направляемой тестерам.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Обязуйтесь завершить к середине очередного спринта треть запланированных элементов журнала запросов на выполнение работ.
- Выставьте для всеобщего обозрения диаграмму, подобную той, которая показана на рис. 11.2.
- В течение следующих трех спринтов с самого начала предлагайте программистам и тестерам найти подходящую среднюю точку каждого элемента журнала запросов на выполнение работ. Обязуйтесь добавить ее в ежедневную сборку, как только эта средняя точка будет достигнута (а не после того, как будет готов соответствующий элемент в целом).

Способствуйте обучению команды

Если ваша команда усвоила концепцию приверженности всей команды, снизила степень своего упования на специалистов и постоянно выполняет понемногу то ту, то другую работу, значит, можно утверждать, что вы уже достигли немалых успехов в деле совершенствования своего коллективного труда. Именно на этой стадии большинство команд успокаивается на достигнутом. Этого ни в коем случае нельзя допускать. У вас есть еще немало возможностей для совершенствования. Чтобы стать по-настоящему высокоэффективной командой и реализовать все выгоды, которые заключает в себе Scrum, ваша команда должна в упреждающем режиме изыскивать все новые и новые способы обучения и совместного использования знаний.

Обучение частично происходит естественным способом: пользователь рассказывает владельцу продукта, что ему нравится и как реализована та или иная функциональная возможность, программист обнаруживает, что потребности, связанные с дальнейшим наращиванием системы, невозможно удовлетворить путем использования определенной технологии и т.д. Какие-то виды обучения работают лишь после

принятия целенаправленных мер. Именно о таких видах обучения я хочу сейчас поговорить. Вместо того чтобы пассивно ожидать, что обучение произойдет само собой, самые эффективные команды и их лидеры играют весьма активную роль в деле оптимизации темпов и содержания обучения.

Позаботьтесь о создании условий для обучения

Как уже было сказано, одной из целей выполнения проекта является командное обучение, которое достигается путем принятия целенаправленных мер. Существует пять условий, без которых командное обучение невозможно.

- Состав команды должен быть таким, чтобы члены команды имели возможность обучаться.
- У членов команды должны быть конкретные способы совместного использования знаний.
- Лидеры должны на личном примере подчеркивать важность обучения.
- Для команд должны быть созданы соответствующие мотивации к обучению.
- В команде необходимо создать атмосферу, способствующую обучению.

Каждое из этих условий обсуждается в следующих разделах.

Состав команды должен быть таким, чтобы члены команды имели возможность обучаться

Как указывалось в предыдущей главе, руководители, а также другие лица, заинтересованные в выполнении проекта, зачастую могут оказывать существенное влияние на состав команды. Они должны использовать эту возможность для формирования команд из таких исполнителей, которые, объединив свои усилия, смогут добиться большего, чем сумма составных частей команды. Состав команды должен быть достаточно разнородным, чтобы в таком коллективе могли возникать оригинальные, творческие идеи, однако эти различия не должны быть настолько глубокими, чтобы не перерасти в непреодолимые противоречия между членами команды.

См. также Из главы 12, "Управление самоорганизующимся коллективом", вы узнаете, как руководителям и другим лидерам следует использовать свое влияние и ответственность.

Самое лучшее, что может сделать менеджер для только что созданной команды, — это оставить ее в неизменном виде на как можно более длительный срок. Членам любой команды нужно время, чтобы "притереться" друг к другу, научиться работать вместе; постоянные перетасовки в команде заставляют ее начинать этот процесс с самого начала каждый раз при появлении нового члена команды или при уходе кого-либо из команды. Ричард Хэкман (Richard Hackman), профессор Гарвардского университета и авторитетный специалист по вопросам коллективного труда, цитирует исследование, которое указывает, что "команды проектировщиков действительно нуждаются в притоке свежих сил, новых талантов; это нужно для поддержания в команде атмосферы творчества и новизны, однако темп таких обновлений не должен быть больше, чем один человек каждые три-четыре года" (Hackman and Coutu, 2009).

У членов команды должны быть конкретные способы совместного использования знаний

Лью Плatt (Lew Platt), бывший главный исполнительный директор Hewlett-Packard, однажды сказал: “Если бы Hewlett-Packard знала, что знает Hewlett-Packard, ее прибыль возросла бы в три раза”. Чтобы компании были успешными, у членов их команд должны быть конкретные способы делиться приобретенными ими знаниями не только друг с другом, но и с остальными сотрудниками организации. Одним из способов, посредством которых Scrum-команды пытаются осуществлять указанный обмен знаниями, являются многочисленные коммуникационные Scrum-форумы. Ежедневные совещания разработчиков способствуют обмену информацией между членами одной команды и, возможно, другими лицами, участвующими в таких совещаниях. Обзоры спринтов, как правило, способствуют дальнейшему распространению знаний, особенно если в этих обзорах участвуют другие лица, заинтересованные в реализации соответствующего проекта, а также члены других команд. А в крупных организациях совещания разработчиков, представляющие разные команды, позволяют командам обмениваться информацией с представителями всех остальных команд организации. Кроме того, Scrum-команды располагают инструментами, способствующими обмену знаниями. Вики-доски и крупноформатные диаграммы позволяют буквально с одного взгляда оценить текущее состояние соответствующего спринта и проекта не только членам команды, но и всем, кто имеет возможность взглянуть на них.

Помимо этих коммуникационных инструментов, высокоэффективные Scrum-команды изыскивают способы напрямую пообщаться с представителями других команд. Например, разработчики баз данных охотно общаются с другими группами разработчиков баз данных, а разработчики пользовательского интерфейса активно ищут способы общения с другими группами разработчиков пользовательских интерфейсов. Часто такое общение носит исключительно неформальный и спонтанный характер (впрочем, нередко общение бывает вполне формальным и заранее спланированным). Крупные Scrum-проекты и подразделения зачастую организовывают сообщества практических пользователей Scrum. В рамках таких сообществ группы единомышленников или людей одной профессии, имеющих примерно одинаковую квалификацию, могут встречаться регулярно, чтобы поделиться друг с другом не только типичными проблемами, но и возможными их решениями. Такие сообщества практических пользователей Scrum предоставляют своим членам замечательную возможность делиться знаниями и практическим опытом.

См. также Подробное описание сообществ практических пользователей Scrum приведено в главе 17, “Изменение масштаба Scrum”.

Лидеры должны на личном примере подчеркивать важность обучения

Члены команды взаимодействуют между собой, ориентируясь на поведение тех, кого они считают лидерами своей команды или организации. Роль таких лидеров могут исполнять владельцы продукта, функциональные менеджеры, которым подчиняются члены команды, а также другие руководители организации. Таким образом,

чтобы воспитывать у людей правильные модели поведения, лидеры команды и организации должны демонстрировать тот тип поведения (в том, что касается обучения), который они хотят видеть у членов своих команд.

Например, недавно я присутствовал на собрании, на котором команда и владелец продукта этой команды, Майкл, знакомили членов исполнительного комитета с идеей нового продукта. Один из членов исполнительного комитета, вице-президент по имени Шон, с особым знанием дела расспрашивал о новом продукте Майкла и разработчиков. Он задавал им сложные вопросы, цель которых заключалась в том, чтобы помочь команде (и другим членам исполнительного комитета) выявить слабые места в представленном плане разработки этого нового продукта. Шон вовсе не стремился к тому, чтобы “подловить” Майкла, заставить его изворачиваться или вообще отказаться от своей идеи. Цель его вопросов (“Можете ли вы указать три причины, которые заставили бы потенциального клиента купить именно наш продукт, а не продукт конкурирующей фирмы?”, “Окажутся ли эти причины достаточными?”) заключалась в том, чтобы инициировать диалог, в котором он должен был выступать в роли активного участника. Поскольку Шон — один из высших руководителей компании — искренне хотел узнать в результате этого диалога что-то новое для себя, его поведение служило примером для остальных участников этого собрания, заставляя их демонстрировать такое же, как у Шона, желание узнать что-то новое.

Помимо постановки вопросов, которые ведут к искреннему диалогу, сфокусированному на обучении, Эдмондсон (Edmondson), Бомер (Bohmer) и Пизано (Pisano) указывают три иные модели поведения, которые должны демонстрировать лидеры, желающие подчеркнуть важность обучения.

- **Будьте доступны для своих подчиненных.** Лидеры должны быть доступны для членов команд, а не отгораживаться от них закрытыми дверями и поселением на этажах, предназначенных исключительно для высшего руководства компании.
- **Предлагайте высказаться каждому из членов команды.** Предлагая высказаться каждому члену команды, вы четко даете им понять, что руководству важно знать их мнение. Предлагая им поучаствовать в принятии важных решений, вы тем самым подталкиваете их поступать точно так же по отношению друг к другу. Предлагая высказаться каждому из членов команды, обязательно про демонстрируйте впоследствии, что их мнения были учтены (а если какие-то из мнений не были учтены, обязательно укажите причины).
- **Выступайте в роли человека, способного признавать собственные ошибки.** Открыто признавая свои ошибки, вы демонстрируете другим, что просчеты, неправильные решения и проблемы можно обсуждать, не опасаясь пагубных последствий (2001).

Для команд должны быть созданы соответствующие мотивации к обучению

Способ, которым та или иная проблема представляется команде, влияет на то, как члены команды будут на нее реагировать. Вообразите владельца продукта, которому нужно, чтобы определенная совокупность функциональных возможностей была реализована к сроку, который представляется членам команды совершенно невозможным. Владелец продукта мог бы сообщить об этом команде, как о неком

совершившемся факте: “Мне нужно, чтобы эта совокупность функциональных возможностей была реализована к такому-то сроку. Любые другие варианты исключаются. Приступайте к исполнению”.

Курт, владелец продукта, представил такую проблему своей южнокалифорнийской команде совсем по-другому. Сначала Курт открыто признал огромную сложность задачи, которую он намерен поставить перед своей командой. Затем он сообщил, что именно нужно сделать и к какому сроку. Не прибегая к фальшивому тону обреченности или к угрозам (на тот случай, если команда не справится с поставленной задачей), он разъяснил людям, насколько важно будет для них добиться поставленной цели. В завершение своего выступления он еще раз подчеркнул важность каждого из исполнителей для достижения поставленной цели. Спустя пять месяцев команда обеспечила функциональность, достаточную, чтобы избежать первоначального кризиса, что позволило им поработать спокойно еще шесть месяцев для создания окончательной версии.

В первом случае владелец продукта просто поставил свою команду перед фактом. В втором случае Курт признал сложность задачи, которую предстоит решить его команде, но высказал убеждение в способности команды справиться с поставленной задачей. Затем он вместе с командой подыскал подходящий начальный вариант, достаточный для того, чтобы крупнейший клиент их компании почувствовал себя в достаточной мере удовлетворенным, и в то же время вполне посильный для команды. Это помогло создать в команде позитивную атмосферу по отношению к данному проекту, что способствовало налаживанию диалога и обсуждения, необходимых для эффективного обучения.

В команде необходимо создать атмосферу, способствующую обучению

Вспоминаю не самые приятные впечатления, которые возникли у меня, когда я впервые привел в подготовительный класс свою дочь, Делани. В помещении, которое на ближайший год должно было стать для моей дочери вторым домом, царил полный беспорядок. Вместо одной или двух больших книжных полок, на которых должны были стоять все учебники, по стенам было развешано множество небольших книжных полок. Вместо стройных рядов парт были беспорядочно расставлены столы, стулья, диваны с подушками (очевидно, для отдыха детей) и корзины для мусора. Свободные места на стенах были сплошь заклеены и увешаны плакатами, большими вырезками из газет, географическими картами и т.п. Моя жена объяснила, что эта хаотическая, на мой взгляд, обстановка в помещении на самом деле является пространством, идеально приспособленным для обучения детей в возрасте от четырех до пяти лет, к числу которых тогда принадлежала и наша Делани.

Точно так задачей лидеров и менеджеров в организациях, внедряющих у себя Scrum, является создание обстановки и атмосферы, способствующей обучению их команд. В то время как создание атмосферы, благоприятной для обучения маленьких детей, предполагает главным образом формирование в помещении соответствующей физической обстановки (нужно, например, позаботиться о том, чтобы ребенок мог без труда достать нужную ему книгу), создание обстановки и атмосферы, благоприятной для обучения Scrum-команд, предполагает внесение определенных организационных, социальных и психологических изменений. Организации, создавшие у себя благоприятную среду для обучения, отличаются следующими характеристиками.

- **Психологическая защищенность.** Один из лучших способов научиться чему-то — попытаться сделать это самому, совершив ошибку, а затем сделать то же самое еще раз, но уже лучше. Другими способами обучения являются постановка вопросов и участие в обсуждениях. Если это вызывает у кого-то ощущение дискомфорта и незащищенности, то ничего такого они делать не будут. Владельцы продукта, Scrum-мастера, функциональные менеджеры и другие лидеры должны найти способы создания вокруг этих видов деятельности ощущения защищенности; в противном случае члены команды не рискнут делать что-то новое из опасения потерпеть неудачу, глупо выглядеть в глазах окружающих или испытывать другие неприятные ощущения. Создание у людей ощущения психологической защищенности особенно важно при переходе к Scrum по той причине, что в этом случае людям приходится приобретать новый опыт, работать над повышением своей квалификации, переоценивать свое место в коллективе. Весьма вероятно, что кто-то привык считать себя крупным специалистом в той или иной области. Переход к использованию Scrum нивелирует имеющийся опыт многих людей, заставляя их приобретать новые знания, опыт и навыки. Лидеры (а по сути — все члены команды) должны создать ощущение психологической защищенности, чтобы, например, специалист по многопотоковому Java-программированию не стеснялся задавать элементарные (а часто просто глупые) вопросы об автоматизированном тестировании исходного кода. Неспособность лидеров создать у людей ощущение психологической защищенности обычно приводит к тому, что сотрудники начинают противиться переменам.
- **Понимание ценности различий между людьми.** Члены команды должны ценить различия, а не бояться их. Если бы все, например, обладали одинаковым практическим опытом и одинаковой квалификацией, пользовались одними и теми же подходами к решению проблем, то результатом явилось бы отсутствие в команде творческого мышления. Вот что говорит по этому поводу профессор Гарвардского университета и автор книги *Leading Teams* Ричард Хэкман (Richard Hackman): “Каждой команде нужен человек с отклонениями от нормы — человек, способный помочь команде, бросив вызов склонности людей к чрезмерной однородности и одинаковости. Такая склонность может подавлять в людях творческие порывы и желание обучаться чему-то новому. Люди с отклонениями от нормы — это те, кто, взглянув на ситуацию со стороны, спрашивают: «Минуточку, а почему, собственно говоря, мы вообще это делаем? А что если посмотреть на это под другим углом зрения или вообще вывернуть наизнанку?»” (Hackman and Coutu, 2009).
- **Открытость для новых идей.** Перед Scrum-командами нередко ставят очень сложные задачи: разработать приложение быстрее, чем выполнялись аналогичные проекты в прошлом, выполнить проект с привлечением меньших ресурсов и т.п. Чтобы решить их, членам команды зачастую приходится отказываться от давно наработанных схем и привычных и испытанных методов работы. Жизненно важной в таких случаях становится открытость команды для новых идей. Разумеется, при этом нужно быть готовым к ошибкам, временными неудачам и некоторому регрессу.
- **Время на раздумья.** Понятно, что Scrum-командам приходится действовать в высоком темпе итеративной разработки. Однако командам требуется время,

в течение которого они могли бы спокойно поразмышлять над тем, что и как они делают. Обучение в реальном времени по ходу выполнения практической работы — вот наилучший способ для команды научиться чему-то новому. Такому обучению во многом способствуют ежедневные совещания разработчиков. Однако большинство команд находит весьма полезным на протяжении каждого спринта тратить от получаса до половины рабочего дня на поиск способов совершенствования своей работы.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если в вашей команде есть специалисты по тем или иным аспектам традиционного способа работы, которые опасаются перехода к новым техническим приемам, предусмотренным методологией Scrum, то лучше всего в такой ситуации привлечь в команду наставника со стороны. Ведущий разработчик, пытающийся разобраться, например, в тонкостях создания программного обеспечения на основе составления тестов, нередко ощущает меньшую опасность, когда получает эти новые для себя знания от постороннего человека, а не от кого-то из членов команды, в отношении кого этот ведущий разработчик должен был выступать в роли наставника.

Избегайте потерь знаний

Наряду с формированием атмосферы, благоприятной для командного обучения, нужно пытаться устраниć организационные недостатки, которые приводят к неэффективному использованию приобретенных командой знаний. Под неэффективным использованием знаний мы подразумеваем либо упущеные благоприятные возможности, связанные с обучением, либо обучение, меньшее, чем то, которое могло бы быть реально достигнуто в конкретной ситуации. Специалист по экономной разработке Аллен Уорд (Allen Ward) полагает, что существует три категории неэффективного использования приобретенных знаний: распыление, передача из рук в руки и принятие желаемого за действительное (2007).

Распыление имеет место, когда по каким-либо причинам работа прерывается. На индивидуальном уровне к распылению приводит множество причин, которые отвлекают нас от работы или разбивают наш рабочий день на мелкие фрагменты, мешая нам сосредоточиться и выполнить сколько-нибудь существенную работу. На уровне проекта к распылению приводит прерывание командной работы (например, когда команде приказывают прекратить текущее задание и приступить к какой-то другой функциональной возможности, когда команда пополняется новым работником или кто-то уходит из нее или когда команде приказывают вносить все новые и новые правки в уже выполненную ею работу, в то время как сроки сдачи проекта начинают “поджимать”).

У распыления две главные причины: препятствия, мешающие плодотворному общению людей, и неэффективные инструменты. Препятствия, мешающие общению людей, могут быть физическими (например, если членов команды разделяют пять тысяч миль или два этажа). Однако препятствия, мешающие общению людей, могут быть и результатом неэффективной корпоративной политики (“все запросы на изменение базы данных должны быть представлены в письменном виде”) или недостаточной квалификации людей (например, неспособность двух групп общаться из-за того,

что они используют разную терминологию). Эд Катмулл (Ed Catmull), один из основателей Pixar² (создавшей, в частности, такие анимационные фильмы, как *Toy Story*, *Finding Nemo*, *The Incredibles*, *Monsters, Inc.*), признает существование таких барьеров.

Заставить людей, представляющих разные дисциплины, относиться друг к другу, как к равным себе, так же важно, как подвигнуть к тому же представителей одной и той же дисциплины. Однако сделать это намного труднее. Барьерами в данном случае являются естественные классовые структуры, возникающие в организациях: в любой организации всегда найдется должность, которая считает себя самой ценной в этой организации (и которую считают таковой все остальные сотрудники данной организации). К тому же представители разных дисциплин нередко разговаривают на разных языках, что не способствует общению между ними. Общению препятствует даже физическое расстояние между офисами (2008, 70).

Под неэффективными инструментами Аллен Уорд подразумевает не только программные продукты, которые уже стали неотъемлемой составляющей нашей повседневной жизни. Неэффективными инструментами, которые приводят к распылению, являются общепринятые методы, применяемые в типичном процессе разработки. Например, одной из компаний, которые я консультировал, не удалось предвидеть последствия изменений в базе данных, совместно используемой многими приложениями. Это привело к появлению нового правила, согласно которому каждая новая функциональная возможность должна сопровождаться *отчетом о влиянии на базу данных*. Подавляющее большинство изменений в приложениях, разумеется, не оказывало влияния на базу данных, но составление этого стандартизированного отчета требовалось в любом случае. Вместо того чтобы в обязательном порядке требовать от исполнителей всех проектов заполнения стандартной формы, можно было бы предложить более подходящий вариант: четко указать на обязанность всех команд, выполняющих проекты, учитывать и формулировать влияние своих проектов на базу данных. Целью должна быть ответственность за результаты, а не точное соблюдение процесса.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ В ходе ретроспективы следующего спринта попытайтесь выявить по меньшей мере десять причин распыления, влияющих на вашу команду. Выберите две причины из этого десятка, которые вы должны будете устраниить в течение следующего месяца. Выявите причины, которые приводят к распылению в течение отдельно взятого дня, а также причины, которые приводят к распылению на протяжении всего времени выполнения проекта.

Передачу из рук в руки Аллен Уорд определяет как разделение знаний, ответственности, действий и обратной связи. Передача из рук в руки происходит буквально на всех этапах процесса последовательной разработки программного обеспечения. Результаты анализа передаются архитекторам, которые, в свою очередь, передают результат своих действий (архитектуру) программистам. Программисты передают написанные ими коды тестерам. Почти вся письменная документация проекта создается таким образом, чтобы обеспечить возможность передачи из рук в руки. Однако не все

² Мультипликационная студия, работающая в жанре 3D-анимации. — Примеч. пер.

такие передачи являются артефактами. Примером передачи из рук в руки ответственности является возложение на традиционного руководителя проекта ответственности за соблюдение проектных спецификаций и сроков выполнения проекта, между тем как руководитель проекта не способен реально влиять ни на то, ни на другое.

Межфункциональные команды приобретают все большую и большую популярность, в том числе и как ответ на проблемы, порождаемые передачей из рук в руки в случае проектов, выполняемых традиционным способом. Вспомним, о чем говорилось в разделе “Воспитание чувства коллективной ответственности” настоящей главы. Основная мысль этого раздела заключалась в том, что, хотя выполнение определенных задач может возлагаться на определенного исполнителя, почти за все отвечает вся команда в целом. Чем больше действует команда в целом и чем больше эта команда в целом ощущает свою совместную ответственность, тем меньше будет передач из рук в руки. Избавляясь от передач из рук в руки, мы избавляемся от проблем, порождаемых ожиданием и необходимостью передавать знания от одного человека к другому.

Третьей категорией неэффективного использования приобретенных знаний, согласно Аллену Уорду, является *принятие желаемого за действительное*. Принятие желаемого за действительное — это не просто неоправданный оптимизм. Здесь мы подразумеваем принятие решений в отсутствие адекватной информации, которая подкрепляла бы эти решения. Срыв сроков выполнения проекта является наиболее очевидным результатом принятия желаемого за действительное. Выбор определенной даты, создание спецификации и наивная надежда на то, что проект будет выполняться в точном соответствии с планом и без каких-либо непредвиденных изменений, является естественным следствием принятия желаемого за действительное. Знания, оказавшиеся на поверку бесполезными, являются вторым типом принятия желаемого за действительное. Такие знания обусловлены неспособностью команд применить приобретенные ими знания с пользой для дела. Когда команда обнаруживает и исправляет редко встречающуюся ошибку, но не способна добавить автоматизированный тест, который предотвращал бы такие ошибки в дальнейшем, мы также имеем дело со знаниями, оказавшимися бесполезными. Если команда полагает, что подобная ошибка никогда не возникнет вновь, то она принимает желаемое за действительное.

Трудно переоценить важность командного обучения. Я повидал на своем веку очень много команд, квалификация и мастерство которых значительно выросли после перехода к Scrum, но которые после перехода к Scrum остановились в своем развитии. Постоянное совершенствование является неотъемлемой составляющей Scrum. Отсутствие стремления к обучению и неэффективное использование приобретенных знаний представляют собой серьезные проблемы.

Стимулируйте сотрудничество, напоминая о совместной цели

Томми Ласорда (Tommy Lasorda), давнишний менеджер бейсбольной команды “Лос-Анджелес Доджерс”, однажды сказал: “Моя обязанность — заставить всех двадцать пять моих парней играть во имя того, что написано на их футболках спереди, а не во имя того, что написано на спине” (LaFasto and Larson, 2001, 100). Вы стремитесь

стать высокоэффективной Scrum-командой? Командное обучение позволит вам достичь на этом пути определенного уровня — и не более того. Чтобы ваша самоорганизующаяся команда продолжала действовать как единое целое, а не как случайным образом составленная группа людей, необходимо постоянно подзаряжать ее энергией и акцентировать ее внимание на целях, общих для всей команды. Для этого нужно изыскивать все новые и новые способы напоминать людям об их совместной цели и об их ответственности друг перед другом. Ниже указано, что можно сделать для формирования и поддержания в людях такой взаимной ответственности.

Вовлекайте людей в как можно более широкий круг деятельности. Одно из наиболее частых нареканий, которые мне приходилось слышать от программистов, заключается в том, что они не хотят, чтобы к ним относились, как к “обезьянам-кодировщикам”. Этим термином они обозначают человека, которому говорят, что именно он должен кодировать, и которому приходится исключить из своей работы любые элементы творчества (и удовольствия). Избежать отношения к разработчикам, как к “обезьянам-кодировщикам” можно, вовлекая их в как можно более широкий круг деятельности, связанной с реализацией проекта (но не теряя при этом чувства меры и практической целесообразности). Вот почему я, например, настаиваю на участии *всех* разработчиков в семинарах, целью которых является написание историй для журнала запросов на выполнение работ. Чем шире картина проекта и продукта, которую могут увидеть члены команды, тем больше они будут вовлечены в этот проект и тем большую ответственность за успешную реализацию этого проекта они будут чувствовать.

См. также Тем, кто хочет получить более полную информацию о проведении семинаров, посвященных написанию историй, советую прочесть книгу *User Stories Applied for Agile Software Development* (Cohn, 2004).

Придумайте цель, способную вдохновить людей и вызвать у них прилив энтузиазма. Профессор Лондонской экономической школы (London Business School) Линда Граттон (Lynda Gratton) использует термин “горячая точка” (hot spot) для обозначения места и времени, “когда работа с людьми приносила наибольшее удовольствие и удовлетворение и когда вы были абсолютно уверены в важности и содержательности вашей совместной работы” (2007, 1). Чтобы место вашей работы превратилось в такую “горячую точку”, поставьте перед людьми цель, способную их вдохновить и вызвать у них прилив энтузиазма (в оригинале такая цель называется “igniting purpose”, словно — “воспламеняющая цель”) (13).

В середине 1990-х годов я работал в компании, которая поставила перед собой цель реформировать систему здравоохранения, изменив взаимоотношения между пациентами и поставщиками медицинских услуг. Основной штат служащих этой компании составляли медсестры, работавшие в колл-центре. Системы для этих медсестер создавала группа разработчиков. Каждую неделю главная медсестра отправляла по электронной почте письмо, в котором резюмировалась важная информация за истекшую неделю. Почти вся эта информация была весьма тривиальной: сколько появилось новых клиентов, на сколько телефонных звонков удалось ответить сотрудникам колл-центра, сколько времени в среднем затрачивалось на один ответ и т.п.

Что в этой системе было по-настоящему нетривиальным и что составляло цель, способную вдохновить сотрудников компании и вызвать у них прилив энтузиазма, так это отчет о количестве спасенных жизней. Я помню один конкретный телефонный звонок, о котором сообщила нам по электронной почте главная медсестра. Звонил мужчина, у которого возникла острые боль под левой лопаткой. Он хотел выяснить, нужно ли ему записаться на прием к врачу или просто принять таблетку ибупрофена³. Задав пациенту несколько вопросов и воспользовавшись экспертной системой, входящей в состав разработанного нами программного обеспечения, медсестра пришла к выводу, что у звонившего случился сердечный приступ. Медсестра направила к нему карету “скорой помощи” еще до того, как он положил телефонную трубку, и тем самым спасла ему жизнь. Цель, способная вдохновить членов команды и вызвать у них прилив энтузиазма, вовсе необязательно должна быть столь высокой и благородной, как спасение человеческих жизней. Это просто должно быть что-то такое, что способно по-настоящему вдохновить людей, вызвать у них неподдельный интерес и заставить их всеми силами стремиться к осуществлению такой цели.

Попытайтесь понять внутренние мотивации людей. Помимо необходимости поставить перед командой цель, способную вдохновить членов команды и вызвать у них прилив энтузиазма, нужно подпитывать внутренние мотивации, уже существующие у членов вашей команды. Мотивации у всех могут быть разные, но проект, структурированный таким образом, что уникальные, личные цели каждого из членов команды согласуются с целями и задачами данного проекта, обязательно заставит всю команду действовать с максимальной целеустремленностью. Возможно, кто-то из Java-разработчиков хочет научиться программировать на C#. Есть ли возможность осуществить это желание в данном проекте? Возможно, кто-то из тестеров желает по-пробовать себя в роли лидера. Можно ли доверить ему такое ответственное дело, как выбор стороннего производителя, который будет разрабатывать какие-то из компонентов нашей будущей системы?

Бойтесь самого немотивированного из членов вашей команды. Один высокомотивированный и квалифицированный исполнитель зачастую оказывает положительное влияние на остальных членов команды. С другой стороны, даже один немотивированный член команды способен потащить за собой вниз всю команду. Вот как описывает Кристофер Эйвери (Christopher Avery) отрицательное влияние такой “паршивой овцы” на все стадо.

По собственному опыту знаю: когда в команду приходит человек с иждивенческими настроениями, от которого невозможно избавиться из-за бюрократической политики, даже самые трудолюбивые члены команды сразу же — и резко — снижают свою производительность и начинают уделять повышенное внимание другим сторонам своей жизни (Avery, Walker, and O’Toole, 2001, 97).

Помогите каждому члену своей команды понять его роль в достижении цели, поставленной перед командой. Никому из нас не нравится чувствовать себя лишним на этом празднике жизни. Никому из нас не нравится осознавать свою второстепенную роль в проекте, выполняемом командой. Членам команды трудно полностью проникнуться целями проекта и ощутить свою ответственность за достижение этих целей, если их собственный вклад в реализацию данного проекта кажется им незначительным.

³ Аналгетический и противовоспалительный лекарственный препарат. — Примеч. пер.

Помочь каждому из членов команды почувствовать важность собственного вклада в достижение цели, поставленной перед командой, должен, в первую очередь, владелец продукта, но нужно, чтобы это подтверждалось и соответствующими комментариями каждого члена команды.

Формируйте в людях чувство уверенности. Зная, насколько сложна поставленная перед ними задача, члены команды хотят быть уверенными в том, что они смогут решить эту задачу. Чувство уверенности порождает не простота цели, поставленной перед командой, а вера в собственные силы и в силы товарищей по работе. Людям нравится работать с теми, кто поддерживает в них уверенность в собственных силах. Уверенной в себе команде любая цель по плечу.

Помните, что формирование у людей чувства ответственности за выполняемый проект не является одноразовым мероприятием. Командам требуется периодическая “подзарядка” энергией, чтобы подкрепить у людей чувство ответственности не только за выполняемый ими проект, но и друг перед другом. В книге *Teamwork Is an Individual Skill* Кристофер Эйвери (Christopher Avery) высказывает следующую мысль: несмотря на то что границы между календарными годами и кварталами являются вполне подходящими моментами для такой “подзарядки” энергией, “наилучшим временем для того, чтобы встряхнуть команду, является любое время, когда вы начинаете замечать, что у людей пропадает ощущение совместной цели и их энергия тает” (107).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Есть ли у вашей команды цель, способная вдохновить членов команды и вызвать у них прилив энтузиазма? Могут ли все члены команды четко сформулировать эту цель? Совпадают ли формулировки цели, предложенные разными членами команды? Если вы не можете ответить на эти вопросы положительно, попросите владельца продукта провести собрание, руководствуясь информацией, изложенной в главе 12, “Управление самоорганизующимся коллективом”.
- Понимаете ли вы мотивацию каждого из членов своей команды? Если нет, попытайтесь уяснить мотивации каждого из них. Как именно? Просто расспросите их.
- Понимают ли другие вашу собственную мотивацию? Если нет, расскажите им о ней.

А теперь все вместе

Выработать у людей ощущение сплоченного коллектива не так-то просто. Scrum-мастера могут вам помочь, позаботившись о том, чтобы команда усвоила концепцию коллективной ответственности и коллективной готовности создавать работоспособное программное обеспечение к окончанию каждого спринта. Поначалу команде будет нелегко преодолевать устоявшиеся привычки, связанные с узкой специализацией каждого и передачей работы из рук в руки. Минимизируя распределение задач между конкретными исполнителями и выполняя на протяжении всего спринта понемногу то одну, то другую работу, вы сможете существенно упростить переход от последовательного способа разработки к командному способу работы. После того как команда научится работать вместе и предоставлять заказчику по окончании каждого очередного спринта то, что она обязалась выполнить в течение этого спринта, она имеет

полное право гордиться своими достижениями. Однако ни в коем случае не следует поддаваться соблазну самодовольства и самоуспокоенности лишь потому, что вам удалось стать работоспособной Scrum-командой. Чтобы стать по-настоящему высоко-производительной Scrum-командой, нужно продолжать учиться и совершенствоватьсья. Способствуйте обучению своей команды, устранийте источники неэффективного использования приобретенных знаний и стимулируйте сотрудничество, постоянно напоминая членам команды об их совместной цели и изыскивая способы взбодрить команду на протяжении всего времени выполнения проекта.

В следующей главе мы рассмотрим способы, с помощью которых лидеры могут оказывать еще большее влияние на самоорганизующиеся команды, способствуя дальнейшему повышению их эффективности и достижению оптимальной производительности.

Дополнительная литература

Avery, Christopher M., Meri Aaron Walker, and Erin O'Toole. 2001. *Teamwork is an individual skill: Getting your work done when sharing responsibility*. Berrett-Koehler Publishers.

Исходный тезис этой книги — каждый член команды должен брать на себя ответственность за результаты команды в целом. Автор приводит подробную информацию и случаи из жизни о том, как любой член команды может улучшить результаты работы и повысить производительность команды в целом.

Katzenbach, Jon R., and Douglas K. Smith. 1993. *The wisdom of teams: Creating the high-performance organization*. Collins Business.

Одна из ранних классических работ, посвященных организации и деятельности команд, которая выдержала проверку временем. В ней освещаются все аспекты команд, включая стадии, через которые они проходят в процессе своего формирования и развития, структуру команд (кто именно должен входить в состав команды), вопросы лидерства (кто именно должен возглавлять команду), роль менеджмента в работе с командами и т.д.

Larson, Carl E., and Frank M. J. LaFasto. 1989. *Teamwork: What must go right/what can go wrong*. SAGE Publications.

Авторы этой книги потратили три года на изучение опыта работы и интервьюирование 32 успешных команд, представляющих достаточно широкий спектр отраслей. В числе этих команд были команда сердечно-сосудистой хирургии, команда, взошедшая на Эверест, спортивные команды, ставшие чемпионами в своих видах спорта, команда конструкторов самолетов и даже команда McDonald's, которая изобрела блюдо под названием "Chicken McNuggets". На основании собранной информации авторы сформулировали восемь характеристик высокоеффективной команды.

Глава 12

Управление самоорганизующимся коллективом

Одна из первых моделей организационных изменений была предложена Куртом Левином (Kurt Lewin) в 1940-х годах. Согласно этой модели изменения представляют собой процесс, состоящий из трех этапов: “размораживание” текущей ситуации, что делает возможным осуществление изменений; переход в новое состояние; последующее “замораживание” новой ситуации, обеспечивающее ее устойчивость. Многие из последующих моделей организационных изменений подобны модели Курта Левина с точки зрения представления ими протяженных периодов относительной стабильности, перемежающихся короткими периодами перехода из одного состояния в другое.

Модель, предложенная Куртом Левином, отражает ситуацию, характерную для начала XX века. С тех пор мир изменился до неузнаваемости. Изменения уже не происходят, как прежде, короткими рывками, которые прерывают длительные периоды относительной стабильности. Вместо того чтобы переходить из одного равновесного состояния в другое, организации XXI века действуют в условиях, весьма далеких от равновесных. Несмотря на беспорядок и сумасшедший темп, которые явились следствием этого, нельзя отрицать и определенных выгод от такого положения. Организации, пребывающие в состоянии равновесия и стремящиеся вернуться в это состояние, будучи выведенными из него, сопротивляются переменам (Goldstein, 1994, 15). Организации, пребывающие в состоянии, далеком от равновесного, лучше приспособлены к постоянным изменениям. Именно лидеры организаций и агенты перемен должны обеспечивать пребывание своих организаций в состояниях, далеких от равновесного.

Лидеры добиваются этого, периодически взбадривая, встряхивая, стимулируя и перестраивая свои организации. Делая это, они пытаются удержать организации от сползания в состояние равновесия, из которого их бывает очень трудно вывести. Тем самым они удерживают организации в тонусе, давая им возможность эффективнее реагировать на перемены, вызванные внешними причинами, или самостоятельно инициировать перемены. Такое подбадривание становится фундаментальным способом,

с помощью которого лидеры и агенты перемен заставляют свои организации становиться более гибкими и динамичными.

Так кто же эти лидеры и агенты перемен? Трудно ответить на этот вопрос, не зная специфики конкретной организации, но когда я говорю “лидеры”, то имею в виду каждого, кто обладает определенным влиянием и авторитетом в соответствующей команде. Это относится к менеджерам, которые принимают на работу и увольняют членов команды. Это относится к владельцу продукта, который определяет рамки продукта (или системы), подлежащего разработке. Это относится к Scrum-мастера, который может осуществлять небольшие, но значимые изменения в соответствующем процессе. Наконец это относится к организационным агентам перемен, работающим над внедрением или дальнейшим распространением Scrum.

В последующих разделах мы рассмотрим, как эти лидеры, менеджеры и агенты перемен могут влиять на самоорганизующийся путь команды или компании. Вы узнаете о трех условиях, при которых возможна самоорганизация, а также о том, как лидеры могут влиять на эти условия. Вы узнаете также о том, как развиваются организации, а также о семи способах, посредством которых лидеры, менеджеры и агенты перемен могут оказывать влияние на эволюцию своих организаций.

Влияние на самоорганизацию

Самоорганизация является одной из фундаментальных концепций гибкой методологии разработки программного обеспечения. В Agile Manifesto имеется следующий принцип: “Источником наилучших архитектурных решений, требований и проектных решений являются самоорганизующиеся команды” (Beck et al., 2001). Отсюда возникает типичное заблуждение: из-за упования на самоорганизующиеся команды снижается или вообще сводится к нулю роль лидеров Scrum-команд. Подобный вывод, однако, не имеет ничего общего с действительностью. В книге *The Biology of Business* Филип Андерсон (Philip Anderson) опровергает это ошибочное предположение.

Самоорганизация вовсе не означает, что организацией работы занимаются не менеджеры, а рядовые исполнители. Самоорганизация вовсе не означает, что людям предоставляется возможность делать все, что они хотят. Самоорганизация означает, что руководство управляет эволюцией поведения, которое возникает в результате взаимодействия независимых агентов, вместо того чтобы с самого начала указывать, в чем именно заключается эффективное поведение (1999, 120).

Самоорганизующиеся команды не освобождаются от контроля со стороны руководства. Руководство выбирает, какой продукт будет создаваться, и зачастую решает, кто именно будет работать над тем или иным проектом; тем не менее команды с полным правом называются самоорганизующимися. Самоорганизующиеся команды отнюдь не свободны от влияния извне. Уже в самых ранних упоминаниях о Scrum не возникает и тени сомнения по этому поводу. В статье “The New New Product Development Game” (1986) Такучи (Takeuchi) и Нонака (Nonaka) пишут о том, что “ненавязчивый контроль отнюдь не противоречит самоорганизующемуся характеру проектных команд”. Позднее, в 1990 году, де Грейс (DeGrace) и Стал (Stahl) описывают в своей книге *Wicked Problems, Righteous Solutions*, как менеджеры косвенным образом контролируют деятельность самоорганизующейся команды.

Несомненно, контроль по-прежнему осуществляется, но он не бросается в глаза и происходит косвенным образом. Он осуществляется путем подбора наиболее подходящих исполнителей, формирования открытой рабочей атмосферы, налаживания обратной связи с исполнителями, создания системы оценки и вознаграждения исполнителей, базирующейся на групповых результатах, сдерживания склонности исполнителей к распылению сил на ранних этапах реализации проекта и удовлетворения потребности в объединении информации и усилий на более поздних этапах реализации проекта, адекватного реагирования на появление ошибок (и даже прогнозирования ошибок) и вовлечения поставщиков в реализацию проекта уже на ранних его этапах, но без контроля над ними (159).

Проблема, которую предстоит решить Scrum-команде, заключается в том, чтобы самоорганизоваться вокруг задач, поставленных перед ней руководством, в рамках ограничений, сформулированных руководством. Обязанность руководства заключается в том, чтобы предложить команде для решения подходящие задачи и устраниć препятствия к ее самоорганизации.

Учитывая сказанное, чем меньше ограничений или контролирующих воздействий оказывается на команду, тем лучше. Если лидеры чрезмерно ограничивают команду в способах решения поставленной перед ней задачи, то не может быть и речи о какой-либо самоорганизации. Команда просто застынет в ожидании: поскольку ей уже сказали более чем достаточно о задаче, которую ей предстоит решить, и о способах ее решения, она будет ждать, когда ей скажут и все остальное.

Итак, каким же образом лидер добивается тонкого баланса между командованием и влиянием? Указанного баланса можно добиться, в частности, уяснив, каким образом даже незначительные изменения трех условий, влияющих на способ самоорганизации команд, могут оказывать огромное воздействие на способ самоорганизации команд (а следовательно, и на результаты их деятельности). Этими условиями являются контейнеры, различия и обмены.

Контейнеры, различия и обмены

В своей докторской диссертации Гленда Эоянг (Glenda Eoyang) описывает три условия, изменение которых влияет на способ самоорганизации команд: контейнеры, существенные различия и трансформирующие обмены (2001).

Контейнер — это определенные рамки, в которых осуществляется самоорганизация. Представьте, что вы находитесь в кинотеатре, причем человек, купивший билет в такой кинотеатр, может занять в зрительном зале любое свободное место. Физические границы кинотеатра образуют контейнер, в котором вы и остальные зрители самоорганизуетесь, занимая те или иные свободные места в зрительном зале. В соседний кинотеатр приходят другие зрители, которые самоорганизуются в пределах своего физического контейнера. Два разных контейнера (кинотеатра) различны, поэтому нельзя сказать, что зрители в одном кинотеатре самоорганизуются со зрителями в другом кинотеатре. Контейнеры не обязаны быть физическими. Как видно из приведенных ниже примеров контейнеров, они могут быть поведенческими, организационными и концептуальными.

- Все, кто работают в университете городке Сан-Хосе
- Все, кто работают в корпусе Б

- Все, кто работают в отделе разработки программного обеспечения
- Все, кто программируют на языке Ruby
- Все, кто являются норвежцами
- Все, кто принадлежат к Agile Alliance
- Все члены команды, выполняющей проект Capricorn

Различия между людьми, находящимися внутри одного контейнера, также влияют на способ их самоорганизации. Если бы между членами нашей Scrum-команды не было никаких различий, то не имело бы значения, кто какую работу выполняет и взаимодействуют ли между собой члены команды. Поскольку квалификация членов такой команды была бы во всех отношениях одинаковой, каждый из них мог бы работать совершенно отдельно от остальных. К счастью, между членами команды разработчиков программного обеспечения всегда есть различия, к числу которых относятся техническая квалификация, знание предметной области, властные полномочия, пол, расовая принадлежность, образование, связи с другими сотрудниками данной компании, подход к решению проблем и т.д. Типы и степени этих различий также влияют на способ самоорганизации команды.

Наконец, трансформирующие обмены влияют на то, как команда организуется в ответ на возникновение той или иной проблемы. Трансформирующие обмены — это такое взаимодействие между членами, находящимися внутри определенного контейнера, в результате которого один или более человек изменяются или подвергаются влиянию. Например, я встречаюсь с владельцем продукта моего проекта, который отвечает на мои вопросы о том, в чем заключается функциональная возможность, в разработке которой я буду принимать участие. Это и будет трансформирующий обмен, поскольку в ходе общения с владельцем продукта я приобретаю новые знания. Трансформирующий обмен — это неизбежно обмен информацией между людьми. В ходе такого обмена может происходить передача денег, властных полномочий, энергии и многое другое. Команда, мотивированная разговором со своим владельцем продукта, также становится объектом трансформирующего обмена: создается энергия, которая передается команде.

Что эти три условия означают для лидеров и агентов перемен с практической точки зрения? Регулируя контейнеры, усиливая или ослабляя различия и изменения трансформирующие обмены, лидеры могут влиять на способ самоорганизации команд. Это и есть одна из форм *ненавязчивого и малозаметного контроля*, упоминавшегося в начале этой главы. Допустим, к примеру, что один из членов команды, Джейф, диктует всем свои условия и никто не решается ему перечить. Эта команда самоорганизовалась: она решила предоставить Джейфу возможность принимать все ключевые решения. Являясь Scrum-мастером этой команды, вы понимаете, что такое положение вещей мешает совершенствованию команды. Вы принимаете решение поговорить один на один с Джейфом, но это не приводит к заметному изменению ситуации. Вы подумываете о том, не отменить ли некоторые из решений, принятых Джейфом, но отдаете себе отчет в том, что, если вы поступите так хотя бы один раз, команда будет рассчитывать, что точно так же вы будете поступать и в дальнейшем, что, конечно же, плохо.

Не зная, как поступить, вы начинаете задумываться о контейнерах, различиях и трансформирующих обменах, которые влияют на способ самоорганизации данной команды. Вы понимаете, что повлиять на ситуацию можно было бы, уменьшив

различия между членами команды. В результате вы решаете принять в состав команды нового члена, который будет время от времени давать отпор Джейфу. Можно также попытаться осуществлять ненавязчивый и малозаметный контроль над командой путем изменения трансформирующих обменов. Для этого вы предлагаете участвовать в важнейших совещаниях кому-либо из членов команды архитекторов предприятия. Какой бы ни была конкретная проблема, если вы видите, что команда самоорганизовалась таким образом, что это снижает эффективность ее работы, вы обязаны найти способ встряхнуть команду, взбудоражить ее или каким-то иным способом нарушить сложившийся порядок вещей, чтобы она внесла соответствующие корректизы в свою работу (возможно, путем реорганизации).

В книге *Facilitating Organization Change* Гленда Эоянг и ее соавтор Эдвин Олсон (Edwin Olson) отстаивают именно такой подход.

Роль агента перемен заключается в использовании понимания эволюционирующих моделей для изменения контейнера, различий и трансформирующих обменов, которое повлияло бы на способ самоорганизации, для наблюдения за реакцией системы и для планирования очередного вмешательства. Цель такого экспериментирования, ориентированного на действия, заключается в том, чтобы предвидеть, адаптировать и оказывать влияние, а не предсказывать или контролировать поведение соответствующей системы (2001, 16).

ВОЗРАЖЕНИЕ

“Это звучит неубедительно. Как может самоорганизоваться команда, если какой-то начальник или агент перемен контролирует ситуацию из-за кулис?”

Самоорганизация отнюдь не означает, что команде можно делать все, что она пожелает. Люди самоорганизуются для решения определенной проблемы, сформулированной для них организацией. (“Нам нужен продукт, который обладает следующими возможностями.”) Контейнеры, существенные различия и трансформирующие обмены создаются путем организационного влияния, но не определяют, как команда самоорганизуется для решения конкретной проблемы.

Кроме того, нужно помнить, что агент перемен возится с командой или контейнерами, различиями и обменами не ради собственного удовольствия. Он занимается этим, в частности, для того, чтобы помочь команде стать максимально эффективной.

Регулировка контейнеров

Колин, директор развития в компании, занимающейся разработкой программного обеспечения для медицинских целей, был в отчаянии от неспособности своей команды создавать работоспособное программное обеспечение к окончанию каждого из спринтов. Его беспокоил вовсе не объем работы, выполняемой командой в течение каждого спринта — каждый раз он был достаточно внушительным. Он был в отчаянии по другой причине: вместо того, чтобы полностью завершить пять элементов к окончанию спринта, команда выполняла наполовину каждый из десяти элементов, запланированных на этот спринт. Колин понимал, что Scrum-команда не должна так работать.

См. также Важность завершения каждого спринта созданием работоспособного программного обеспечения обсуждается в главе 14, “Спринты”.

Мы вместе с Колином обсудили ситуацию и я ознакомил его с моделью CDE — Container (контейнер), Difference (различие), Exchange (обмен). Колин полагал, что в его команде общаются между собой именно те, кто надо, и что вносить какие-либо корректизы в текущие обмены (или предлагать какие-либо новые) нет никакой нужды. Мы обсудили различия между членами команды и пришли к выводу, что одним из подходящих вариантов решения возникшей проблемы было бы включение в команду опытного разработчика, хорошо знающего гибкую методологию разработки. Такой разработчик мог бы помочь команде уяснить проблемы, над которыми она работала. К сожалению, под рукой у Колина не оказалось опытных разработчиков, которых можно было бы включить в состав этой команды.

См. также Достоинства команд, реализующих функциональные возможности, и компонентных команд обсуждаются в главе 10, “Структура команды”.

Обсуждая контейнеры, в рамках которых действовала эта команда, Колин понял, что возможным решением было бы расширение сферы ответственности команды. По мнению Колина, неспособность этой команды создавать работоспособное программное обеспечение к окончанию каждого из спринтов во многом объяснялась тем, что данная команда была зависима от функциональных возможностей нижнего уровня, разрабатывавшихся другой командой. Колин решил объединить эти две команды. Объединив их, он сделал бы такую команду полностью ответственной за работу, которую на тот момент приходилось разделять между двумя командами. Это устранило бы одну из причин, которые оправдывали неспособность команды создавать работоспособное программное обеспечение к окончанию спринта. Обратившись ко мне впоследствии по электронной почте, Колин так описал ход своей мысли.

Лишь часть их проблем была вызвана задержками, связанными с ожиданием другой команды. Но они привыкли оставлять элементы журнала запросов на выполнение работ выполненными лишь наполовину. Нагружая команду новыми обязанностями и вводя в ее состав новых членов, я старался тем самым подчеркнуть, что полностью завершить к окончанию спринта меньшее количество работ все же лучше, чем взяться за выполнение большого количества работ, не имея возможности справиться с ними к окончанию спринта.

Подход Колина к расширению обязанностей, заключающийся в расширении контейнера, — это лишь один из возможных способов регулировки контейнеров с целью оказать влияние на команду. Этот, а также ряд других способов, перечислены в табл. 12.1.

Таблица 12.1. Варианты использования контейнеров для влияния на самоорганизацию команды

Изменить количество людей в команде.

Изменить персональный состав команды.

Окончание табл. 12.1

Воспользоваться новым контейнером, таким как сообщество практических пользователей Scrum.

Наделить команду большей или меньшей ответственностью.

Изменить физическое пространство, предоставленное команде. Предоставить членам команды большее или меньшее пространство. Убрать или сделать ниже перегородки между отсеками, занимаемыми членами команды. Сосредоточить всех членов команды на одном этаже.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Перечислите все контейнеры, в которых работают члены команды. Можно ли утверждать, что размер и охват этих контейнеров выбраны правильно? Может быть, этих контейнеров слишком много (мало)?
- Для каждого из контейнеров решите, как (положительно, отрицательно или нейтрально) он влияет на производительность команды.
- Определите контейнер, который, по вашему мнению, оказывает на команду наибольшее влияние. Следует ли как-то его изменить?

Усиление или ослабление различий

Кейри, директор развития, инициировавшая внедрение Scrum в своей компании, была озабочена недавним, но непрекращающимся снижением скорости работы одной из ее команд. Качество работы этой команды оставалось высоким, но объем выполняемой работы становился все меньшим и меньшим по сравнению с тем, каким он был еще несколько месяцев назад. В ходе регулярных 30-минутных встреч один на один с каждым из своих подчиненных Кейри спрашивала у некоторых членов этой команды, почему, на их взгляд, это произошло. Проведя ряд таких встреч, Кейри приняла участие в ретроспективе очередного спринта этой команды.

В результате Кейри узнала, что шесть-девять месяцев назад команда приняла несколько неудачных архитектурных решений. Кейри сопоставила то, что она узнала из этой ретроспективы и своих ежемесячных встреч с членами команды, и то, что ей уже было известно о каждом из членов команды. В итоге этого сопоставления Кейри пришла к выводу, что, несмотря на неизбежность появления части неудачных решений, некоторые из неудачных решений этой команды явились результатом задания друг другу неправильных вопросов.

Раньше я рассказал Кейри о том, что, воспользовавшись концепцией контейнеров, различий и обменов, она существенно облегчила бы себе задачу управления своими командами. Впоследствии она рассказала мне, что воспользовалась этим подходом в данной ситуации. Проанализировав модель CDE, Кейри пришла к выводу, что различия между членами этой команды явно недостаточны. Она решила, что помочь команде можно, усилив различия между членами команды. Кейри осуществила это с помощью одного из моих любимых приемов: она задала множество “прощупывающих” вопросов.

Кейри всегда относилась к той категории руководителей, которые с удовольствием перекладывают решение проблем на своих подчиненных, однако в данном случае она решила, что команда нуждается в большем внимании с ее стороны. Она начала

посещать импровизированные собрания команды. Во время этих собраний она задавала вопросы, целью которых было “вытащить” из людей особое, нестандартное мнение. Она задавала, например, такие вопросы.

- Какие альтернативные подходы вы рассматривали и отвергли, прежде чем принять данный вариант решения?
- Что могло оказаться ошибочным в этом подходе?
- Что нужно изменить, чтобы этот подход оказался эффективным?
- Что может заставить нас сожалеть об этом решении?
- Есть ли какая-то информация, которой мы не располагаем, но которая помогла бы нам удостовериться в этом?

Даже когда Кейри соглашалась с мнением большинства, она задавала “трудные” вопросы, пытаясь выявить в этом мнении какие-то просчеты и надеясь, что кто-то сможет предложить еще более удачное решение.

Еще один эффективный способ усилить различия заключается в том, чтобы изменить сам способ принятия командой решений. Если, например, в настоящее время команда принимает решения путем голосования (принимается решение, за которое проголосовало большинство членов команды), то можно предложить, чтобы в течение очередных двух спринтов решения принимались единогласно. Эти и другие методы усиления или ослабления различий представлены в табл. 12.2.

Таблица 12.2. Варианты усиления или ослабления различий для влияния на способ самоорганизации команды

Ввести в состав команды нового члена, имеющего значительно большие властные полномочия, практический опыт, знания и т.п.

Задавать команде “трудные” вопросы, добиваясь таким образом вынесения на суд команды разных точек зрения.

Изменить стиль принятия решений командой.

Поощрять выражение особых мнений, не совпадающих с преобладающей точкой зрения.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- По каждому из различий между членами команды (например, технические знания, знание предметной области, продолжительность работы в данной компании, уважение со стороны коллег, стиль решения проблем) оцените различия между членами команды по шкале от 1 до 10. Не свидетельствуют ли ваши оценки о слишком больших (или слишком малых) различиях между членами команды?
- Определите одно различие, усиление которого привело бы к повышению производительности команды. Можно ли ввести в состав команды нового члена, который усилил бы это различие?
- Определите одно различие, ослабление которого привело бы к повышению производительности команды. Можно ли ослабить это различие в результате выведения кого-либо из состава данной команды?

Изменение обменов

Лидер или агент перемен в организации может также оказать влияние на команду путем изменения обменов, в которых участвуют члены команды. Александро, технический руководитель в одной из студий, занимающихся разработкой видеоигр, обнаружил проблему, участвуя в третьем за день обзоре спринтов. В состав каждой из команд входил программист искусственного интеллекта (ИИ). Программисты ИИ отвечали за поведение “плохих парней”, которые по ходу видеогры атаковали игрока. Во время обзора спринтов Александро обратил внимание на некоторые высказывания, натолкнувшие его на мысль о том, что каждая из команд программирует свой ИИ несколько по-другому, чем остальные команды. Это не только могло привести к противоречиям и путанице в ходе игры, но и порождало дублирование ряда работ.

Я встретился с Александро уже после того, как он выявил и решил эту проблему. Его решение заключалось в том, чтобы ввести в действие новый обмен. Поскольку программисты ИИ общались между собой сравнительно мало, Александро решил, что они должны встречаться не реже одного раза в неделю. Участие других специалистов в таких встречах не предполагалось. Несмотря на то что сам Александро не был программистом, его авторитет в организации был достаточно высок, чтобы убедить программистов ИИ в целесообразности такого общения. В ходе двухнедельного спринта программисты ИИ встречались на следующий день после планирования спринта. Таким образом, каждому из них уже было известно, какие работы обязались выполнить в течение данного спринта другие программисты ИИ. Вторая встреча проводилась в начале второй недели спринта, что давало возможность программистам ИИ сравнивать достигнутый прогресс и свои ожидания.

Александро организовал сообщество практических пользователей Scrum, т.е. группы единомышленников или людей, имеющих одинаковую специальность. В главе 4, “Движение в направлении гибкости”, мы уже сталкивались с использованием сообществ практических пользователей Scrum как основы сообщества в поддержку перехода к Scrum (Enterprise Transition Community – ETC) и сообществ усовершенствования, которые помогают организациям внедрять Scrum. Мы рассмотрим их более подробно в главе 17, “Изменение масштаба Scrum”. Помимо организации сообщества практических пользователей Scrum, в табл. 12.3 указаны другие методы изменения обменов.

Таблица 12.3. Варианты изменения обменов для влияния на способ самоорганизации команды

Подключить к обмену (или исключить из него) тех или иных сотрудников.

Формализовать или деформализовать обмен.

Изменить способ осуществления обмена (общение “тет-а-тет”, обмен письменными документами).

Изменить частоту обмена.

Теперь, после трех факторов, влияющих на способ самоорганизации команды на основе поставленной перед ней задачи, нужно рассмотреть способы, с помощью которых лидеры могут обеспечить постоянное развитие команды или компании.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- С кем из лиц, не входящих в состав данной команды, эта команда должна, по вашему мнению, общаться чаще? Есть ли какой-нибудь способ стимулировать такие обмены?
- Представьте в графическом виде интенсивность общения между членами команды. Обозначьте каждого человека кружком. Затем проведите линии между парами членов команды, общающихся между собой. Чтобы обозначить интенсивность, или частоту, таких обменов, используйте цвет или толщину соединительных линий. Не свидетельствует ли ваше графическое представление о наличии каких-либо проблем?
- Тщательно наблюдайте за командой в течение спринта. Участвуют ли в обменах все те, кто должны в них участвовать? Не должны ли какие-то из обменов включать большее (или меньшее) число людей?

Влияние на эволюцию команды или компании

Много лет назад я работал под руководством директора по информационным технологиям, Джима, который славился в своей компании частыми реорганизациями своего подразделения. В компании ходила даже такая шутка: если вам не нравится нынешняя организация подразделения Джима, просто подождите денек-другой. Разумеется, он не проводил эти реорганизации ежедневно, но у людей складывалось впечатление, что подразделение Джима пребывает в перманентном состоянии реорганизации. Реорганизации, проводившиеся Джимом, — лишь один пример, указывающий на то, что сегодняшняя компания совсем не та, какой она была еще неделю назад. Компании постоянно эволюционируют. Организационная эволюция происходит в ответ на действие внешних факторов, конкурентных сил, достоинств и недостатков работников и прочих влияний.

Эволюция является результатом трех элементов: отклонения, отбора и сохранения. Филип Андерсон (Philip Anderson) объясняет взаимосвязанность этих трех элементов на примере жирафа. Путем случайной мутации некий жираф родился с необычайно длинной шеей. Это пример *отклонения*. Такая длинная шея помогает этому конкретному жирафу дотянуться до ветвей, до которых не могут дотянуться остальные жирафы. Это обеспечивает нашему жирафу повышенную живучесть, что мы обозначаем термином *отбор*. Наконец, этот жираф передает ген необычайно длинной шеи своим потомкам, что мы обозначаем термином *сохранение* (1999, 120–1).

Организации также эволюционируют посредством отклонения, отбора и сохранения. Организации требуется отклонение (разброс) среди работников, команд, процессов и прочего, чтобы добиться определенной совокупности результатов. Кроме того, должно существовать четкое определение успеха, чтобы работники могли различать отклонения, которые ведут к желаемым результатам, и отклонения, которые к ним не приводят. В сущности, отклонение и отбор дают в итоге следующее: кто-то в организации замечает, что, “когда мы делаем больше того-то и того-то, это приводит к лучшим результатам”. Наконец, должны существовать механизмы, достаточные для того, чтобы закрепить это новое, более эффективное поведение. Если культура или кадровая политика организации вступает в противоречие с новым способом поведения, то этот новый способ поведения сохранить не удастся.

Лидеры и агенты перемен не сидят сложа руки, пока их организации эволюционируют. Напротив, они помогают направлять эволюцию своей организации в нужное русло посредством отклонения, отбора и сохранения.

Самоорганизация исходит из того, что эффективная организация эволюционирует, а не конструируется. Она нацелена на создание такой среды, в которой успешные подразделения и процедуры не только возникают, но и саморегулируются в ответ на перемены в окружении. Это происходит потому, что руководство формирует определенную атмосферу и стимулирует быстрое развитие в направлении более высокой эффективности, а вовсе не потому, что руководство в совершенстве овладело искусством планирования и мониторинга потоков работы (Anderson, 1999, 119).

Филип Андерсон предлагает семь рычагов, которыми могут воспользоваться лидеры, чтобы направлять эволюцию своей организации в нужное русло. Эти рычаги перечислены в табл. 12.4. Обсуждая модель CDE, мы уже рассказывали о таких рычагах, как *выбирайте людей* (подобно изменению контейнеров или усилинию различий) и *реконфигурируйте сеть* (подобно обменам). Остальные рычаги описаны в последующих разделах.

Таблица 12.4. Методы влияния на эволюцию организации

Выбирайте внешнее окружение.
Сформулируйте, что такое успешное функционирование.
Управляйте смыслом.
Выбирайте людей.
Реконфигурируйте сеть.
Внедряйте замещающие системы отбора.
Подзаряжайте систему энергией.

Выбирайте внешнее окружение

Самоорганизация и эволюция представляют собой реакцию на окружение, в котором работает команда. Лидеры способны оказывать значительное влияние на это окружение. Под *окружением* я понимаю не просто физическое место работы команды. Существует немало более важных факторов окружения, на которые могут оказывать влияние лидеры. Например, лидеры контролируют бизнес, которым занимается их организация, и влияют на него. Они определяют подход своей организации к инновациям (является ли данная компания новатором или относится к числу тех, которые быстрее других следуют примеру новаторов?). Кроме того, лидеры контролируют тип проектов, над реализацией которых им предстоит работать, и темп появления в организации новых проектов. Каждый из этих факторов влияет на то, как организация эволюционирует и приспосабливается к изменениям в окружающей среде.

Джулия занимала пост генерального менеджера крупного подразделения в компании, занимающейся разработкой программного обеспечения. Под ее началом трудилась примерно половина из пятисот разработчиков ее компании. Она первой начала внедрять Scrum в своем подразделении. Когда первые результаты применения Scrum

оказались вполне обнадеживающими, Джулия разработала годовой план внедрения Scrum во всех остальных командах. В рамках реализации этого плана Джулия приняла решение замедлить поток новых проектов в организацию. Дело не в том, что Scrum-команды работали медленнее — напротив, разработка велась довольно быстрыми темпами. Но уже первый опыт применения Scrum помог Джулии понять, что ее организация пытается одновременно выполнять слишком большое количество проектов. Опыт работы с первыми Scrum-командами продемонстрировал Джулии все преимущества такой организации работы, при которой каждый из сотрудников получает возможность сосредоточиться на участии в одной, максимум двух командах. Чтобы добиться этого, Джулии нужно было привести во взаимное соответствие поток новых проектов в организацию и скорость выполнения проектов.

См. также Проблема одновременного выполнения слишком большого числа проектов обсуждается в главе 10, “Структура команды” (раздел “Один человек — один проект”).

Сформулируйте, что такое успешное функционирование

Организации и организмы эволюционируют, стремясь соответствовать своей окружающей среде. Согласно принципу отбора сохраняются (будут сохранены) именно те качества, которые с наибольшей вероятностью помогут выжить в данной организации отдельному человеку или группе людей. Именно лидеры и менеджеры организации решают, какие качества помогут выжить в данной организации отдельному человеку или группе людей. Если такие качества гибкой методологии разработки, как открытость и прозрачность, обеспечивают выживание в форме продвижения по службе и публичного признания заслуг, то люди выберут для себя именно такие модели поведения.

Огромное влияние лидеров и менеджеров обусловлено тем, что именно они формируют, *что* следует понимать под успешным функционированием. Например, они определяют отношение организации к подбору подходящего баланса между краткосрочными и долгосрочными результатами. Организация, которая отдает предпочтение достижению успеха в долгосрочной перспективе, будет склонна к инвестированию в обучение и в повышение квалификации своих работников и к поддержанию стабильного темпа работы, будет охотно предоставлять работникам время для опробования новаторских идей и не пойдет на то, чтобы любой ценой (в том числе ценой создания кода, малопригодного к сопровождению) уложиться в запланированные сроки завершения работ.

Управляйте смыслом

Члены самоорганизующейся системы эволюционируют в ответ на получаемые ими послания. Эти послания могут генерироваться как внутри самой системы, так и вводиться в нее извне. Менеджеры и лидеры управляют смыслом этих посланий, предоставляя контекст, который помогает работникам правильно интерпретировать эти послания. Контекст обеспечивается в форме историй, мифов и ритуалов, повторяемых лидерами. Лидеры отбирают и рассказывают истории, сквозь призму которых работникам следует, по мнению лидеров, интерпретировать текущую ситуацию.

Вспоминаю, как директор по разработкам одной из компаний, которая должна была стать моим новым клиентом, пытался впечатлить меня следующими словами: “Обратите внимание: когда у нашей компании возникает какая-либо серьезная проблема, каждый день в 17:00 наш генеральный менеджер выходит на улицу и подсчитывает количество автомобилей на парковочной стоянке нашей компании. Он делает это до тех пор, пока компания не удастся справиться с возникшей проблемой”.

Эта история довольно быстро стала частью корпоративного фольклора. Директор по разработкам рассказывал ее для того, чтобы сотрудники компании знали, какой тип поведения приветствуется ее новым генеральным менеджером.

Я понял, что при таком генеральном менеджере внедрение Scrum в этой компании является совершенно безнадежным делом, и поэтому не смог устоять перед соблазном переформатировать эту историю так, чтобы она вырабатывала у людей поведение, на которое хотелось бы рассчитывать мне. “Замечательно, — сказал я. — Мне не терпится познакомиться с ним. Я с удовольствием познакомлюсь с любым начальником, который появляется на парковочной стоянке своей компании в 17:00, чтобы подсчитать количество сотрудников, все еще остающихся на работе, чтобы поскорее отправить их домой”. В тот день моя попытка потерпела неудачу. Встретившись впоследствии с генеральным менеджером этой компании, я потерпел еще большую неудачу. Даже после того как я перечислил все проблемы окружения, препятствующие внедрению Scrum в этой компании, генеральный менеджер не изъявил ни малейшего желания пересмотреть свои взгляды.

Внедряйте замещающие системы отбора

Главной системой отбора, которая должна управлять организационной эволюцией, является долговременный успех на рынке. Продукты, которые приносят прибыль, должны приходить на смену продуктам, которые приносят убытки; структуры команд, которые ведут к появлению прибыльных продуктов, должны приходить на смену структурам команд, которые не приводят к появлению прибыльных продуктов; а методы, которые ведут к появлению прибыльных продуктов, должны приходить на смену методам, которые не приводят к появлению прибыльных продуктов. Для этого, конечно, требуются годы и годы. Кроме того, учитывая, что в организации одновременно происходит множество изменений, невозможно полностью изолировать последствия какого-то одного изменения. Чтобы ускорить темп эволюции, руководители могут внедрять замещающие системы отбора.

Замещающая система отбора — это такой процесс отбора желательных проектов, продуктов или моделей поведения, который не задействует обратную связь рыночной системы отбора, связанную с большими затратами времени. Используемая в *Google* политика, согласно которой разработчикам разрешается тратить 20% своего рабочего времени на выбранные ими самими проекты, представляет собой замещающую систему отбора. Таковой же является используемая в *Google* политика, согласно которой разработчикам разрешается свободно и в любое время переходить из одной команды в другую и из одного проекта в другой (Yegge, 2006). Поскольку разработчики хотят работать над проектами, желательными для *Google* (т.е. успешными, новаторскими и т.д.), такие замещающие системы отбора являются весьма эффективными: они отбирают за относительно короткое время проекты, продукты или модели поведения, которые рынок признал бы желательными лишь по истечении достаточно длительного

времени. Новые проекты, которым не удается привлечь внимания, быстро утрачивают свою актуальность. Или, в эволюционном контексте, такие проекты не будут отобраны и сохранены.

Замещающие системы отбора встречаются в организациях очень часто. Во многих организациях используется система, согласно которой один сотрудник может выдвигать кого-либо из своих товарищей по работе на получение небольшой денежной премии. Подобные вознаграждения, если только они не используются для придания индивидуальным достижениям большей важности по сравнению с командными достижениями, могут быть полезны как демонстрация типа поведения, желательного для данной организации.

К сожалению, не все замещающие системы отбора являются надежными предсказателями поведения, которое выбрал бы рынок. Менеджеры должны с осторожностью внедрять те или иные замещающие системы отбора. Джеймс, вице-президент по разработке, отнесся без должного внимания к одной из замещающих систем отбора, используемых в его организации, — системе поощрения. Хаос был его стихией: Джеймсу всегда были нужны чрезвычайные ситуации, поиском выхода из которых ему нравилось заниматься. Если чрезвычайные ситуации не возникали сами по себе, Джеймс создавал их искусственно. Он собаку съел на преодолении всевозможных трудностей и поощрял других сотрудников, обладавших этим качеством. Подчиненные Джеймса хорошо усвоили, что их начальник ценит в людях умение выходить из чрезвычайных ситуаций выше, чем умение избегать таковых.

Подзаряжайте систему энергией

Командам и организациям требуется энергия. Если в команду или организацию не будет “закачиваться” энергия, они начнут страдать от энтропии. Вдохновляя работников и ставя перед ними задачи, менеджеры и лидеры обеспечивают приток энергии, которая поддерживает самоорганизацию и эволюцию. Задачи создают “зазоры” между текущим состоянием продукта и желательным его состоянием или между текущими и желательными показателями функционирования группы. Когда группа людей с готовностью и воодушевлением берется за реализацию той или иной задачи, она самоорганизуется вокруг способа ее реализации.

В своей книге *Teamwork* Ларсон (Larson) и Лафасто (LaFasto) обращают особое внимание читателей на важность постановки перед командой “четкой, вдохновляющей цели” (1989, 27). В книге *Hot Spots* Линда Граттон (Lynda Gratton) приходит к аналогичному выводу, утверждая, что высокопроизводительным командам требуется “воодушевляющая задача” (2007, 3). В своей памятной записке “Internet Tidal Wave” (май 1995 года) Билл Гейтс сформулировал для всей компании *Microsoft* воодушевляющую задачу. Описав, как Интернет способен изменить продукцию и весь бизнес компании *Microsoft*, Билл Гейтс формулирует четкую, вдохновляющую цель, заявляя, что *Microsoft* должна “сначала воспользоваться всеми возможностями, которые предоставляет Интернет, а затем расширить эти возможности”. В заключение он предлагает еще одну мотивацию.

Интернет — это волна увлечения. Интернет изменяет правила. Интернет — это невероятная возможность и невероятная проблема. Я ожидаю от вас предложений

о том, как нам следует усовершенствовать нашу стратегию, чтобы продолжить победное шествие нашего невероятного успеха.

В книге *Agile Project Management* Джим Хайсмит (Jim Highsmith) подчеркивает, насколько важно начинать каждый проект со своего рода хартии — краткого и хорошо запоминающегося разъяснения того, *зачем* реализуется данный проект или *что* должно стать его результатом. Правильно сформулированная и хорошо запоминающаяся хартия проекта может дать членам команды четкую, вдохновляющую цель и поставить перед ними воодушевляющую задачу. Джим Хайсмит предлагает три метода формулирования хартии проекта.

- Составить краткое (одно-два предложения) резюме соответствующего проекта или продукта (“духоподъемная формулировка”)
- Придумать конструкцию коробки, в которой будет поставляться соответствующий продукт (даже если этот продукт никогда не будет поставляться в коробке)
- Составить описание продукта, занимающее одну страницу (2009)

Помимо этих инструментов формулирования хартии проекта, я время от времени использую еще два.

- Составить воображаемый пресс-релиз, которым вы хотели бы сопроводить выпуск своего продукта
- Составить обзор данного продукта, который, будь такая возможность, можно было бы опубликовать в журналах

Один из моих клиентов с успехом воспользовался вариантом “обзор в журнале”. Этот клиент занимается разработкой программного обеспечения, выявляющего “шпионские” программы, и недавно стал лауреатом премии “Продукт года” (второе место в этой номинации), учрежденной одним из отраслевых журналов. Для многих продуктов занять второе место в своей категории является несомненным и весьма желанным успехом. Например, фильм, занявший второе место по величине кассового сбора за год, пользуется, наверное, не меньшей известностью, чем фильм, занявший первое место. Но для продукта, который большинство пользователей покупают только однократно, оказаться на втором месте не так уж почетно.

Я посоветовал Scrum-мастеру этой команды, Эрину, привлечь всех членов своей команды к составлению хартии для новой версии этого продукта путем написания обзора, который им самим хотелось бы увидеть в каком-либо из журналов. Они составили такой обзор. Затем этот гипотетический обзор был выведен на самых видных местах в помещении, где работала команда. Через шесть месяцев была выпущена новая версия этого продукта, а его обзор был опубликован в том же самом журнале, который присудил второе место предыдущей версии продукта. На этот раз продукт получил приз “Выбор редакции” как лучший продукт в серии программного обеспечения, выявляющего “шпионские” программы. Столь высокое достижение команды частично объяснялось и тем, что Scrum-мастеру удалось внести в систему энергию в форме четкой, вдохновляющей цели, которая привела к формулированию воодушевляющей задачи.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Составьте перечень всех замещающих систем отбора в своей организации. Какие формальные и неформальные механизмы определяют или влияют на то, какие именно проекты, типы поведения, подходы и тому подобное получат право на жизнь в будущем? Не противоречат ли внедрению Scrum какие-либо из этих механизмов? Что вы можете сделать для устранения таких механизмов и чья помощь может вам понадобиться для этого?
- В достаточной ли степени заряжена энергией ваша команда? Если нет, составьте хартию проекта, воспользовавшись одним из описанных выше методов.
- Выявите всех, кто не является членами вашей команды, но фактически посылает ей те или иные послания. Есть ли у команды надлежащий контекст, в котором она могла бы правильно интерпретировать эти послания? Можете ли вы предотвратить получение командой противоречивых посланий, особенно если в них указывается на конфликтующие между собой цели проекта, такие как качество проекта, его масштаб и календарный план?

Лидерство — это не только покупка пиццы

Наблюдая матчи теннисистов, вы, наверное, обращали внимание на то, что игрок, принимающий подачу, опирается не на всю ступню, а лишь на переднюю ее часть. Это позволяет ему подготовиться к любому варианту подачи, слева или справа, длинной или короткой. Лидерам и агентам перемен, участвующим в переходе организации к использованию Scrum, нужно, чтобы их организация всегда находилась в таком же состоянии полной готовности, т.е. была готова в любой момент броситься влево или вправо, вперед или назад. Организация, постоянно пребывающая в состоянии готовности, способна молниеносно реагировать на любые перемены. Она привыкает к постоянным переменам, которые становятся неотъемлемой формой ее существования. Никакие перемены не являются для такой организации неожиданностью, а потому она достаточно быстро к ним приспосабливается.

Лидеры, менеджеры и агенты перемен держат организацию в тонусе, регулируя ее контейнеры, различия и трансформирующие обмены. Лидеры влияют на направление, в котором эволюционирует организация, нажимая на какой-либо из семи рычагов Андерсона.

Для самоорганизации требуется нечто большее, чем покупка пиццы и отказ от навязчивого контроля за деятельностью команды. Существуют малозаметные и косвенные способы, посредством которых лидеры могут оказывать влияние на свои команды. Лидер не в состоянии точно предсказать, как его команда будет реагировать на то или иное изменение, какова бы ни была природа этого изменения (например, изменение контейнера, принятие новых стандартов оценки результатов работы, замещающая система отбора или что-либо другое). У лидера нет ответов на все вопросы. Однако он может воодушевить организацию на повышение своей гибкости и динамичности.

Дополнительная литература

Anderson, Philip. 1999. Seven levers for guiding the evolving enterprise. В книге *The biology of business: Decoding the natural laws of enterprise*, ed. John Henry Clippinger III, 113–152. Jossey-Bass.

Это одна из лучших глав в замечательной книге *The biology of business: Decoding the natural laws of enterprise*. В ней излагаются принципы организационной эволюции и описываются семь рычагов, речь о которых шла в разделе “Влияние на эволюцию команды или компании” данной главы.

Goldstein, Jeffrey. 1994. *The unshackled organization: Facing the challenge of unpredictability through spontaneous reorganization*. Productivity Press.

Это одна из первых книг, посвященных самоорганизации внутри корпораций. К числу самых интересных материалов в этой книге я отнес бы многочисленные случаи из практики самоорганизации разных компаний (как с указанием, так и без указания названий конкретных компаний).

Olson, Edwin E., and Glenda H. Eoyang. 2001. *Facilitating organization change: Lessons from complexity science*. Pfeiffer.

В основу этой превосходной книги положены идеи из докторской диссертации Гленды Эоянг, посвященной самоорганизации. Эти идеи автор применяет к организационным изменениям. В книге представлена модель, согласно которой на организационные изменения можно влиять посредством контейнеров, различий и обменов, целенаправленно создающихся или стимулирующихся агентами перемен соответствующей организации.

Глава 13

Журнал запросов на выполнение работ

Самый большой вопрос, который возникает в начале реализации любого проекта, — “Что именно мы создаем?” Мы представляем себе общую форму системы, которую нам предстоит создать. Нам может быть известно, например, что мы создаем текстовый редактор. Но всегда есть “темные углы”, которые нам еще предстоит исследовать, или вопросы о том, как будут работать те или иные функциональные возможности, — и эти вопросы нам также предстоит уладить. Будет ли наш текстовый редактор включать, например, такую функциональную возможность, как интерактивное построение таблиц, или таблицы будут строиться путем ввода тех или иных параметров в ряде диалоговых окон?

В случае процесса последовательной разработки мы пытаемся начать с продолжительной фазы предварительного сбора требований, в течение которой, как предполагается, соответствующий продукт будет полностью специфицирован. Идея в данном случае заключается в том, что, тщательно и неспешно обдумав все детали в самом начале проекта, можно рассчитывать на то, что во время основной фазы разработки проекта никакие “темные углы” нам уже не встретятся.

Scrum-команда сознательно отказывается от продолжительной фазы предварительного сбора требований, отдавая предпочтение подходу “just-in-time” (“в нужный момент”). Общие описания функциональных возможностей можно составить заблаговременно, но поначалу это будут описания лишь общего характера, которые впоследствии будут постепенно уточняться по мере выполнения проекта. Все они документируются в *журнале запросов на выполнение работ*, который представляет собой перечень всех требуемых функциональных возможностей, еще не реализованных в соответствующем продукте. Журнал запросов на выполнение работ ведется владельцем продукта и упорядочен согласно приоритетам функциональных возможностей (именно поэтому журнал запросов на выполнение работ иногда называют *приоритизированным перечнем функциональных возможностей*). В отличие от традиционного

документа, в котором отображаются требования к продукту, журнал запросов на выполнение работ является очень динамичным документом: отдельные его пункты добавляются, удаляются и повторно приоритизируются в ходе каждого спринта по мере расширения и углубления наших знаний о продукте, пользователях, команде и т.п.

В этой главе мы рассмотрим три изменения, которые должны произойти в организации с целью эффективного использования журнала запросов на выполнение работ. Во-первых, мы рассмотрим необходимость перехода от письменного описания функциональных возможностей продукта к их обсуждению. Во-вторых, мы узнаем, почему так важно наращивать подробности постепенно, по ходу выполнения проекта, вместо того чтобы задокументировать их с самого начала. В-третьих, мы выясним, почему “спецификация на примере” должна быть для команды предпочтительным подходом к документированию функциональных возможностей продукта. Завершается глава расшифровкой аббревиатуры DEEP, который позволяет легче запомнить важнейшие атрибуты журнала запросов на выполнение работ.

Переход от документов к обсуждениям

Существует распространенный миф о требованиях: если вы формулируете их в виде бумажных документов, то пользователи получают именно то, что им нужно. Это неправильно. При таком подходе пользователи в лучшем случае получат именно то, что было задокументировано (что, однако, может оказаться не совсем тем (или совсем не тем), что им было нужно на самом деле). Слова, изложенные на бумаге, вводят в заблуждение: они кажутся более точными, чем в действительности. Например, недавно я хотел провести трехдневный курс обучения для всех желающих. Я уже обсудил этот вопрос вместе со своей помощницей и потому решил отправить ей по электронной почте сообщение “Пожалуйста, забронируйте мне в Денвере номер в гостинице «Hyatt», а также напомнил ей о сроке моего пребывания в Денвере. На следующий день она ответила мне по электронной почте сообщением такого содержания: “Номер в гостинице уже забронирован”. В ответ я поблагодарил ее и сразу же переключился на другие вопросы.

Примерно через неделю она еще раз обратилась ко мне по электронной почте: “Номер в гостинице забронирован на указанные вами дни. Сообщите, пожалуйста, что еще требуется от меня. Может быть, вы хотите поселиться в какой-то другой гостинице Денвера? На другой неделе? В другом городе?” Оказывается, мы по-разному поняли смысл слова “забронирован”. Ответив мне “Номер в гостинице уже забронирован”, она имела в виду, что номер, который мы обычно снимаем в гостинице “Hyatt”, уже кем-то забронирован. Когда я прочитал ее послание “Номер в гостинице уже забронирован”, я понял это как подтверждение, что моя помощница действительно забронировала номер в гостинице, как я и просил. В ходе этого обмена сообщениями по электронной почте ни я, ни она не допустили никакой ошибки. Скорее, это можно рассматривать как пример того, насколько легко можно запутаться, если положиться исключительно на письменную фиксацию своих соображений. Если бы мы с ней общались, например, по телефону, а не по электронной почте, то, после того как она ответила мне: “Номер в гостинице уже забронирован”, я наверняка поблагодарил бы ее. По моему удивлению тону помощница поняла бы, что произошла какая-то путаница, и попыталась бы внести ясность. В результате нам удалось бы тотчас же устранить это недоразумение.

Помимо проблем такого рода, существуют и другие причины, которые заставляют нас отдавать предпочтение обсуждениям, а не документированию.

Письменные документы могут заставить вас полностью положиться на то, что в них написано. Информация, представленная в письменном виде, выглядит более официальной, формальной и “окончательной”, особенно в случае использования впечатляющего форматирования. Недавно один из моих клиентов, в офисе которого мне приходилось бывать неоднократно, решил назначить мне встречу, так сказать, на “нейтральной территории”. Он отправил мне по электронной почте подробное описание, как добраться от гостиницы, в которой я проживал, до загородного клуба, где должна была состояться наша встреча.

- Выйдя из гостиницы, повернуть налево, на North Commerce Parkway, и пройти 0,4 мили
- Повернуть налево, на SW 106th Avenue, и пройти 0,2 мили
- Повернуть направо, на Royal Palm Boulevard, и пройти 1,1 мили
- Повернуть налево, в сторону Town Center

Но я не мог “поворнуть налево, к Town Center”! Пройдя 1,1 мили по Royal Palm Boulevard, я оказался на каком-то перекрестке, но в сторону Town Center можно было пройти, лишь повернув направо. Мне было сказано повернуть налево, однако налево вела дорога под названием Weston Hills Boulevard. Я мог увидеть нужный мне загородный клуб слева и мне казалось, что именно туда я и должен был повернуть. Однако инструкции моего клиента были предельно конкретны и к тому же совершенно правильно вывели меня на перекресток, где меня и начали терзать сомнения. Поэтому я решил не отступать от инструкции и прошагал еще две мили, наблюдая, как загородный клуб проплывает мимо меня слева, оставаясь позади. В конце концов я понял, что в инструкции была допущена ошибка. Я повернулся на 180° и направился по Weston Hills Boulevard, а не в сторону Town Center, как было указано в инструкции. А ведь вместо того, чтобы составлять эту подробную письменную инструкцию, мой клиент мог бы просто сказать мне: “Направляйтесь в сторону моего офиса, как вы обычно делаете. Но когда увидите загородный клуб, поверните налево. Я не помню названия этой улицы, но мимо загородного клуба вы наверняка не пройдете”.

При наличии письменного документа мы не имеем возможности постепенно уточнять смысл, как это обычно происходит при обсуждении. Несколько лет назад я изучал документ с требованиями, который содержал описание интерфейса, подобного Windows Explorer, для управления папками данных. Одно из требований гласило: “Название папки может состоять из 127 символов”. Я был уверен, что суть этого требования такова: “Название папки может состоять *максимум* из 127 символов”. Но речь шла о приложении, касающемся биоинформатики, и среди требований встречались довольно необычные (например, текстовые поля, которые могут содержать лишь буквы A, C, G и T). Название папки, состоящее именно из 127 символов, было для меня несколько неожиданным, но, учитывая специфику данного приложения, ничего нельзя было исключать.

Итак, поскольку в требованиях была указана конкретная длина названия папки, я исходил из того, что для этого должна быть какая-то разумная причина. Такой причины не могло не быть. К тому же сама природа документа, содержащего требования, давала мне гораздо меньше оснований подвергать сомнению это “требование 127”,

чем в случае, если бы я обсуждал это требование непосредственно со сформулировавшим его аналитиком. Если бы такое обсуждение состоялось, то я мог бы задать аналитику ряд вопросов, начинающихся, например, словами “Итак, вы утверждаете, что...”, “Если я правильно понял вас, это означает, что...” или “Не следует ли из этого, что...” Ответы на эти вопросы должны были бы подтвердить, что “передача понимания” произошла и я правильно понял своего собеседника. Именно такое постепенное уточнение смысла невозможно в случае использования письменных документов.

См. также Ответственность команды в целом и проблемы, вызванные передачей работы из рук в руки, обсуждались в главе 11, “Организация коллективного труда”.

В случае использования письменных документов снижается ответственность команды в целом. Одна из причин перехода к использованию Scrum заключается в том, чтобы заставить всю команду работать во имя достижения общей цели — создания великолепного продукта. Нам нужно избавить свой процесс разработки от плохих привычек, которые мешают достижению цели. Письменные документы приводят к последовательным передачам работы из рук в руки, что лишает команду такого важного качества, как единство цели. Один человек (или группа людей) описывает продукт, другой (или группа людей) создает его. Налаживание двустороннего обмена информацией лишается смысла. При наличии письменного документа кто-то из членов команды говорит: “Вот что мы должны сделать”, а остальным остается лишь это сделать. Такой тип отношений (“начальник — подчиненный”) вряд ли способен создать у подчиненных сильное чувство личной причастности и ответственности за конечный продукт. Вместо того чтобы чувствовать себя ответственными за успех конечного продукта, они чувствуют себя ответственными за исполнение того, о чем написано в документе. Обсуждения ведут к противоположному эффекту: участие в обсуждении всей команды приводит к появлению у каждого из ее членов чувства большей заинтересованности и ответственности за успех конечного продукта.

ВОЗРАЖЕНИЕ

“Я не могу избавиться от всех документов — мой проект содержит требования ISO 9001 (и подобные им), поэтому все должно быть задокументировано и зафиксировано, чтобы можно было легко найти нужную информацию”.

Как будет указано в следующем разделе, избавляться от всей документации нет необходимости. Избавьтесь от документации, которая не является абсолютно необходимой для вас, и сократите до минимума объем документации, без которой вы не можете обойтись (еще лучше, если такую документацию можно генерировать автоматически). Важно также отдавать себе отчет в том, что документирование может осуществляться главным образом для тех, кто захочет ознакомиться с соответствующим проектом впоследствии (например, через несколько лет), тогда как в ходе выполнения проекта вы можете полагаться в основном на обсуждения.

Не выплесните ребенка вместе с документацией

Перечисленные выше недостатки письменного общения вовсе не означают, что нужно вообще отказаться от изложения требований в письменном виде. Отнюдь! Скорее документами нужно пользоваться лишь в случаях, когда это оправданно. Поскольку Agile Manifesto гласит, что предпочтение нужно отдавать “работоспособному программному обеспечению, а не исчерпывающей документации” (Beck et al., 2001), многие пришли к неправильному выводу о том, будто использование гибкой методологии разработки ведет к полному отказу от документирования. Цель гибкой методологии разработки заключается в том, чтобы найти оптимальный баланс между документированием и обсуждением. В прошлом очень часто наблюдался чрезмерный крен в сторону документирования.

Кроме того, нужно отдавать себе отчет в том, что документы, содержащие перечень требований, — это лишь одна из форм документирования, которые могут использоваться при выполнении проекта. Вместе с тем при выполнении проекта применяются и другие артефакты: планы тестирования, выполняемые контрольные примеры и даже код, документирующий поведение (или желательное поведение) системы.

См. также В разделе “Учитесь начинать без спецификации”, приведенном далее в этой главе, будут продемонстрированы преимущества специфицирования поведения в контрольных примерах.

Поскольку код и автоматизированные контрольные примеры будут создаваться с целью разработки определенного продукта, опытной Scrum-команде предстоит научиться полагаться на них в максимальной степени. Она дополнит эти формы документирования каким-либо письменным документом с перечнем требований в той мере, в какой этот документ может оказаться полезным или может потребоваться для регуляторных, договорных или юридических целей. Письменный документ с перечнем требований по-прежнему будет полезен для многих проектов. Том Поппендик (Tom Poppendieck), соавтор ряда книг по экономной разработке программного обеспечения, писал, что, “когда документы необходимы главным образом для того, чтобы облегчить передачу работы из рук в руки, такие документы являются злом. Когда же в них зафиксировано обсуждение, которое обязательно нужно сохранить в человеческой памяти, такие документы являются благом”.

Используйте в журнале запросов на выполнение работ истории пользователей

Истории пользователей — наилучший способ перенести акцент с письменного изложения функциональных возможностей на их устное обсуждение. История пользователя — это краткое, простое описание той или иной функциональной возможности, изложенное с точки зрения человека, которому эта новая функциональная возможность необходима (как правило, таким человеком является пользователь или клиент соответствующей системы). Истории пользователей нередко записываются на специальных учетных карточках или клейких отрывных листках, хранятся в коробке из-под обуви и развешиваются на стенах (или раскладываются на столах) для использования

в ходе планирования и обсуждения. Как таковые истории пользователей переносят акцент с письменного изложения функциональных возможностей на их устное обсуждение. Истории пользователей, как правило, следуют простому шаблону:

Будучи <тип пользователя>, я хочу <конкретная цель>, чтобы <конкретная причина>.

ПРИМЕЧАНИЕ

Возможны и другие шаблоны. Например, приведенный ниже шаблон отличается тем, что на первое место в нем выносится ценность соответствующей истории пользователя: *для того чтобы <важная цель>, будучи <тип пользователя>, я хочу <конкретная цель>*. Используя в свое время оба эти формата, я все же отдаю предпочтение первому варианту. Если читателя интересуют конкретные причины такого предпочтения, обратитесь по адресу <http://blog.mountangoatsoftware.com/advantages-of-the-as-a-user-i-want-user-story-template>. Важен, однако, не столько формат письменной части истории пользователя, сколько обсуждения, вызванные этой историей.

Истории пользователей могут храниться с помощью какого-либо программно-инструментария (существует немало причин, заставляющих вас выбрать именно такой вариант), но по мере возможности я предпочитаю записывать их на простых учетных карточках размером 3×5 дюймов. Несмотря на то что истории пользователей часто записываются на учетных карточках или клейких отрывных листках бумаги, текст, записанный на них, представляет собой лишь начало. Такая карточка с историей пользователя вовсе не предназначена служить полным описанием соответствующей функциональной возможности, как мы привыкли видеть в спецификациях требований к программному обеспечению, наподобие “Данная система должна...” Скорее всего, такая карточка с историей пользователя выполняет роль своего рода договора между командой разработчиков и владельцем продукта. Члены команды обещают поговорить с владельцем продукта до того, как начнут работать над соответствующей историей, а владелец продукта обещает выслушать команду, когда она будет готова к разговору с ним.

Обещание команды поговорить с владельцем продукта до того, как она приступит к работе над соответствующей историей, очень важно, поскольку это позволяет владельцу продукта не заботиться о том, чтобы на карточке было записано все, до мельчайших подробностей. По сути, это является одной из причин использования в этом важном деле столь легкомысленного и несерьезного носителя, как бумажные карточки. Они служат постоянным напоминанием о том, что такая карточка вовсе не обязательно должна содержать все подробности. Эти подробности всплывут в ходе обсуждений между владельцем продукта и командой.

Ответное обещание владельца продукта выслушать команду, когда она будет готова поговорить с ним, также важно, поскольку это позволяет команде принять работу в спринт, не рассматривая всех подробностей (что было бы невозможно в любом случае). Владельцу продукта необязательно постоянно находиться рядом с командой (хотя это было бы полезно и привело бы к повышению производительности). Вместо этого владелец продукта обещает быть настолько доступным для команды, чтобы не потребовалось за две недели планировать обычный телефонный звонок.

ВОЗРАЖЕНИЕ

“У меня нет возможности излагать требования на учетных карточках”.

Ну и прекрасно! Проекты, предусматривающие использование рассредоточенных команд и очень крупных команд, а также требования отслеживания или подобные им нередко предполагают применение того или иного программного инструмента. Хороший инструмент может улучшить планирование продукта верхнего уровня, обсуждение сценариев типа “а что если...” и широкое общение. Однако командам, использующим вместо пера и бумаги программные инструменты, придется труднее при внедрении Scrum, т.е. при выполнении перехода от документов к обсуждению. Команда, использующая тот или иной программный инструмент, имеет гораздо больше шансов попасть в одну из многочисленных опасных ловушек.

- Составление чрезмерно длинных описаний функциональных возможностей
- Работа над изучением потребностей пользователей лишь части команды (бизнес-аналитиков)
- Сопротивление необходимости дробления историй пользователей, с тем чтобы в течение каждого спринта можно было реализовать историю полностью
- Цепляние за истории, необходимость в которых уже отпала, поскольку иногда их легче выполнить, чем удалить из базы данных

ВОЗРАЖЕНИЕ

“Мы разрабатываем серверное программное обеспечение, к которому пользователь не имеет никакого доступа, поэтому истории пользователей в нашем случае не имеют смысла”.

Слово *пользователь* в историях пользователей заставляет людей воспринимать этот подход как имеющий более ограниченный круг применения, чем это есть на самом деле. Истории пользователей успешно применяются во всех сферах. История “будучи системой авторизации ссуд, я хотела бы принимать все данные в виде корректного, правильно структурированного XML, чтобы мне не приходилось проверять синтаксис” является совершенно приемлемой. Кроме того, хотя я и отдаю предпочтение написанию историй в формате “Будучи <тип пользователя>, я хочу <конкретная цель>, чтобы <конкретная причина>”, этот формат может подходить не для всех проектов. Если такой синтаксис не подходит для вашего проекта, составляйте журнал запросов на выполнение работ в каком-либо другом формате. Например, в свое время я с успехом применял синтаксис

<действие> <результат> <объект>

Ниже приведены примеры возможного использования такого синтаксиса.

- Оценить риск предоставления ссуды
- Разрешить снятие определенной денежной суммы с банковского счета
- Активизировать световой сигнал “Обслужить сейчас” на табло

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если у вас еще нет журнала запросов на выполнение работ или если он представлен не в лучшем виде, составьте его. Организуйте собрание всех участников проекта и совместными усилиями составьте журнал запросов на выполнение работ с использованием учетных карточек. Напомните участникам собрания, что текст на карточке выполняет роль обещания провести в будущем обсуждение; никаких подробностей в этом случае не требуется.
- Распечатайте все документы, которые были составлены при выполнении предыдущего проекта (или просто типичного проекта). Обсудите со всеми участниками проекта, сколько времени понадобилось для составления и ведения каждого из этих документов, использовался ли он впоследствии и что случилось бы (хорошего или плохого), если бы он не был составлен. По результатам обсуждения составьте перечень документов, которые будут полезны для текущего проекта, и перечень документов, от которых можно отказаться.
- Если в настоящее время вы используете какой-либо программный инструмент для ведения журнала запросов на выполнение работ и журнала спринтов, откажитесь от него хотя бы на два спринта. В конце запланированного количества спринтов воспользуйтесь ретроспективой, чтобы обсудить результат своего эксперимента. Проверьте, можно ли вообще отказаться от использования данного программного инструмента или хотя бы сократить степень его использования, перенеся акцент на обсуждение или на использование письменных документов.
- В ходе ретроспективы вашего следующего спринта попросите членов команды написать на листе бумаги программный инструмент, от использования которого они готовы отказаться. Когда они выполнят вашу просьбу, огласите полученные результаты всем присутствующим. Если названия каких-либо программных инструментов упоминались особенно часто, обсудите возможность отказа от их использования.

Постепенное уточнение требований

Начиная новый проект, его участники хорошо помнят трудности, с которыми им пришлось столкнуться при выполнении предыдущего проекта. Размышление над этими трудностями, как правило, приводит к заключению, что, “если бы мы работали усерднее или сделали то-то и то-то по-другому, результат оказался бы намного лучше”. Иногда такие выводы представляются вполне оправданными, но в случае сбора информации о требованиях это чаще всего не отвечает действительности. Как бы долго и упорно мы ни работали в самом начале проекта над выяснением всех желаемых функциональных возможностей, мы не можем рассчитывать на успех. Всегда остается много такого, что даже не приходит в голову пользователям и разработчикам до тех пор, пока разрабатываемая система не начнет принимать более или менее ясные очертания.

Спонтанно возникающие требования

Функциональные возможности, которые невозможно указать заранее, называются *спонтанно возникающими* или просто *спонтанными требованиями* (*emergent requirements*). Когда кто-либо выявляет то или иное спонтанно возникающее требование, он обычно объявляет об этом другим членам команды: “Наблюдая это, я прихожу

к выводу, что...” или “Это привело меня к мысли, что...”, или даже “Черт возьми, нам никогда не приходило в голову, что...” Всегда обнаруживается что-то такое, что приходит нам в голову только после того, как мы создадим, хотя бы в первом приближении, требуемое программное обеспечение. Одна из причин того, что столь сильный акцент в Scrum сделан на разработке работоспособного кода к моменту завершения каждого спринта, заключается в стремлении искусственно создать ситуацию, при которой спонтанно возникающие требования выявлялись бы как можно раньше.

Спонтанно возникающие требования типичны для каждого нетривиального проекта. Увы, они могут приводить к определенным проблемам. Например, они делают невозможным составление идеальных календарных планов. Кроме того, фаза предварительного проектирования всегда будет несовершенной, потому что разработчики не могут учитывать спонтанно возникающие требования до тех пор, пока они действительно не возникнут — совершенно спонтанно.

При использовании процесса последовательной разработки руководители проекта пытаются учитывать возможность появления спонтанно возникающих требований путем включения в план так называемых “буферов непредвиденных обстоятельств” и затрачивания значительных усилий на упреждающее управление рисками. Когда возникает какое-либо из таких спонтанных требований, это рассматривается как очередное упущение при составлении календарного плана. Напротив, любая Scrum-команда исходит из того, что спонтанные требования обязательно будут возникать в ходе выполнения проекта, как бы тщательно члены команды ни составляли свои планы. Вместо того чтобы рассматривать появление тех или иных спонтанных требований как упущение при составлении календарного плана, они рассматриваются либо как результат слишком раннего составления плана, либо как результат составления слишком подробного плана.

Имея дело со спонтанно возникающими требованиями, прежде всего нужно исходить из того, что предусмотреть все просто невозможно. Если исходить из того, что какие-то из требований обязательно будут возникать в ходе выполнения проекта, то нам будет легче свыкнуться с мыслью о том, что нет никакой необходимости заранее составлять некий идеальный документ с полным перечнем требований к разрабатываемой нами системе. В самом деле, вместо того чтобы стремиться к этой степени совершенства, не лучше ли указать функциональные возможности с разными уровнями точности, исходя из того, когда именно мы будем работать над той или иной функциональной возможностью.

ВОЗРАЖЕНИЕ

“Я понимаю, что ситуация может изменяться и что могут возникать новые требования. Но мне нужно указать все требования в самом начале проекта, поскольку эти требования должны стать частью договора между заказчиком и исполнителем”.

Как бы нам ни хотелось перечислить все требования в договоре, мы не сможем этого сделать. Мы можем, конечно, сделать вид, будто все требования перечислены в договоре и все они останутся неизменными, но мы должны отдавать себе отчет в том, что какие-то из этих требований наверняка изменятся. Хороший договор отличается от плохого тем, что отражает этот факт или по крайней мере признает, что изменения действительно будут иметь место. Вот что пишет по этому поводу Тронд Педерсен (Trond Pedersen): “Нарекать на возможность изменения требований — это все равно, что нарекать на плохую погоду. Мы не можем заставить окружающий мир жить по нашим законам, но мы можем найти способ жить в согласии с законами этого мира. Не умоляйте Тора [бога грома у викингов] прекратить дождь, лучше возьмите с собой зонтик”.

Айсберг журнала запросов на выполнение работ

К счастью, не так уж сложно составить такой журнал запросов на выполнение работ, который содержал бы перечень функциональных возможностей с разными уровнями детализации. Элементы журнала запросов на выполнение работ, над которыми команда намерена работать в ближайшее время, должны быть известны с достаточной степенью детализации для того, чтобы каждый из них можно было программировать, тестировать и интегрировать в рамках отдельно взятого спринта. Это приводит к тому, что истории пользователя в верхней части журнала запросов на выполнение работ окажутся достаточно компактными, но вполне понятными. Истории пользователя, расположенные ниже, будут более пространными и понятны с меньшими подробностями. Эти *эпические* истории пользователя оставляются большими, зачастую известными лишь с такой степенью детализации, которая достаточна для того, чтобы каждую из них можно было приблизительно оценить и затем приоритизировать. В результате журнал запросов на выполнение работ принимает форму айсберга, как показано на рис. 13.1.



Рис. 13.1. Айсберг журнала запросов на выполнение работ

На вершине айсберга журнала запросов на выполнение работ располагаются небольшие функциональные возможности, которые команда может полностью реализовать в рамках одного спринта. Если опуститься к подножию айсберга журнала запросов на выполнение работ (а следовательно, заглянуть в более отдаленное будущее), то можно заметить, что с приближением к подножию элементы становятся все более крупными. Команда не имеет представления о том, что скрывается под водой. Можно лишь предположить, что там находятся функциональные возможности, которые команда еще даже не обсуждала.

Уход за журналом запросов на выполнение работ

По мере разработки элементов и их удаления с вершины журнала запросов на выполнение работ вершина айсберга утрачивает свою остроконечную форму и принимает форму плоской горизонтальной площадки. Чтобы противодействовать этому явлению, нужно время от времени заниматься *уходом за журналом запросов на выполнение работ* (*grooming the product backlog*). Под уходом за журналом подразумевается вовсе не его “причесывание”¹. У большинства журналов запросов на выполнение работ, как

¹ В переводе с английского “grooming” означает “вычесывание лошади”. — Примеч. пер.

и у меня, нет волос. Таким образом, речь в данном случае идет именно об “ходе”, а не о “причесывании”.

Опыт работы эффективных Scrum-команд показывает, что примерно десять процентов времени каждого спринта нужно затрачивать на уход за журналом запросов на выполнение работ при подготовке к будущим спринтам. Эту работу может выполнять кто-то из членов команды (например, аналитик), чья роль в команде сосредоточивается главным образом на журнале запросов на выполнение работ. Возможен и вариант, при котором работа по уходу распределяется между всеми членами команды.

Обсуждения журнала запросов на выполнение работ не являются одномоментным мероприятием: они могут происходить в любое время, а участие в них могут принимать все члены команды.

Именно обсуждение историй пользователей дает разработчикам возможность понять, что им предстоит создать. Обсуждению историй пользователей — до собраний, посвященных планированию, во время собраний, посвященных планированию, и после собраний, посвященных планированию — должен способствовать Scrum-мастер (Davies and Sedley, 2009, 75).

Ваша цель должна заключаться не в том, чтобы начинать каждый спринт с идеального уяснения элементов журнала запросов на выполнение работ, которые предстоит разработать во время данного спринта. Эффективной Scrum-команде не требуется идеального уяснения функциональной возможности, к реализации которой она приступает. Скорее, в начале спринта разрабатываемую функциональную возможность следует уяснить лишь в той мере, в какой это требуется для того, чтобы у команды было достаточно шансов завершить ее реализацию к окончанию спринта. Вместо того чтобы пытаться заранее уяснить все функциональные возможности, мы предлагаем такой подход к пониманию функциональных возможностей в журнале запросов на выполнение работ, который можно было бы обозначить как “в нужный момент и не более того, без чего совершенно невозможно обойтись” (just-in-time, just-enough). Крупные функциональные возможности расчленяются на составные части, а к малым функциональным возможностям подробности добавляются “в нужный момент”, по мере того как они поднимаются к вершине айсберга журнала запросов на выполнение работ. Каждая из функциональных возможностей описывается именно с той степенью детализации, которая необходима для ее реализации командой в течение спринта.

См. также Предварительный просмотр журнала запросов на выполнение работ осуществляется аналитиками, проектировщиками пользовательского интерфейса и другими исполнителями, обладающими соответствующей квалификацией. Как это делается, описано в главе 14, “Спринты”.

Я вовсе не хочу сказать, что команда не может принять решение уяснить элементы вплоть до самого основания айсберга журнала запросов на выполнение работ. Более того, иногда это просто необходимо. Если команда полагает, что какой-либо из элементов, расположенных ближе к основанию айсберга журнала запросов на выполнение работ, может оказывать влияние на элементы, расположенные выше, она может потратить какое-то время на уяснение этого элемента. Зачастую это приводит к расщеплению такого элемента на ряд более мелких элементов журнала запросов на

выполнение работ. Однако учитывая наш прошлый опыт, когда мы отдавали предпочтение заблаговременному уяснению всех функциональных возможностей, команда должна убедиться в реальной необходимости лучшего уяснения того или иного элемента, прежде чем тратить время и силы на заблаговременное выполнение того, что можно сделать позднее, исходя из положения элемента в журнале запросов на выполнение работ.

ВОЗРАЖЕНИЕ

“Мы не сможем найти время для ухода за журналом запросов на выполнение работ. Мы едва успеваем решать задачи, связанные с кодированием”.

Не забывайте, что Scrum требует планирования изменений. Обязательно выделите время для ухода за журналом запросов на выполнение работ. Возможно, это не нужно делать в ходе каждого спринта, но это нужно делать достаточно часто, чтобы небольшие элементы, рассчитанные на один спринт, всегда находились в верхней части журнала запросов на выполнение работ, и чтобы не отвлекаться на элементы, уяснение которых можно отложить на будущее.

Зачем нужно постепенно уточнять требования

Может оказаться удобным начинать новый проект с идентификации “всех” требований. Но поскольку в каждом проекте какие-то требования могут “всплывать” по ходу выполнения проекта, заблаговременно идентифицировать все требования не представляется возможным. К счастью, у постепенного уточнения требований есть свои преимущества.

- **Многое может измениться.** В ходе выполнения проекта приоритеты могут меняться. Некоторые функциональные возможности, которые поначалу казались важными, утрачивают свою важность после того, как соответствующая система продемонстрирована потенциальным пользователям и клиентам. Могут быть выявлены какие-то другие потребности, у каждой из которых есть свои приоритеты. Если мы исходим из неизбежности таких изменений, то преимущества структуризации журнала запросов на выполнение работ в виде айсберга становятся более очевидными. Функциональные возможности, вероятность изменения которых является самой высокой, — это такие функциональные возможности, которые предстоит реализовать в более отдаленном будущем. Чтобы учесть повышенную вероятность изменений, такие функциональные возможности описываются лишь на верхнем уровне.
- **Нет необходимости видеть очень далеко.** Писатель Э.Л. Доктороу (E.L. Doctorow) однажды сказал, что “написание романа подобно ночной поездке на автомобиле в условиях густого тумана. Вы можете видеть перед собой не дальше, чем достает свет фар вашего автомобиля, причем вся ваша поездка может проходить именно в таких условиях”. Это очень похоже на разработку программного обеспечения. Свет моих фар не достигает самого горизонта, но в этом, вообще говоря, нет необходимости. Света моих фар достаточно ровно настолько, чтобы я успевал вовремя реагировать на препятствия, которые могут возникнуть на

моем пути. Журнал запросов на выполнение работ в форме айсберга действует аналогичным образом. Он обеспечивает видимость элементов, достаточную для того, чтобы команда могла заглядывать достаточно далеко в будущее и избегать, таким образом, большинства потенциальных проблем. Чем быстрее движется команда, тем дальше ей приходится заглядывать в журнале запросов на выполнение работ.

- **Дефицит времени.** Почти все проекты выполняются в условиях жестко ограниченного времени. Мы всегда пытаемся сделать больше того, что позволяет выделенное нам время. Рассматривать все требования как равноценные совершенно неразумно. Учитывая ограниченность одного из самых важных ресурсов проекта (времени), относиться к нему нужно очень бережно. Если в данный момент нам достаточно описать какую-либо будущую функциональную возможность на верхнем уровне, то именно этим и следует ограничиться. Если эту будущую функциональную возможность требуется уяснить получше (то ли потому, что она переместилась в верхнюю часть журнала запросов на выполнение работ, то ли потому, что нам кажется, что она повлияет на реализацию какой-либо другой функциональной возможности), имеет смысл описать ее подробнее.

Постепенное уточнение историй пользователя

Гибкий процесс формулирования требований должен поддерживать их создание на разных уровнях, представленных в виде айсберга журнала запросов на выполнение работ (см. рис. 13.1). Члены команды должны уметь быстро создавать крупные требования-заполнители, которые располагаются у основания айсберга журнала запросов на выполнение работ, впоследствии расчленять их на элементы среднего размера, а в дальнейшем разделять и их на элементы достаточно малого размера, каждый из которых может быть реализован командой в рамках отдельно взятого спринта. Истории пользователя не только позволяют смещать акцент с письменного изложения требований на их обсуждение, но и удачно вписываются в айсберг журнала запросов на выполнение работ. Это обусловлено легкостью, с которой мы можем совершать переходы между крупными и мелкими историями пользователя.

Крупные истории пользователя обычно называются эпическими историями. Хотя я не могу указать какой-то конкретный размер, начиная с которого историю пользователя можно называть эпической, так, как правило, называют историю пользователя, реализация и тестирование которой занимают более одного или двух спринтов. Поскольку команда должна иметь возможность полностью завершить историю пользователя в рамках спринта, в котором она начинает ее реализацию, эпические истории должны быть расчленены на меньшие по размеру истории пользователя до того, как с ними начнется работа. Рассмотрим пример эпической истории и выясним, как можно расчленить ее на меньшие по размеру истории пользователя.

- Будучи пользователем, я должен войти в систему таким образом, чтобы только я имел доступ к своей информации.

Эта история может показаться неэпической; да она и не может быть таковой во всех ситуациях. Однако, исходя из наших целей, допустим, что владелец продукта вносит уточнение: эта простая история должна охватывать все, что касается входа в систему: запрос нового пароля, изменение пароля и т.д. Иначе говоря, речь идет не о

простом щелчке на кнопке Login на одном экране. Исходя из этого, команда приходит к выводу, что разработка и тестирование этой истории займет примерно два-три спринта. Это и делает данную историю эпической. Поскольку это эпическая история, она расчленяется на истории меньшего размера, каждая из которых, по предположению команды, может быть реализована в течение одного спринта. Вот возможная совокупность историй пользователя, меньших по своему размеру.

- Будучи зарегистрированным пользователем, я могу войти в систему, задав свои имя пользователя и пароль.
- Будучи новым пользователем, я хочу зарегистрироваться, создав имя пользователя и пароль, чтобы система могла запомнить мою персональную информацию.
- Будучи зарегистрированным пользователем, я могу изменить свой пароль, чтобы быть уверенным в его надежности или чтобы мне было легче его запомнить.
- Будучи зарегистрированным пользователем, я хочу, чтобы система предупреждала меня в случае, если я задал пароль, который легко взломать (т.е. система должна частично взять на себя заботу о том, чтобы посторонним лицам было как можно труднее проникнуть в мою учетную запись).
- Будучи забывчивым пользователем, я хочу, чтобы мне была предоставлена возможность запросить новый пароль, чтобы в случае, если я его забуду, мне не приходилось долго его вспоминать.
- Будучи зарегистрированным пользователем, я не хочу знать наверняка, что задал неправильное имя пользователя, пароль или то и другое в случае неудачной попытки входления в систему. Это нужно для того, чтобы тому, кто пытается выдать себя за меня, подобные попытки стоили большего времени.
- Будучи зарегистрированным пользователем, я хочу получать уведомление каждый раз, когда были совершены три последовательные неудачные попытки обратиться к моей учетной записи. Это нужно для того, чтобы я знал о любых несанкционированных попытках обратиться к моей учетной записи.

После того как эпическая история будет расчленена на более мелкие истории, я рекомендую вообще от нее избавиться. Удалите ее из используемого вами инструмента или уничтожьте соответствующую карточку. Сохранить эпическую историю можно для отслеживания (если в этом есть необходимость) или если она может обеспечить контекст для меньших историй, созданных на ее основе. Во многих случаях контекст пользовательских историй меньшего размера очевиден, поскольку соответствующие эпические истории должны расчленяться по принципу “в нужный момент”, как указывалось выше. Когда эпическая история удаляется и преобразуется в пользовательские истории меньшего размера непосредственно перед тем как команда приступит к работе над ними, помнить контекст этих небольших историй гораздо легче.

Иногда эпические истории приходится расчленять на эпические же истории

В случае, когда приходится иметь дело с гораздо большей эпической историей, чем в приведенном нами примере с паролем, расчленение может выполняться в несколько приемов: вначале большая исходная эпическая история расчленяется на ряд историй среднего размера (которые сами по себе являются эпическими историями);

те, в свою очередь, расчленяются на пользовательские истории меньшего размера. В качестве примера крупной эпической истории рассмотрим пользовательскую историю, которая относится к компании, разрабатывающей программное обеспечение для крупных магазинов розничной торговли.

- Будучи вице-президентом по маркетингу, я хотел бы выполнить обзор эффективности рекламных кампаний прошлого, что дало бы мне возможность выявить прибыльные кампании, которые стоило бы повторить еще раз.

Идея заключалась в том, что вице-президент по маркетингу хотел изучить статистические данные по различным рекламным кампаниям, проводившимся в прошлом, и, выбрав лучшие из них, попытаться воспроизвести их еще раз. Например, какая из рекламных кампаний привела к наилучшему результату: телевизионная реклама во время показа сериала *Отчаянные домохозяйки*, реклама, передаваемая по радио дважды в сутки, рекламные объявления в выпуске газеты за четверг или рекламная кампания по электронной почте?

Всем, кто участвовал в этом проекте, было ясно, что эта исходная история слишком велика, чтобы ее можно было реализовать в течение одного из их двухнедельных спринтов. Поэтому было решено расчленить эту историю на две.

- Будучи вице-президентом по маркетингу, я хотел бы выбрать отрезок времени для выполнения обзора эффективности рекламных кампаний прошлого, что дало бы мне возможность выявить прибыльные кампании.
- Будучи вице-президентом по маркетингу, я хотел бы выбрать тип рекламных кампаний (прямое обращение по почте, телевидение, электронная почта, радио и т.п.), которые я включу в свой обзор эффективности рекламных кампаний прошлого.

Команде показалось, что эти, меньшие по своему размеру, истории все же могут оказаться слишком велики, чтобы их можно было реализовать в течение одного спринта, а потому их следует подвергнуть дальнейшему расчленению. История о выборе используемого отрезка времени была расчленена на три истории.

- Будучи вице-президентом по маркетингу, я хочу выбрать простые диапазоны времени, которые будут использоваться для выполнения обзора эффективности рекламных кампаний прошлого, что дало бы мне возможность выбрать точную совокупность дат.
- Будучи вице-президентом по маркетингу, я хочу выбрать сезоны (весна, лето, зима, осень), которые будут использоваться для выполнения обзора эффективности рекламных кампаний прошлого, что дало бы мне возможность выявить тенденции, действующие на протяжении нескольких лет.
- Будучи вице-президентом по маркетингу, я хочу выбрать праздничный период (Пасха, Рождество и т.п.), который будет использоваться для выполнения обзора эффективности рекламных кампаний прошлого, что дало бы мне возможность выявить тенденции, действующие на протяжении нескольких лет.

После этого окончательного расчленения команда пришла к выводу, что результатирующие истории достаточно малы для того, чтобы их можно было реализовать в течение одного спринта, а потому остановилась на достигнутом. Следует, однако, обратить

внимание на то, что даже эти истории могут оказаться недостаточно тривиальными для того, чтобы их можно было реализовать. Выбрать такие праздничные периоды, как “начиная со Страстной пятницы и заканчивая Воскресеньем Христовым” или “начиная с Дня благодарения и заканчивая Рождеством”, было бы довольно трудно, поскольку точные даты в этом случае в разные годы могут быть разными. Не исключено, что команда может счесть эти истории слишком большими.

Во многих случаях вполне возможно перейти от крупной эпической истории вблизи ватерлинии айсберга непосредственно к небольшим историям, которые можно реализовать в течение одного спринта. Решение пройти через промежуточный этап расчленения крупной эпической истории на несколько эпических историй, меньших по своему размеру, зависит только от вас и в значительной мере определяется контекстом конкретного проекта.

Добавление условий удовлетворенности

Постепенно истории оказываются настолько малыми, что дальнейшее их расчленение становится нецелесообразным. На этой стадии постепенное уточнение соответствующего требования все еще возможно за счет добавления в пользовательскую историю так называемых *условий удовлетворенности*. Условие удовлетворенности — это не более чем приемочный тест верхнего уровня, который будет выполнен после того, как соответствующая пользовательская история будет завершена. Рассмотрим в качестве примера следующую историю.

- Будучи вице-президентом по маркетингу, я хочу выбрать праздничный сезон, который будет использоваться для выполнения обзора эффективности рекламных кампаний прошлого, что позволит мне выявить прибыльные рекламные кампании.

Мы уже пришли к выводу, что это достаточно малая история для того, чтобы команда могла реализовать ее в течение одного спринта. Давайте продолжим — совместно с владельцем продукта — постепенно уточнять это требование, добавляя его условия удовлетворенности. Для этого мы выписываем на обороте учетной карточки (выражаясь метафорически, если вы используете какой-либо из программных инструментов управления журналом запросов на выполнение работ) следующие условия удовлетворенности.

- Убедиться, что это работает с основными (для розничной торговли) праздниками: Рождество, Пасха, День Президента, День матери, День отца, День труда, Новый год.
- Поддерживает праздничные периоды, которые могут переходить из одного года в другой (периоды, которые охватывают три календарных года, не рассматриваются).
- Праздничные сезоны могут простираться от одного праздника до следующего (например, от Дня благодарения до Рождства).
- Праздничные сезоны могут определяться как ряд дней, предшествующих данному празднику.

Постепенное уточнение путем добавления условий удовлетворенности помогает членам команды, информируя их об ожиданиях владельца продукта от данной

функциональной возможности. Это может быть список того, что будет включено и что не будет включено в данную возможность. Например, учитывая условия удовлетворенности для этой истории, должно быть ясно, что нам нет нужды поддерживать китайский Новый год. Несмотря на то что я стараюсь не упустить ни одной возможности вкусить пряных китайских блюд, китайский Новый год в Соединенных Штатах Америки вовсе не относится к числу тех праздников, когда ожидается большая распродажа в магазинах. Разумеется, владелец продукта мог бы сделать это еще более очевидным, объяснив членам команды, что поддерживать китайский Новый год не нужно. Но в этом, наверное, нет особой необходимости, поскольку обсуждения, которые поддерживают эту письменную часть данной истории пользователя, должны выявлять подробности наподобие этой.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Преобразуйте уже имеющийся у вас журнал запросов на выполнение работ в пользовательские истории. Напишите на учетных карточках каждый из элементов текущего журнала запросов на выполнение работ. Разложите учетные карточки на большом столе и сгруппируйте между собой схожие. Для высокоприоритетных групп учетных карточек напишите индивидуальные пользовательские истории. Учтите, что отношения “один к одному” между старыми элементами журнала запросов на выполнение работ и новыми пользовательскими историями, наверное, не будет. Для низкоприоритетных групп учетных карточек замените каждую группу одной эпической историей. Прикрепите скрепкой старые учетные карточки к новой эпической истории. Таким образом, они будут у вас под рукой, когда настанет время расчленять эту эпическую историю на пользовательские истории размером в один спринт.
- Во время совещания, посвященного планированию вашего следующего спринта, позаботьтесь о том, чтобы к моменту окончания совещания у каждой пользовательской истории, которую вы включаете в этот спринт, были четко обозначенные условия удовлетворенности. Во время последующей ретроспективы спринта обсудите, насколько вам помогла идентификация этих условий удовлетворенности.

Учитесь начинать без спецификации

Поскольку Scrum-команда переносит свои акценты с написания требований на их обсуждение, а затем, в ходе выполнения проекта, постепенно уточняет эти требования, она уже не может позволить себе такую роскошь, как начинать выполнение проекта с традиционного документа со спецификациями. Этот факт может привести многие группы (а среди них, в первую очередь, группы обеспечения качества и подготовки технической документации) в полное замешательство. Внедрение Scrum и обеспечение долговременного успеха в связи с использованием этой методологии зависят, в частности, и от того, насколько комфортно вы будете чувствовать себя, приступая к выполнению проекта при отсутствии документа, содержащего “исчерпывающие” спецификации.

Должен сразу же пояснить: цель заключается вовсе не в том, чтобы отказаться от любой документации. Есть документы, которые могут оказаться весьма полезными. Просто документы со спецификациями нужно использовать надлежащим образом.

Помимо удовлетворения регуляторных требований или требований, предполагающих следование определенным правилам, основное применение документов со спецификациями заключается в том, чтобы передавать информацию, а делать это лучше всего в письменном виде. Хорошими примерами в этом отношении могут служить сложные или подробные вычисления, характерные, как правило, для научных и математических приложений, но можно привести и множество других примеров.

Одна из опасностей, связанных с документами со спецификациями, заключается в том, что их регулярное обновление проводится лишь в редких случаях. Прежде чем составлять тот или иной документ, спросите у себя, готовы ли вы регулярно его обновлять. Если не готовы, то хорошенько задумайтесь, прежде чем приступать к его составлению, или хотя бы укажите на документе срок его годности (что-то наподобие “употребить до такого-то числа” на упаковке с молоком).

Создание спецификаций на основе примеров

Изменить придется, наверное, и то, как именно вы составляете спецификации. Рассмотрите вариант специфицирования продукта на основе примеров. Примеры — замечательный способ сформулировать желаемое поведение системы, особенно если примеры дополняются обсуждениями и небольшим письменным текстом с соответствующими разъяснениями. Вот как описывает полезность использования примеров для объяснения поведения системы Гойко Аджич (Gojko Adzic), автор книги *Bridging the Communication Gap*.

Использование примеров из жизни помогает нам добиваться лучшего понимания наших мыслей другими людьми. Кроме того, иногда легче выявить несоответствия между реалистичными примерами. В обсуждении примеров должны участвовать разработчики, бизнесмены и тестеры. Разработчики получают дополнительную информацию о соответствующей области знаний, формируя таким образом прочный фундамент для последующей реализации проекта. Тестеры получают из первых рук необходимые им знания, что позволяет им влиять на разработку, предлагая важные ситуации для обсуждения (2009, 32).

Чтобы увидеть, как это работает, предположим, что мы создаем систему, которая будет использоваться в нашей собственной компании и которая будет автоматически утверждать или отвергать запросы на предоставление отгулов. Первое, что хотел бы получить от данной системы наш владелец продукта, — это автоматическое утверждение ею запросов, в которых указано меньшее количество дней, чем уже накоплено к этому времени работником, сделавшим запрос. Владелец продукта составляет следующую пользовательскую историю²: “*Будучи работником компании, я хочу, чтобы запрос на предоставление отгулов, число которых не превышает количества накопленных у меня отгулов, утверждался автоматически, т.е. чтобы мне не приходилось ожидать, пока это утверждение не поступит от моего начальства*”. Затем владелец продукта, возможно, совместно с тестером, уточняет это требование, подкрепляя его рядом примеров, приведенных в табл. 13.1.

² В связи с отличием американской системы отпусков от нашей мы говорим здесь об отгулах, а не о накапливаемом отпуске, что никак не влияет на суть излагаемого материала. — Примеч. ред.

Таблица 13.1. Примеры, показывающие, что запрос на большее число отголов, чем накоплено у работника, не будет утвержден автоматически

Количество накопленных отголов	Количество запрошенных отголов	Утвержден?
6	5	Да
5	6	Нет
5	5	Да

В строках табл. 13.1 демонстрируются разные тестовые ситуации. В первых двух столбцах показаны тестовые данные этих тестовых ситуаций. В последнем столбце показано, каким должен быть результат выполнения теста. Так, в первой строке описан работник, у которого накопилось шесть отголов и который попросил предоставить ему пять отголов. В последнем столбце мы видим, что этот запрос должен быть утвержден.

Нужно сказать, что это довольно простая иллюстрация спецификации на основе примеров. Давайте посмотрим, что происходит, когда владелец продукта составляет следующую пользовательскую историю: *“Будучи сотрудником, который работает в этой компании уже более года, я хочу, чтобы мой запрос на предоставление отголов утверждался автоматически, если количество запрошенных мною отголов превышает количество накопившихся у меня на текущий момент отголов не более чем на пять дней”*. Специфицируя это с помощью примеров, владелец продукта составляет табл. 13.2.

Таблица 13.2. Несколько более сложный пример демонстрирует эффективность составления спецификации на основе примеров

Количество накопленных отголов	Количество запрошенных отголов	Стаж работы в компании — более одного года	Утвержден?
10	11	Нет	Нет
10	11	Да	Да
10	11	Нет	Нет
10	15	Да	Да
10	16	Да	Нет

Табл. 13.2 все еще достаточно проста, но уже начинает демонстрировать эффективность составления спецификации на основе примеров.³ Я не стану приводить более подробных примеров, но укажу на то, как спецификация на основе примеров становится еще более полезной по мере усложнения сценариев. Например, в описанных выше пользовательских историях количество отголов, накопленных работником, было фиксированной величиной. Во многих компаниях отгулы накапливаются помесячно. Поэтому запрос, который мог быть отвергнут сегодня, мог бы оказаться утвержденным, если бы такой запрос на предоставление отголов был выдан спустя три

³ Я намеренно упростил этот пример, чтобы продемонстрировать, как работает спецификация на основе примеров. Более основательная реализация того же примера, но демонстрирующая более эффективные способы построения эквивалентных таблиц, предложена Джейфом Лангром (Jeff Langr), автором книги *Agile Java*. С реализацией, предложенной Лангром, можно ознакомиться по адресу www.informit.com/articles/article.aspx?p=1393274.

месяца. Чтобы указать подобную ситуацию с помощью примеров, нам нужно было бы добавить в таблицу столбцы с датой выдачи запроса, датой, начиная с которой работник хотел бы взять отгулы, и скоростью накопления. Разъяснение детального требования, подобного этому, путем сочетания обсуждений и примеров, повышает вероятность того, что представления владельца продукта о требованиях, предъявляемых к этому продукту, полностью совпадут с тем, что намерены создать разработчики.

Составление спецификации на основе примеров становится особенно эффективным средством, когда эти примеры удается трансформировать в автоматизированные тесты. Это не так уж неосуществимо, как может показаться на первый взгляд. Одно из наибольших преимуществ заключается в том, что мы можем сразу же сказать, не устарела ли та или иная спецификация. Для этого нужно выполнить автоматизированные тесты: если они пройдут успешно, значит, данное приложение соответствует своим спецификациям. Такие тесты становятся самопроверяющимися спецификациями: с одной стороны, они выражают детальные проектные решения, а с другой — автоматически проверяют, соответствует ли данное приложение своим спецификациям.

См. также Инструмент FitNesse, который можно загрузить с сайта www.fitnesse.org, является моим любимым инструментом, который позволяет создавать и выполнять тесты, специфицирующие функциональные возможности на основе примеров почти так же, как показано в тексте книги. Еще одним популярным инструментом является Cucumber.

Это практически тот же подход, который применяет норвежец Тронд Вингард (Trond Wingård), руководитель проектов, выполняемых с помощью гибкой методологии разработки. Команда Тронда Вингарда широко использует FitNesse, вики-доску для создания выполняемых тестов и спецификаций в форме тестов. Используемый им подход к реализации проектов представлен на рис. 13.2 и описан Вингардом так.

Мы применяем политику, согласно которой все без исключения требования и тесты должны быть представлены в FitNesse. Даже если мы приходим к выводу, что нам требуются какие-то тесты, выполняемые вручную, они также должны быть описаны на вики-доске FitNesse, а не каким-либо другим способом. Титульный лист содержит список из девяти “пользовательских эпических историй”, снабженных ссылками на страницы, содержащие соответствующие эпические истории. На каждой из этих страниц эпическая история описывается в формате пользовательских историй и сопровождается списком пользовательских историй, составляющих данную эпическую историю. Каждая из них соответствует определенному элементу в журнале запросов на выполнение работ и имеет ссылку на страницу с данной историей. На странице истории описание соответствующей истории сопровождается сгруппированным перечнем условий удовлетворенности (Conditions of Satisfaction — COS). Эти условия сгруппированы определенным образом, причем у каждой группы есть собственная страница с FitNesse-тестами для них. Создание и уяснение членами команды такой структуры оказалось очень легким делом. Такая структура оказала команде реальную помощь.

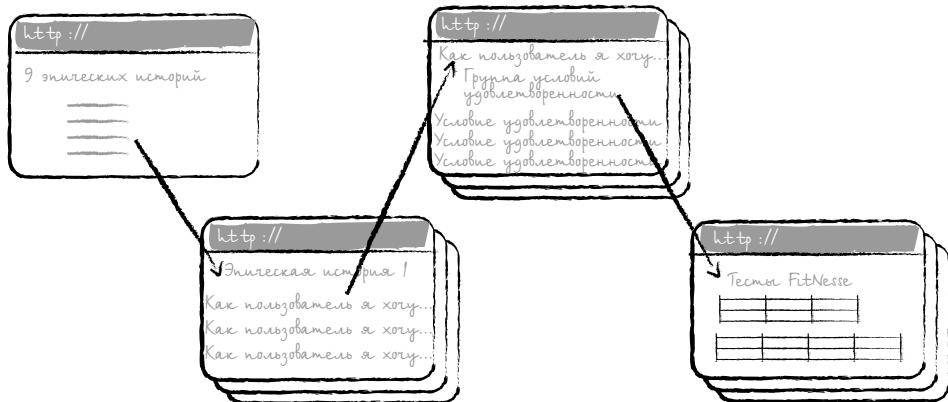


Рис. 13.2. Серия вики-страниц FitNesse, которая начинается с требований верхнего уровня, записанных как эпические истории, и продолжается вплоть до контрольных примеров

Межфункциональные команды снижают потребность в документации

Типичное возражение против полного отказа от документов со спецификациями или хотя бы сокращения их объема сводится к тому, что эти документы являются для некоторых групп единственным способом уяснить, что же должна делать система, которую они намерены разрабатывать. Например, группа контроля качества может утверждать, что без документов со спецификациями она не будет знать, как именно должна вести себя проектируемая система и какое поведение этой системы следует считать неправильным. В организации, которая еще не перешла к использованию Scrum, такая постановка вопроса вполне правомерна. Например, программисты могут собраться по собственной инициативе, принять какие-то решения, а затем довести эти решения до сведения тестеров посредством документов со спецификациями. После многих лет такой работы тестеры могут с полным правом заявлять, что, не имея на руках спецификаций, они не знают, что именно им следует тестировать.

Однако при выполнении Scrum-проекта программисты и тестеры работают, как единая команда. Нет команды программистов, которая передает выполненную работу команде тестеров. Выполнением Scrum-проекта занимается межфункциональная, многодисциплинарная команда. В этом случае тестеры уже не нуждаются в документах со спецификациями, поскольку работа не передается им так, как раньше. По сути, работа не передается им вообще! Тестер должен быть участником обсуждения каждый раз, когда его предметом является то, что в прошлом могло бы стать частью документа со спецификациями.

В прежние времена мне часто приходилось быть участником дискуссий, которые велись между программистами и тестерами, работавшими над тем или иным проектом. Тестеры жаловались, что программисты слишком безответственно относятся к вопросу регулярного обновления документации; программисты, со своей стороны, жаловались, что эта документация им вовсе ни к чему. Неоднократно наблюдая такие перепалки, я пришел к заключению, что составлять документацию должны именно

те, кому она приносит какую-то пользу. Поскольку именно тестеры заявляли, что им нужны документы с подробными спецификациями (регулярным обновлением которых не желали заниматься программисты), именно им нужно было поручить составление и постоянное обновление таких документов. Это позволяло не только решить указанную проблему, но и обеспечивало дополнительное преимущество, которое заключалось в том, что программистам и тестерам приходилось общаться между собой уже на самых ранних стадиях выполнения проекта (и гораздо чаще, чем прежде), что давало возможность тестерам получить информацию, необходимую для составления документов с подробными спецификациями. Йоханнес Бродуолл (Johannes Brodwall) сообщает об использовании аналогичной стратегии.

Тестеры привыкли получать очень неконкретную и расплывчатую (и вместе с тем очень подробную) документацию, которую им приходилось преобразовывать в контрольные примеры. При использовании более гибкого подхода именно тестер становится тем, кто фактически отвечает за составление документов с подробными спецификациями. Сейчас в нашей компании в начале каждой итерации именно тестер составляет “спецификации” в виде сценариев, подлежащих тестированию.

Критерии DEEP для журнала запросов

Роман Пихлер (Roman Pichler), автор книги *Agile Product Management with Scrum: Creating Products That Customers Love*, и я используем аббревиатуру “DEEP” для обозначения важнейших характеристик хорошего журнала запросов на выполнение работ.

- **Detailed Appropriately** (Оптимальный уровень детализации). Пользовательские истории, представленные в журнале запросов на выполнение работ и подлежащие реализации в ближайшее время, должны быть настолько понятными, чтобы их можно было реализовать в течение предстоящего спринта. Пользовательские истории, которые предстоит реализовать в более отдаленном будущем, должны быть описаны менее подробно.
- **Estimated** (Точность оценок). Журнал запросов на выполнение работ — это не просто перечень всех работ, подлежащих выполнению. Это еще и полезный инструмент планирования. Поскольку элементы, расположенные в нижней части журнала запросов на выполнение работ, еще не совсем понятны, связанные с ними оценки неизбежно окажутся менее точными, чем оценки, данные элементам в верхней части журнала запросов на выполнение работ.
- **Emergent** (Изменчивость). Журнал запросов на выполнение работ не является чем-то статичным. С течением времени он изменяется. По мере получения дополнительной информации в нем появляются новые пользовательские истории, какие-то из пользовательских историй удаляются, изменяются приоритеты пользовательских историй.
- **Prioritized** (Приоритизация журнала). Журнал запросов на выполнение работ должен быть отсортирован таким образом, чтобы самые важные элементы находились наверху, а наименее важные — внизу. Действуя в строгом соответствии с этими приоритетами, команда может максимизировать ценность разрабатываемого продукта или системы.

Не забывайте обсуждать

Несмотря на то что журнал запросов на выполнение работ того или иного проекта поддерживается на каком-то носителе (как правило, на учетных карточках) или вводится в какой-либо программный инструмент, такой журнал не является стопроцентной заменой традиционного документа с требованиями по данному проекту. Не менее важным, чем записанное в фактическом журнале запросов на выполнение работ, являются связанные с ним обсуждения. Эти обсуждения происходят, когда команда совместно с владельцем продукта, пользуясь методом мозгового штурма, определяет элементы, которые будут включены в первоначальный вариант журнала запросов на выполнение работ. Эти обсуждения происходят и во время спринта, когда команда совместно с владельцем продукта постепенно уточняет свое понимание той или иной функциональной возможности. Стремясь повысить эффективность использования вашей командой журнала запросов на выполнение работ, не забывайте о важности таких обсуждений!

Дополнительная литература

Adzic, Gojko. 2009. *Bridging the communication gap: Specification by example and agile acceptance testing*. Neuri Limited.

В этой превосходной книге показано, почему формулирование требований является столь трудной задачей. В качестве возможного решения этой проблемы предлагается использование спецификаций на основе примеров. Особенno ценной представляется глава, посвященная выбору подходящих примеров. Кроме того, в книге есть глава, посвященная инструментам, которые облегчают составление спецификаций на основе примеров.

Cao, Lan, and Balasubramaniam Ramesh. 2008. Agile requirements engineering practices: An empirical study. *IEEE Software*, January/February, 60–67.

Авторы этой научно-исследовательской статьи изучили процесс сбора требований в 16 организациях, занимающихся разработкой программного обеспечения и использующих гибкую методологию разработки. На основе своего исследования они определили выгоды и проблемы, связанные с использованием ряда конкретных методов формулирования требований в условиях использования гибкой методологии разработки.

Cohn, Mike. 2004. *User stories applied: For agile software development*. Addison-Wesley Professional.

В этой книге представлено подробное пояснение работы с пользовательскими историями. В ней идет речь об идентификации ролей пользователей, составлении пользовательских историй, проведении семинаров по составлению пользовательских историй, шести характеристиках хороших пользовательских историй и даже о том, как планировать проект с помощью пользовательских историй.

Mugridge, Rick, and Ward Cunningham. 2005. *Fit for developing software: Framework for integrated tests*. Prentice Hall.

Fit, или *Framework for Integrated Tests*, является продуктом с открытым кодом. Его можно использовать для создания удобочитаемых автоматизированных тестов, которые могут определять поведение на основе примеров, подобно таблицам, представленным в последнем разделе этой главы. Первые 180 страниц книги предназначены для всех участников проекта (как технических специалистов, так и всех остальных) и демонстрируют преимущества использования *Fit* при выполнении проектов. Следующие 150 (или что-то около того) страниц предназначены для тех, кто имеет опыт программирования, и посвящены специфике использования *Fit*.

Глава 14

Сprintы

Подобно всем процессам гибкой методологии разработки Scrum является итеративным и инкрементным подходом к разработке программного обеспечения. Каждый из терминов *итеративный* и *инкрементный* имеет собственное значение, но очень часто они употребляются вместе. Давайте попробуем их разделить, чтобы уяснить смысл каждого из них по отдельности.

Инкрементная разработка предполагает пошаговое построение системы. Сначала разрабатывается одна часть, затем к первой добавляется следующая и т.д. Алистер Кокбурн (Alistair Cockburn) описывает инкрементный процесс разработки главным образом как “стратегию разбиения на отдельные стадии и составления календарного плана реализации этих последовательных стадий” (2008). Инкрементный подход к разработке сайта для проведения онлайн-аукционов может предусматривать (1) разработку способности создания учетных записей на таком сайте, (2) разработку способности составлять перечень предметов, выставляемых на продажу, (3) разработку способности проводить торги по отдельным предметам, выставленным на продажу, и т.д.

В отличие от инкрементной разработки итеративная разработка — это то, что Алистер Кокбурн называет “стратегией переработки календарного плана” (2008). Итеративный процесс разработки исходит из нереальности (или по крайней мере малой вероятности) создания той или иной функциональной возможности с первого раза. При итеративном построении сайта для проведения онлайн-аукционов мы можем сначала разработать некую предварительную версию всего сайта, получить от пользователей сведения о функционировании этой версии, разработать следующую версию всего сайта (с учетом замечаний, полученных от пользователей в ходе работы с предварительной версией) и повторять указанный цикл до тех пор, пока не будет получен приемлемый результат.

Таким образом, в инкрементном процессе мы сначала полностью разрабатываем одну функциональную возможность, а затем переходим к разработке следующей функциональной возможности. В итеративном процессе разработки мы создаем всю

систему, которая с первого раза получается очень несовершенной. Затем, получив информацию о функционировании этой первоначальной версии, мы повторяем процесс разработки всей системы столько раз, сколько потребуется для получения системы, которая устраивает пользователей. Недостатки, присущие по отдельности итеративному и инкрементному подходам к разработке, устраняются при совместном использовании этих подходов, как это имеет место в Scrum.

В данной главе речь идет о том, как в Scrum-спринтах сочетаются итеративный и инкрементный подходы к разработке. Мы рассмотрим важность завершения каждого спринта созданием работоспособного программного обеспечения, которое представляет практическую ценность для пользователей или заказчиков соответствующей системы. Вы узнаете, почему всей команде необходимо работать вместе на протяжении всего спринта. По ходу дела мы рассмотрим также, как Scrum-команда добивается того, чтобы, завершая один спринт, она уже была готова к началу следующего спринта; рассмотрим важность постановки цели для каждого спринта и неукоснительного следования этой цели. Кроме того, мы укажем на необходимость задания четких временных рамок, в которых команде предстоит достичь поставленной цели.

Каждый спринт должен заканчиваться созданием работоспособного программного обеспечения

К концу каждого спринта Scrum-команда обязана создать работоспособное программное обеспечение. Работоспособное программное обеспечение — это готовое программное обеспечение, которое можно передать заказчику. Его должны создавать как команды, разрабатывающие определенные функциональные возможности, так и компонентные команды. Научиться создавать работоспособное программное обеспечение к моменту окончания каждого спринта — вот одна из самых сложных задач, которые предстоит решить начинающей Scrum-команде. Тем не менее, чтобы овладеть гибкой методологией разработки, эту задачу обязательно нужно решить. По сути, это настолько важно, что одна из четырех ценностей, указанных в Agile Manifesto, заключается в том, что мы должны отдавать предпочтение “работоспособному программному обеспечению по сравнению с исчерпывающей документацией” (Beck et al., 2001). Гибкие методологии разработки предпочитают работоспособное программное обеспечение в силу трех важнейших причин.

См. также Команды, разрабатывающие определенные функциональные возможности, и компонентные команды обсуждались в главе 10, “Структура команды”.

- **Работоспособное программное обеспечение стимулирует обратную связь.** Команда может собирать информацию обратной связи в большем объеме и более высокого качества, если она демонстрирует (а еще лучше — передает) пользователям неполный, но функционирующий продукт, чем в случае, когда она передает этим пользователям документ о том, как будет функционировать данный продукт. Пользователи, которые могут увидеть и “пощупать” данный продукт, не только

будут снабжать изготавителей этого продукта более качественной информацией обратной связи, но и смогут принимать более активное участие в формировании запросов на предоставление такой информации. В то же время 50-страничная спецификация на продукт, скорее всего, останется невостребованной.

- **Работоспособное программное обеспечение помогает команде оценивать объем работ.** Один из наибольших рисков при выполнении любого проекта состоит в том, что вы далеко не всегда знаете, какой объем работы вам остается выполнить. Когда очень большая часть системы остается в незавершенном и неработоспособном состоянии, трудно понять, какой объем работы потребуется для приведения системы к более или менее “товарному” виду. Делая акцент на создании работоспособного программного обеспечения и требуя поставки какой-то части “пользовательской стоимости” к моменту завершения каждого спринта, Scrum-командам удается избежать этой проблемы.
- **Работоспособное программное обеспечение позволяет передавать продукт заказчику раньше запланированного срока, если в этом возникает потребность.** В нынешнем высококонкурентном и быстро меняющемся мире вариант досрочной поставки продукта заказчику (даже если этот продукт заключает в себе меньше функциональных возможностей, чем запланировано) может представлять собой немалую ценность. Приведение разрабатываемого программного обеспечения в состояние готовности к поставке (или близкое к нему) к моменту окончания каждого спринта позволяет нам реализовать эту ценность.

См. также Подробнее о корректировках масштаба рассказывается в главе 15, “Планирование”.

Определение понятия “потенциально готов”

В середине 1990-х годов особой популярностью у команд пользовалась итеративная и инкрементная разработка программного обеспечения, имеющая своей целью периодическое приведение приложения к контрольной точке с нулевым уровнем дефектов (Zero Defect — ZD). Джим Маккарти (Jim McCarthy), бывший директор группы Microsoft Visual C++, часто писал и рассуждал об этих контрольных точках.

Нулевой уровень дефектов не означает, что в соответствующем продукте отсутствуют дефекты и что в нем реализованы все запланированные функциональные возможности. Нулевой уровень дефектов означает, что данный продукт достигает такого уровня качества, который был установлен для данной контрольной точки. Исходя из этого осуществляется тестирование продукта. Важный момент таких контрольных точек заключается в том, что никакой отдельно взятый исполнитель не достигает этой контрольной точки, пока ее не достигнут все исполнители, и никакой отдельно взятый исполнитель не покидает этой контрольной точки, пока ее не покинут все исполнители. Это дает возможность команде определять, в каких аспектах проекта возникают проблемы.

Достигнув той или иной контрольной точки, команда и ее руководство получают возможность уяснить состояние проекта в целом, сделать выводы о

неэффективных или ошибочных методах, устраниТЬ последствия неправильных проектных решений и провести реорганизацию с целью достижения максимальной производительности... По достижении каждой контрольной точки у команды появляется возможность полностью сосредоточиться на полученных результатах и уяснить их (2004).

Несмотря на то что ZD-контрольные точки Джима Маккарти являются вполне подходящей целью для многих команд, Scrum-команды устремляют свои взгляды выше и нацеливаются на создание программного обеспечения, потенциально готового к отправке заказчику, к окончанию каждого спринта. Но что же мы подразумеваем под продуктом, *потенциально готовым к отправке заказчику*? Чтобы дать исчерпывающее определение продукта, *потенциально готового к отправке заказчику*, нужно знать предметную область и приложение, о которых идет речь в том или ином конкретном случае. Таким знанием обладает лишь сама команда, включая владельца продукта и Scrum-мастера. По сути, единственное, что должна сделать любая команда-новичок, — это прийти к согласию относительно понятия *выполнено*, которое определяет приращение (инкремент) продукта, потенциально готового к отправке заказчику, применительно к конкретной ситуации. При этом каждый элемент журнала запросов на выполнение работ, который планируется выполнить в течение определенного спринта, может считаться готовым, если он отвечает этим критериям. В главе 13, “Журнал запросов на выполнение работ”, вы ознакомились с понятием условий удовлетворенности как критерия приемки для индивидуальных пользовательских историй. Во многих отношениях элементы, которые входят в определение понятия *выполнено* для той или иной команды, подобны условиям удовлетворенности, которые применяются ко всем пользовательским историям в журнале запросов на выполнение работ.

В качестве примера можно привести компанию ePlan Services, которая определяет понятие *выполнено* как “закодировано, протестировано, зарегистрировано, качественно задокументировано, интегрировано и имеет автоматизированные тесты”. Каждый элемент журнала запросов на выполнение работ, над которым работает команда, должен отвечать этим ожиданиям, помимо условий удовлетворенности, специфических для данного элемента. Рассмотрим следующую пользовательскую историю *ePlan Services*: “*Будучи пользователем, я могу оплатить стоимость обслуживания своего счета с помощью кредитной карточки таким образом, чтобы соответствующая сумма не снималась с моего пенсионного счета*”. Исходя из этой пользовательской истории, допустим, что владелец продукта задал следующие условия удовлетворенности.

- Принимать кредитные карточки Visa, MasterCard и American Express
- Не хранить информацию кредитных карточек в нашей системе
- Обрабатывать все транзакции в режиме обеспечения максимальной безопасности

Таким образом, нужно, чтобы выполнялись не только эти условия удовлетворенности, но и условия, заданные в определении понятия *выполнено* применительно к данному проекту (закодировано, протестировано, зарегистрировано, качественно задокументировано, интегрировано и имеет автоматизированные тесты).

ВОЗРАЖЕНИЕ

“Разрабатываемое нами приложение является слишком сложным, чтобы его можно было разрабатывать инкрементным способом”.

Обычно этот аргумент означает не то, что инкрементный способ создания продукта невозможно реализовать на практике, а то, что людям просто трудно представить себе конкретные способы практической реализации этого способа. Когда мне заявляют: “Разрабатываемое нами приложение является слишком сложным, чтобы его можно было разрабатывать инкрементным способом”, я прошу своего собеседника указать мне несколько естественных “контрольных точек” в приложении, о котором он ведет речь. Обычно он указывает мне три-четыре такие точки, делящие систему на три-четыре сравнительно автономные части, и совершенно справедливо настаивает на том, что эти части чересчур велики, чтобы их можно было разработать в течение одного спринта. Однако после того как он подтвердил сам факт возможности инкрементной разработки системы (даже если такие приращения слишком велики, по крайней мере на данной стадии обсуждения), я уже не принимаю доводов типа “разрабатываемое нами приложение является слишком сложным, чтобы его можно было разрабатывать инкрементным способом”. На данной стадии мы оба согласны с тем, что инкрементный способ разработки системы возможен, и нам остается лишь изыскать способы разбиения системы на ряд логических функциональных возможностей, чтобы каждую из них можно было разработать в течение одного спринта.

Затем я указываю своему собеседнику: несмотря на то что мы действительно хотим создавать к окончанию каждого спринта фрагменты продукта, потенциально готовые к отправке заказчику, мы вовсе не обязаны завершать каждый спринт созданием продукта, представляющего собой единое целое. Иначе говоря, несмотря на то, что к окончанию каждого спринта разрабатываемый нами продукт должен представлять собой вполне работоспособную систему, мы называем его *потенциально готовым* к отправке заказчику, напоминая тем самым о том, что разработанных на данный момент функциональных возможностей еще недостаточно для того, чтобы наш продукт был *по-настоящему* готов к отправке заказчику.

Рекомендации по определению понятия “потенциально готов”

Несмотря на то что никто, кроме самой организации или команды, не сможет дать наиболее подходящее определение понятия “выполнено” применительно к контексту этой организации или команды, мы можем все же привести ряд рекомендаций, которые подходят для большинства Scrum-проектов, выполняемых в большинстве организаций.

“Потенциально готов” означает “протестирован”. Несмотря на то что точное определение того, что именно означает “потенциально готов”, должна дать сама организация или команда, я не могу представить себе ситуацию, при которой команда исключила бы из этого определения тестирование. К окончанию спринта мы имеем полное право рассчитывать на то, что в новых функциональных возможностях окажутся выловленными все ошибки, а в ранее разработанные функциональные возможности не будут внесены новые ошибки. В случае некоторых продуктов мы не можем быть уверены в этом полностью, но по крайней мере мы хотим, чтобы такая уверенность была почти стопроцентной. Следует отметить, что особые типы тестирования, такие

как интегральное тестирование, проверка производительности, проверка удобства использования и другие, могут и не выполняться в каждом спринте. Вместо этого данные типы тестирования могут осуществляться в выпускных спринтах (release sprints), которые могут выполняться после нескольких “обычных” спринтов.

В качестве примера надлежащего использования выпускного спринта рассмотрим банк, в котором мне однажды пришлось работать. В нем использовалось крупное приложение, унаследованное еще с тех времен, когда в этом банке эксплуатировались мэйнфреймы. Это приложение состояло из нескольких миллионов строк на языке COBOL, в написании и сопровождении которых принимало участие не менее десятка разработчиков. При разработке этого приложения использовался процесс последовательной разработки, который эти разработчики применяли уже не менее двух десятилетий. Активные разработки в этой части системы практически не велись, и это было просто замечательно, поскольку тестирование такого монстра заняло в свое время три недели упорного ручного труда.

Этот банк использовал также относительно небольшое (порядка 300 тысяч строк) Java-приложение, которое обеспечивало веб-доступ к тем же финансовым данным. Оба приложения пользовались одной и той же базой данных, так что веб-приложение могло заносить в нее информацию, которая могла отрицательно сказаться на COBOL-приложении. Это веб-приложение в основном было создано Scrum-командой.

Будучи “правильной” Scrum-командой, наша веб-команда намеревалась разрабатывать продукт, “потенциально готовый к отправке заказчику”, к окончанию каждого спринта. Члены команды определили продукт, *потенциально готовый к отправке заказчику*, как программный продукт, качественно написанный и протестированный до такой степени, чтобы разработчики имели все основания рассчитывать на отсутствие в нем существенных (т.е. способных иметь финансовые последствия) ошибок. Это предполагало написание ими максимально качественного кода, а также добавление к данному веб-приложению и последующее выполнение полного набора автоматизированных тестов. По их мнению, это позволяло перевести данное веб-приложение в состояние “потенциально готового к отправке заказчику”. Переход от “потенциально готового” к просто “готовому” требовал проведения время от времени выпускных спринтов, в ходе которых проводилось трехнедельное ручное тестирование приложения, написанного на языке COBOL для мэйнфрейма.

“Потенциально готов” вовсе необязательно означает “готов к использованию”. Сам по себе факт, что продукт потенциально готов к отправке заказчику, еще не означает, что заказчик хочет, чтобы мы действительно отправили ему этот продукт. Иногда требуется два, три и даже больше спринтов, чтобы тот или иной набор функциональных возможностей принял вид, в котором его можно было бы применять с большей или меньшей пользой для дела. Однако в течение спринтов, выводящих продукт на данный уровень готовности, команда все равно должна стремиться перевести продукт в состояние “потенциально готового к отправке заказчику” к окончанию каждого спринта.

Рассмотрим в качестве примера компанию, которая добавляла к своему продукту функциональную возможность распечатки и предварительного просмотра на экране информации, подготовленной к печати. В ходе планирования спринта команда пришла к выводу, что не сможет добавить в течение одного спринта функциональную возможность распечатки и предварительного просмотра на экране информации, подготовленной к печати. С согласия владельца продукта команда решила реализовать сначала функцию предварительного просмотра. К окончанию спринта команда успешно справилась с

этой задачей — функция предварительного просмотра была полностью готова, она была качественно написана, тщательно протестирована и отправлена заказчику вместе с соответствующим продуктом. Однако кому нужен продукт, в котором реализована функция предварительного просмотра на экране информации, подготовленной к печати, но не реализована функция печати? Тем не менее отсутствие после завершения первого спринта полностью готовой к использованию функциональной возможности отнюдь не помешало соответствующему продукту быть потенциально готовым к отправке заказчику. Если бы кто-то хотел получить этот продукт лишь с функцией предварительного просмотра, то команда могла бы предоставить ему такой продукт.

“Потенциально готов” означает “интегрирован”. Продукт, потенциально готовый к отправке заказчику, не имеет вид 14 разных наборов исходного кода. В случае проекта, который выполняет несколько команд, эти команды должны определить понятие *выполнено* таким образом, чтобы оно включало интеграцию потоков разработки. Насколько возможно интеграция должна выполняться непрерывно на протяжении всего спринта.

См. также Подробнее об интеграции работы нескольких команд рассказывается в главе 17, “Изменение масштаба Scrum”.

ВОЗРАЖЕНИЕ

“Мы не можем выполнять спринты в начале своего проекта: сначала нам нужно выстроить определенную инфраструктуру”.

Разработчики, лишь приступающие к освоению Scrum, зачастую приходят к заключению, что итеративная и инкрементная разработка возможна лишь после того, как будет написана значительная часть приложения. Я совершенно не согласен с такой точкой зрения. Даже инфраструктурные элементы можно создавать инкрементным способом. Во время первых же спринтов зачастую необходимо работать в основном (или исключительно) над инфраструктурными аспектами разрабатываемого продукта. Я согласен с тем, что зачастую трудно находить способы продемонстрировать ценность этой работы конечным пользователям, однако подчас полезно проявить в этом отношении настойчивость и упорство, особенно на ранних стадиях. Если что-то кажется нам трудным, то это еще не повод для того, чтобы отказаться от этого. Напротив, нужно искать способы расчленения этих первых инфраструктурных элементов на более мелкие фрагменты, которые можно было бы реализовать в течение одного спринта. Я, например, поступаю следующим образом: я стараюсь представить себе естественные пункты, достигая которых, я могу позвать кого-то из своих сотрудников и сказать: “Посмотри-ка на эту классную штуковину, которую я только что изготовил”. Как только какая-нибудь работа достигает у вас такой степени готовности, что вы можете предложить кому-то из сотрудников оценить полученный вами результат (даже если эта оценка будет выражаться словами одобрения, вроде “а неплохо получилось”), это и можно считать такой частью функциональной возможности, на реализацию которой следует ориентироваться в одном из первых спринтов.

Нахождение таких естественных пунктов поможет вам добиться того, что даже самые первые спринты команды будут приводить к созданию чего-то ощутимого или представляющего известную ценность для пользователя или заказчика. Однако этот вопрос является темой следующего раздела.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Попросите владельца продукта обсудить вместе с командой и прийти к какому-то определенному выводу о том, что именно следует понимать под словом *выполнено* в конце каждого спринта. Сформулируйте соответствующее определение и поместите эту формулировку на самом видном месте.
- ❑ Вместе с командой составьте перечень всех проблем, с которыми вам приходилось сталкиваться при выполнении предыдущих проектов. Речь должна идти о проблемах, когда разрабатываемый продукт оказывался слишком далеко от состояния готовности к отправке заказчику. Обсудите, что можно сделать, чтобы избежать возникновения этих проблем в будущем.

Создавайте к окончанию каждого спринта что-то значимое

От Scrum-команд требуется не только, чтобы каждый спринт завершался созданием работоспособного программного обеспечения, но и чтобы к окончанию каждого спринта они создавали что-либо ценное для пользователей или заказчика соответствующей системы или продукта. Правда, не совсем понятно, что именно следует считать ценным для пользователей или заказчика: понятие ценности в этом случае может быть довольно растяжимым (в том числе и “намеренно растяжимым”). Например, команда может сказать, что модернизация всех компьютеров разработчиков и установка новейшей версии операционной системы позволит им ускорить процесс разработки, что даст возможность быстрее сдавать заказчикам новые функциональные возможности. Несмотря на то что это заявление вполне может соответствовать действительности, все же цель заключается в том, чтобы каждый спринт завершался созданием чего-либо такого, что представляет очевидную ценность для пользователей или заказчиков. Поскольку одно из преимуществ работы спринтами заключается в возможности получения отзывов пользователей или заказчиков в конце каждого спринта, добытая командой информация будет более качественной, если хотя бы часть работы, выполняемой во время каждого спринта, приведет к созданию функциональных возможностей, которые пользователи смогут “пощупать” собственными руками.

В качестве примера функциональных возможностей, которые пользователи смогут “пощупать” собственными руками я предлагаю рассмотреть следующую ситуацию. Допустим, что некая команда разрабатывает веб-сайт с информацией о жилых домах, выставленных на продажу. Во время одного из совещаний, посвященных планированию спринта, владелец продукта изъявил желание добавить возможность получения пользователем списка домов, удовлетворяющих определенным критериям поиска, представляющего собой то или иное сочетание 20 поисковых параметров. Команда говорит владельцу продукта, что в течение одного спринта такую функциональную возможность реализовать не удастся. По мнению команды, эту работу нужно разделить на составные части и выполнить ее в течение двух спринтов. В результате совместного обсуждения данной проблемы владелец продукта и команда пришли к следующим вариантам того, что они могли бы реализовать в ходе предстоящего спринта.

1. Команда сосредоточивается лишь на поиске, скрытом за пользовательским интерфейсом. Во время обзора спрингта пользователи увидят лишь текстовые результаты поисковых запросов, выполненных из командной строки.
2. Команда сосредоточивается исключительно на пользовательском интерфейсе. Во время обзора спрингта пользователи увидят полностью функциональные экраны, но лишь с имитацией данных, а не с данными, реально найденными в базе данных.
3. Команда одновременно занимается и поиском, скрытым за пользовательским интерфейсом, и собственно пользовательским интерфейсом. Во время обзора спрингта команда демонстрирует приложение, которое поддерживает 10 из 20 запланированных полей поиска, а также пользовательский интерфейс, который, хотя и является функциональным, остается все же неполным.

Какой из этих вариантов является оптимальным? Начнем с того, что иногда приемлемыми могут оказаться все варианты. Однако, вообще говоря, предпочтение следовало бы отдать третьему варианту. В этом случае команда посыпает (посредством своего приложения) то, что Энди Хант (Andy Hunt) и Дэйв Томас (Dave Thomas) называют “трассирующей пулей”. Согласно Дэйву Томасу трассирующая пуля — это попытка “произвести как можно раньше нечто такое, что даст возможность пользователю увидеть, насколько близко мы подошли к поставленной цели. В дальнейшем мы можем слегка подкорректировать нашу цель, увидев, насколько далеко мы отстоим от цели нашего пользователя” (Venners, 2003).

Но что можно сказать о первом и втором вариантах? Если бы я был наставником этой команды, то мог бы согласиться с любым из этих двух подходов, но лишь при невозможности послать “трассирующую пулю” посредством соответствующей функциональной возможности, как в случае с третьим вариантом. Из первых двух вариантов я, несомненно, предпочитаю первый (создание части, скрытой за пользовательским интерфейсом, и демонстрация ее работоспособности посредством интерфейса командной строки). В этом сценарии команда может продемонстрировать работоспособность функциональной возможности, которая интересует пользователя. Правда, эта функциональная возможность выглядит пока еще не так красиво, как хотелось бы пользователю, и пока еще не добавлена на веб-страницу. Если бы все это было продемонстрировано во время обзора спрингта, то большинство пользователей или заказчиков согласилось бы с тем, что реализация этой функциональной возможности явила определенным шагом вперед.

Второй вариант (разработка только пользовательского интерфейса) кажется мне малопривлекательным. Несмотря на то что можно было бы утверждать, будто реализация только части, скрытой за пользовательским интерфейсом, или только собственно пользовательского интерфейса являются, по сути, двумя сторонами одной и той же медали, мне кажется, разница между этими двумя вариантами все же есть. Когда команда демонстрирует только часть, скрытую за пользовательским интерфейсом, вряд ли кому-то покажется, что реализована вся функциональная возможность. Кто-то из заинтересованных лиц наверняка подумает: “Все, что нам остается, — это снабдить данное приложение соответствующим пользовательским интерфейсом”. Но в случае, когда команда демонстрирует только ту часть, которая представляет собой пользовательский интерфейс как таковой, кто-то из заинтересованных лиц может подумать, что реализована вся функциональная возможность, поскольку он уже видел ее в действии.

Итак, хотя первые два варианта действительно представляют определенную ценность для пользователей системы, команде следует отдать предпочтение отправке “трассирующей пули” посредством соответствующего приложения. Я упоминаю здесь об альтернативных вариантах (первом и втором), но считаю их допустимыми лишь в случае, когда команда находит совершенно невозможным послать “трассирующую пушку”.

Невидимые функциональные возможности

Хотя не все продукты включают функциональные возможности, видимые конечным пользователям, в любом случае каждый продукт содержит функциональность, кому-то видимую. Допустим, например, что над созданием веб-сайта, который позволит людям искать жилые дома, выставленные на продажу, работает не одна, а сразу пять команд. Четыре из них работают над реализацией определенных функциональных возможностей, видимых посетителям этого веб-сайта, а пятая представляет собой компонентную команду, создающую уровень доступа к общим данным, который будет использоваться первыми четырьмя командами.

См. также В главе 10, “Структура команды”, обсуждаются преимущества команд, работающих над реализацией определенных функциональных возможностей, и компонентных команд.

Эта пятая команда может полагать, будто ничего из того, что она создает, не будет видно пользователям. Неправильность такого вывода становится очевидной, когда эта команда поймет, что ее пользователями являются остальные четыре команды, а не так называемые “конечные пользователи”. Эта пятая команда, которую мы назовем командой, обеспечивающей доступ к данным, нуждается в получении информации обратной связи от своих пользователей — программистов и тестеров, работающих в остальных четырех командах. Именно им будет видна функциональность, созданная пятой командой. Это означает, что в обзоре своих спринтов команда, обеспечивающая доступ к данным, могла бы продемонстрировать чисто техническую функциональную возможность (например, каскадное удаление в базе данных), которую нет никакого смысла демонстрировать команде, работающей над реализацией определенных функциональных возможностей.

Йоханнес Бродуолл (Johannes Brodwall) работал над проектом, связанным с созданием новой версии системы для пакетной обработки платежей, отправляемых в электронном виде. Несмотря на ее огромную ценность для заказчиков, она является хорошим примером системы, малозаметной для ее пользователей. Бродуолл описывает, как члены этой команды решили указанную проблему.

Для обзора последнего нашего спринта мы обработали в новой системе данные за четыре недели функционирования производственной системы. Демоверсия представляла собой веб-страницу, которая собирала выходные данные новой системы и сравнивала их с выходными данными старой системы. На веб-странице было представлено в виде таблицы количество транзакций, которые имели различающиеся результаты. Эти транзакции были сгруппированы по величине отклонения в столбцах (например, какие-то данные нужно было обрабатывать по-разному) и по каждомуциальному дню в строках. Шелкнув в любой из ячеек, можно было показать, в каких именно транзакциях наблюдаются отклонения. Хотя мы не

показали систему в общепринятом смысле, мы все же продемонстрировали этот отчет. Это помогло команде и владельцу продукта убедить пользователей в том, что после поставки данная новая программа окажется вполне работоспособной.

Я лично сталкивался со многими ситуациями, подобными той, которая описана Бродуоллом. Команда принимает на себя дополнительное обязательство создать нечто такое, что можно было бы с успехом продемонстрировать пользователям или заказчику во время обзора спрингта. Каждый раз меня интересует в таких случаях главным образом то, как часто этот обходной маневр приносит пользу самой команде. Нередко эти дополнительные усилия помогают команде протестировать систему и позволяют ей с меньшими затратами сил и времени проанализировать возможные непредвиденные результаты такого тестирования.

ВОЗРАЖЕНИЕ

“Мы безнадежно провалили все сроки сдачи нашей системы заказчику. Доказывать ценность проделанной нами работы после каждого спрингта — совершенно непозволительная для нас роскошь”.

Человек, выдвигающий подобные аргументы, так и не понял всей важности и полезности итеративного подхода к выполнению проекта. Надлежащее применение итеративного подхода не требует от исполнителей чрезмерных дополнительных усилий. Да, потребуются автоматизированные тесты, чтобы командам не приходилось тратить недели на повторное тестирование каждого спрингта вручную. Но итеративная работа не требует от исполнителей каких-то дополнительных усилий в конце каждого спрингта, которые в конечном счете не оправдывали бы себя. Все, что требуется от команды в данном случае, — это логические точки останова, когда команда может продемонстрировать пользователям или заказчику достигнутые ею результаты.

“В течение нескольких первых месяцев мы не сможем продемонстрировать пользователям или заказчику достигнутые нами результаты”.

Подобные представления характерны для команд, которые выполняют проекты, рассчитанные на год и более. Сталкиваясь с подобной точкой зрения, важно помнить, что двумя главными преимуществами создания работоспособного программного обеспечения к окончанию каждого спрингта являются возможность скорейшего получения информации обратной связи и гарантия того, что члены команды никогда не впадут в заблуждение (пусть даже совершенно искреннее) о достигнутом ими прогрессе.

Я допускаю, что в случае некоторых проектов мы не сможем получить от конечных пользователей ценной обратной связи в течение нескольких первых месяцев выполнения проекта, но я по-прежнему настаиваю на том, что к окончанию каждого спрингта команда должна создавать завершенную, работоспособную и протестированную функциональную возможность. Тем не менее, если эта функциональная возможность не является чем-то таким, что мои конечные пользователи могли бы увидеть, пощупать и оценить (т.е. осознать ее ценность для себя), то я должен позаботиться о том, чтобы ценность выполненной мною работы смог понять мой владелец продукта. При условии, что владелец продукта сможет строго и объективно судить о том, удалось ли команде сделать шаг в правильном направлении (всегда отдавая предпочтение работоспособному программному обеспечению, а не документации), команда вряд ли будет предпринимать неэффективные действия, когда большой объем проделанной работы будет давать небольшой результат.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- В конце каждого из последующих трех-пяти спринтов позаботьтесь о том, чтобы разработанное вами программное обеспечение оказалось в руках его будущих пользователей. Если формальный выпуск продукта вашего типа нецелесообразен, найдите дружелюбно настроенных первых пользователей, которые смогут сообщить вам свое мнение о качестве разработанных функциональных возможностей. По завершении последнего спринта проанализируйте, насколько полезной оказалась эта мера.
- По каждому из элементов журнала запросов на выполнение работ из последующих трех спринтов попросите команду указать конкретного человека или группу людей, для которых тот или иной элемент может представлять реальную ценность. По завершении этих спринтов проанализируйте урок, извлеченный вами из указанной меры, и оцените, насколько он оказался полезным для вас.

Готовьтесь в текущем спринте к следующему

Однажды мне позвонил директор по разработкам одной из компаний и попросил меня выступить в роли наставника по отношению к трем его командам. Начало использования Scrum проходило у них достаточно гладко, правда, планирование спринтов у каждой из команд продолжалось не менее трех дней. Я не мог понять, как такое вообще возможно, поэтому изъявил желание лично пообщаться с этими командами и посмотреть, как они работают. Я рисовал в своем воображении, что они закрываются на все три дня где-нибудь в конференц-зале и погружаются в бесконечные дебаты о том, как разделить задачи по каждому из элементов журнала запросов на выполнение работ, над которыми им предстояло работать, или пытаются чрезмерно детализировать каждую из задач. Однако то, что я увидел в действительности, оказалось спринтами типа “бильярдный шар”.

Спринты типа “бильярдный шар”

Покатите бильярдный шар по столу, постаравшись попасть им в другой бильярдный шар. В результате попадания этот другой бильярдный шар также покатится по столу. В случае спринтов типа “бильярдный шар” команда, завершив один спринт, оказывается не готовой к началу следующего спринта, в результате чего начало следующего спринта переносится на более отдаленную дату. Второй спринт зачастую начинается лишь номинально: команда оказывается настолько неподготовленной к проведению этого спринта, что ей приходится тратить несколько дней на то, чтобы уяснить, чем именно ей предстоит заниматься. Именно это и происходило с командами моего клиента, который пожаловался мне на то, что планирование спринтов у каждой из команд длится не менее трех дней.

См. также Правильное ведение журнала запросов на выполнение работ, о котором рассказывалось в главе 13, “Журнал запросов на выполнение работ”, является одним из способов подготовки к очередному спринту непосредственно во время выполнения текущего спринта.

Наилучший способ избежать спринтов типа “бильярдный шар” — следовать пионерскому лозунгу “Будь готов!” В каждом спринте нужно уделять какое-то время подготовке к следующему спринту. Кен Швабер (Ken Schwaber) рекомендует выделять примерно 10% времени каждого спринта подготовке к следующему спринту (2009). Мне кажется, что в целом этого вполне достаточно, чтобы подготовиться к следующему спринту. Конечно, каждая команда может скорректировать объем времени, уделяемого подготовке к следующему спринту, опираясь на собственный опыт.

Включайте в спринт только реальный объем работы

Мы уже знаем, что команде не следует включать в спринт пользовательскую историю (или какой-либо другой элемент журнала запросов на выполнение работ), если она заведомо велика для того, чтобы ее можно было выполнить в течение одного спринта. Эпическую пользовательскую историю, реализация которой требует нескольких месяцев, следует расчленить на более мелкие части, каждую из которых можно завершить в течение одного спринта. То же самое можно сказать о слишком неопределенных, “расплывчатых” пользовательских историях: если какая-то история недостаточно хорошо понятна, чтобы ее можно было завершить в течение одного спринта, то ее не следует включать в спринт. Вместо этого команде нужно потратить какое-то время на предварительное изучение этой истории.

Обратите внимание: я воспользовался словосочетанием “достаточно хорошо понятна”, а не “абсолютно понятна”. Чтобы пользовательская история, представленная в журнале запросов на выполнение работ, была включена в спринт, она вовсе не обязательно должна быть продумана во всех подробностях. Более того, необязательно должны быть продуманы *во всех подробностях* даже элементы журнала запросов на выполнение работ. Вместе с тем владелец продукта и другие члены команды должны в ходе спринта тесно сотрудничать между собой в деле уяснения истории. Но каждая из пользовательских историй, включенных в спринт, должна быть уяснена с достаточной степенью детализации, чтобы ее можно было, подкрепив соответствующими дискуссиями в ходе спринта, успешно завершить к моменту окончания спринта.

Чтобы выяснить, как это осуществляется на практике, рассмотрим ситуацию проектирования пользовательского интерфейса. Некоторые из пользовательских историй нуждаются в значительных объемах проектирования пользовательского интерфейса. Допустим, что команда решила реализовать такую пользовательскую историю: “*Будучи пользователем, я могу просматривать диалоговое окно “О программе”, в котором указываются владелец авторского права, номер версии и контактная информация компании, что позволяет мне найти контактную информацию компании-изготовителя*”. Мне кажется, что даже в течение достаточно короткого (например, двухнедельного) спринта проектировщики пользовательского интерфейса могли бы смоделировать пару-тройку экранов, предложить некоторым пользователям поработать с этими экранами, получить от этих пользователей информацию обратной связи и, воспользовавшись ею, за-программировать и протестировать подходящие варианты экранов. Иными словами, эта пользовательская история сама по себе вполне приемлема. Краткого описания, представленного на учетной карточке, вполне достаточно для проектирования, кодирования и тестирования этой истории в рамках одного спринта.

Далее, рассмотрим действия команды, добавляющей новые функциональные возможности к сайту электронной торговли, недавно запущенному в эксплуатацию.

Команде показывают новую пользовательскую историю: “*Будучи уже зарегистрированным клиентом, я могу аннулировать еще невыполненный заказ, что дает мне возможность изменить свое мнение без каких-либо финансовых издержек для себя*”. Эта история предполагает возникновение новых потоков работы, которые не учитывались при запуске в эксплуатацию первоначального варианта рассматриваемого нами сайта электронной торговли. В ходе совещания, посвященного планированию спринта, проектировщики пользовательского интерфейса, входящие в состав этой команды, выявляют следующие задачи.

- Создать три первоначальные имитации в Photoshop, по 12 часов каждая
- Запланировать демонстрационные варианты с 15 пользователями, 2 часа
- Провести четыре демонстрационных сеанса, всего — 8 часов
- Собраться, чтобы обсудить изменения в дизайне, 4 часа
- Создать новый дизайн в Photoshop, 8 часов
- Запланировать второй раунд демонстрационных вариантов, 2 часа
- Провести еще четыре демонстрационных сеанса, всего — 8 часов
- Переписать HTML и CSS с учетом окончательных изменений, 16 часов

Это потребует несколько большего объема работ, чем требовалось для истории, касающейся диалогового окна “*О программе*”. Члены команды обсуждают план и приходят к выводу о невозможности выполнения всей этой работы — плюс кодирование полного экрана в соответствующее приложение и его тестирование — в течение одного спринта. Но они полагают, что если бы по крайней мере первые демонстрационные сеансы были проведены в рамках первого спринта, то они смогли бы реализовать данную пользовательскую историю уже в следующем спринте. Именно так и решает поступить команда.

Обратите внимание: когда начинается второй спринт и команда включает историю в свой спринт, она не начинает свою работу с полностью специфицированной пользовательской истории. Окончательные подробности будут определены в течение спринта. Объем подробностей, которые должны сопровождать тот или иной элемент журнала запросов на выполнение работ, включаемый в спринт, является минимально необходимым для того, чтобы в течение спринта данный элемент мог быть реализован в виде работоспособной и протестированной функциональной возможности. Для каждого отдельно взятого элемента журнала запросов на выполнение работ это будет выглядеть по-разному.

ВОЗРАЖЕНИЕ

“*Непохоже, чтобы это было в духе методологии Scrum. Считается, что в каждом спринте Scrum-команды должны создавать продукт, потенциально готовый к поставке заказчику*”.

Scrum-команда в любом случае должна создавать продукт, потенциально готовый к поставке. Но было бы недальновидным полагать, что все свое рабочее время в течение спринта команда должна потратить только на развитие продукта, запланированное для данного спринта. Члены команды тратят время на любые виды деятельности, которые представляются им важными, причем вовсе необязательно, чтобы эта деятельность

была непосредственно связана с развитием продукта, запланированным для текущего спринта. Например, затраты времени на проведение собеседований с работниками, намеревающимися стать членами данной команды, являются инвестицией, которая не окупится до тех пор, пока эти работники не станут членами данной команды. Оценивание элементов журнала запросов на выполнение работ, предпринимаемое с целью приоритизации журнала владельцем продукта, не связано непосредственно с развитием продукта, запланированным для текущего спринта; тем не менее это очень важный вид деятельности. Не меньшую ценность представляет и время, затраченное на то, чтобы обеспечить понимание работы, запланированной на следующий спринт, с такой степенью детализации, которая окажется достаточной для ее успешного выполнения.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Обсудите журнал запросов на выполнение работ. Определите пять самых высокоприоритетных элементов, которые необходимо заблаговременно продумать. Для каждого из этих элементов обсудите, кто именно из членов команды должен заняться таким продумыванием (архитектор? проектировщик пользовательского интерфейса? разработчик базы данных? кто-то другой?), и определите, с каким запасом времени (выраженном в количестве спринтов) следует начать это продумывание.
- ❑ В ходе трех последующих обзоров спринтов обсудите, был ли в достаточной степени детализирован каждый из элементов журнала запросов на выполнение работ и был ли он добавлен своевременно.
- ❑ В течение одного-двух спринтов попытайтесь отслеживать количество времени, затраченного на продумывание дальнейших действий. Оказались ли эти затраты времени достаточными? Не оказались ли они чрезмерными? Помните, что продумывание дальнейших действий должно занимать примерно 10% рабочего времени, имеющегося в распоряжении команды.

Работайте вместе

Компания Apple всегда славилась своим новаторством. В числе самых значительных инноваций Apple в эпоху персональных компьютеров были Apple II, Macintosh, а также iPod. У Стива Джобса (Steve Jobs), основателя Apple, спросили, как его компания удаётся неизменно входить в число самых новаторских компаний мира. Он ответил в форме истории.

Вам, наверное, приходилось видеть на мировых выставках по-настоящему крутые автомобили, которые, будучи спустя четыре года доведеными до стадии серийного производства, в конечном счете оказывались весьма заурядными, ничем не привлекательными транспортными средствами. Почему такое происходит? Это становится возможным, потому что гениальная идея, родившаяся в головах проектировщиков и воплощенная в замечательном выставочном образце, попадает впоследствии в руки инженеров и оборачивается очередной банальностью. Увидев выставочный образец, инженеры говорят: “Нет, довести такую модель до стадии серийного производства невозможно”. В результате из рук инженеров выходит

нечто, совершенно непохожее на оригинал и ощутимо хуже оригинала. Затем этот плод инженерных трудов попадает в руки производственников, которые говорят: “Нет, мы не в состоянии наладить серийное производство такой модели!” и продолжают ухудшать ее (Grossman 2005, 68).

Из приведенной выше цитаты можно сделать вывод о необходимости глубокого сотрудничества между членами опытной Scrum-команды. Вместо того чтобы передавать работу от одной группы к другой, команда, работающая над реализацией Scrum-проекта, должна быть межфункциональной командой, члены которой тесно сотрудничают между собой. В Apple “продукты не передаются от одной команды к другой. Здесь не предусмотрены дискретные, следующие друг за другом стадии разработки. Напротив, разработка носит параллельный и органичный характер. Продукты разрабатываются параллельно и одновременно всеми подразделениями компании (проектирование, техническое обеспечение, программное обеспечение). Этот процесс имеет характер бесконечных циклов междисциплинарных обзоров проектных решений (Grossman 2005, 68).

Это, конечно же, легче сказать, чем сделать. Очень легко, например, попасть в ловушку последовательного выполнения работ в рамках спринтов. Команда, оказавшаяся в такой ловушке, может решить, что первую неделю спринта нужно посвятить анализу, вторую — проектированию, третью — кодированию, а четвертую — тестированию. Очевидно, что последовательный подход к выполнению работы, запланированной для спринта, неэффективен. Такой подход чреват большими потерями времени, чрезмерной специализацией и многочисленными передачами работы из рук в руки. К счастью, несмотря на то, что многие Scrum-команды начинают работать именно так, для большинства из них очень быстро становятся очевидными проблемы, порождаемые таким подходом. Столкнувшись с этими проблемами, они начинают искать способы параллельного выполнения работы. Цель заключается в том, чтобы добиться как можно большего распараллеливания разных видов деятельности, требующихся для того, чтобы перейти от идеи к продукту, потенциально готовому к отправке заказчику.

См. также В главе 11, “Организация коллективного труда”, излагались рекомендации о том, как члены команды могут распараллеливать свою работу.

Вначале поиск способов повышения степени распараллеливания разных видов деятельности покажется вам нелегким делом. Но большинство команд вскоре осознает, что значительную помощь в этом деле могут оказать приемы гибкой методологии разработки. Например, написание автоматизированных тестов исходного кода позволяет сократить количество ошибок, что, в свою очередь, дает возможность растянуть программирование на большую часть спринта и предоставить время для другого тестирования. Разработка программного обеспечения на основе составления тестов (особенно на основе составления приемочных тестов) объединяет анализ, проектирование и кодирование с тестированием.

См. также Информация об этих методах приведена в главе 9, “Технические приемы”.

Избегайте спринтов, посвященных определенным видам деятельности

Хороший Scrum-мастер постоянно подталкивает команду к внедрению более совершенных технических приемов, которые помогают ей научиться эффективно распараллеливать свою работу. Если команда не осваивает эффективные способы распараллеливания своей работы, то ее члены могут прийти к использованию менее желательного подхода: планированию спринтов, посвященных определенным видам деятельности. Спринты, посвященные определенным видам деятельности, — весьма порочная практика. В этом случае команда решает целиком посвятить один спринт анализу и проектированию, второй спринт — кодированию, а третий — тестированию, как показано на рис. 14.1. При таком подходе команда делит свою работу на три последовательно выполняемые части: в первом спринте работают аналитики, во втором — программисты, в третьем — тестеры.



Рис. 14.1. Планирование спринтов, посвященных определенным видам деятельности, — порочная идея

Такой подход кажется довольно привлекательным. Он не только решает, на первый взгляд, проблему, связанную с тем, как распараллелить работу, но и позволяет каждому из специалистов работать в компании таких же специалистов, как он сам (а это кажется весьма привлекательным для тех, кто привык работать именно в таком “формате” и кому в результате внедрения Scrum приходится привыкать к другому “формату” — тесному сотрудничеству с всеми остальными членами Scrum-команды). К сожалению, спринты, посвященные определенным видам деятельности, обладают теми же недостатками, что и команды, специализирующиеся на определенных видах деятельности: чрезмерное количество передач работы из рук в руки, а также отсутствие ответственности за выполненную работу у команды в целом. Спринты, посвященные определенным видам деятельности, характеризуются еще тремя важными недостатками.

См. также Подробнее о проблемах с командами, специализирующимиися на определенных видах деятельности, рассказывается в главе 10, “Структура команды”.

- **Повышенный риск составления неправильного календарного плана.** Планирование объема работы, который может быть выполнен в рамках одного спринта, оказывается более трудной задачей, поскольку такое планирование сильно зависит от качества работы, выполненной в предыдущем спринте. Например, программисты до тех пор не будут знать, какой объем работы потребуется от них в спринте, посвященном тестированию, пока тестеры не приступят к своей работе. Это означает, что программисты не будут знать, какой объем работы нужно запланировать для спринта, посвященного кодированию.
- **Более продолжительный путь от идеи к работоспособной протестированной функциональной возможности.** Планирование спринтов, посвященных определенным видам деятельности, порочно не только само по себе. Оно также удлиняет время, которое требуется для получения информации обратной связи от заказчиков, пользователей и других заинтересованных лиц.
- **Планирование таких спринтов не решает проблему распараллеливания работы.** Когда вся работа выполняется в течение одного спринта, команда действует в одном ритме. Члены команды помогают друг другу и осваивают смежные дисциплины, пытаясь таким образом достичь общекомандной цели. В случае планирования спринтов, посвященных определенным видам деятельности, мы приходим к дроблению команды на отдельные группы, действующие с разными скоростями. Это приводит, например, к тому, что после какого-то периода вынужденного бездействия тем или иным группам приходится наверстывать упущенное время, работая в авральном режиме. В итоге получается, что скорость работы команды ограничивается скоростью работы самой медленной из групп, входящих в ее состав. И это еще не все: накопление работы приводит к тому, что содержащиеся в ней ошибки будут выявлены и исправлены лишь после того, как в действие вступят группы, расположенные в самой нижней части потока.

Замените отношения “от окончания к началу” отношениями “от окончания к окончанию”

Одна из наибольших проблем планирования спринтов, посвященных определенным видам деятельности, заключается в том, что они создают отношения типа “от окончания к началу”. В отношениях такого рода одна задача должна быть завершена до того, как может начаться следующая. Например, при выполнении последовательного проекта диаграмма может указывать, что, прежде чем начнется кодирование, должен быть завершен анализ, а прежде чем начнется тестирование, должно завершиться кодирование. Хорошие Scrum-команды понимают, что так не должно быть: многие виды деятельности можно распараллелить. Важно не то, когда начинаются задачи, а то, когда они заканчиваются. Кодирование не может завершиться до тех пор, пока не завершится анализ, а тестирование не может завершиться до тех пор, пока не завершится кодирование. Такие отношения называются отношениями “от окончания к окончанию” и подкрепляются самим механизмом Scrum-спринтов. Вся работа к концу спринта либо завершается, либо возвращается в журнал запросов на выполнение работ.

После приобретения хотя бы небольшого опыта большинство команд узнает, как распараллеливать те или иные виды работ и формировать между ними отношения “от окончания к окончанию”. Команды легко находят способы распараллеливать

обсуждения потребностей пользователей и программирование. Они так же быстро находят способы распараллеливать программирование и тестирование. Эти виды деятельности очень удачно вписываются в итеративный и инкрементный подходы: узнать у пользователей кое-какие детали об их потребностях, а затем реализовать какую-то часть этих потребностей; затем, реализовав эти потребности, протестировать то, что реализовано.

Другие виды деятельности вписываются в итеративный и инкрементный подходы не столь удачно. Проектирование пользовательского интерфейса, проектирование баз данных и архитектура часто относятся к тем категориям работ, которые нужно выполнять заранее. Неспособность рассматривать эти виды деятельности как единое целое неминуемо ведет к новым проблемам.

Распараллеливание проектирования пользовательского интерфейса

Рассмотрим подробнее проектирование пользовательского интерфейса (User Experience Design — UED), чтобы разобраться, как некоторые Scrum-команды успешно включили в свои спринты проектировщиков пользовательского интерфейса. Понимание того, как это делается в случае UED, позволит вам понять, что делать в случае проектирования баз данных, архитектуры, а также других видов деятельности, которые, как вначале может показаться, не очень подходят для гибкой методологии разработки.

В случае проектов, выполняемых традиционным методом, UED обычно рассматривается как работа, которая обычно выполняется до того, как начнутся другие действия, связанные с разработкой программного обеспечения. Работа по проектированию пользовательского интерфейса разделяется на определенную последовательность этапов, которая обычно начинается с оценки текущих методов работы и потребностей пользователей и завершается созданием проектов пользовательского интерфейса. Эти проекты создаются и оцениваются итеративным способом. Разработанные проекты пользовательского интерфейса демонстрируются потенциальным пользователям, выясняется их мнение и в проекты вносятся соответствующие корректизы. Полученный результат вновь демонстрируется потенциальным пользователям. Таким образом, проектировщики пользовательского интерфейса в какой-то мере уже привыкли работать итеративно. Точнее говоря, они привыкли осуществлять итерации еще до начала выполнения остальной части проекта. Однако в случае Scrum-проекта нам не хотелось бы, чтобы работа начиналась с предварительной фазы UED. Вместо этого желательно, чтобы проектировщики пользовательского интерфейса работали параллельно с другими членами команды. Важность тесного сотрудничества проектировщиков пользовательского интерфейса с другими членами команды подтверждает Дезире Сай (Desirée Sy), проектировщик гибкого взаимодействия из компании Autodesk.

Помимо поддержания тесных контактов со всей командой посредством ежедневных совещаний, мы работаем в тесном контакте с разработчиками на протяжении всего периода проектирования и разработки... Проектировщикам гибкого взаимодействия приходится общаться с разработчиками практически каждый день. Это нужно не только для того, чтобы гарантировать надлежащую реализацию проектных решений, но и для того, чтобы у всех нас было исчерпывающее понимание технических ограничений, которые влияют на проектные решения (2007, 126).

Нужно, чтобы во время спринта все члены команды, независимо от специализации каждого из них, работали вместе над достижением общекомандного результата. Но не следует забывать, что во время спринта команда преследует две цели: выполнить всю работу, запланированную для текущего спринта, и подготовиться к предстоящему спринту. Естественно, разные члены команды будут тратить неравные количества своего времени на эти разные цели. Большинство программистов тратят большую часть своего времени на добавление новых функциональных возможностей. Проектировщики пользовательского интерфейса, с другой стороны, обычно тратят большую часть своего времени на получение максимально полной информации о функциональных возможностях, которые еще только предстоит реализовать. Они выясняют дополнительные подробности, которые добавляются к сложным элементам журнала запросов на выполнение работ. Но — и это очень важно! — они также тратят время на уточнение и разъяснение вопросов относительно проектных решений, программируемых и тестируемых в текущем спринте. Несмотря на то что проектировщики (или архитекторы, или технические дизайнеры), работающие в данной команде, могут тратить какое-то время на то, чтобы “заглянуть в будущее”, они остаются членами команды, работающими над реализацией задач текущего спринта.

В результате мы получаем нечто, подобное представленному на рис. 14.2. На этом рисунке показано, как во время кодирования и тестирования одной части журнала запросов на выполнение работ проектировщики пользовательского интерфейса потратят какую-то часть своего времени (возможно, даже большую его часть) на изучение элементов, расположенных в нижней части журнала запросов на выполнение работ, т.е. элементов, которые предстоит реализовать в будущем. Тем не менее они остаются членами единой команды, работающей в каждый отдельный момент времени только над одним спринтом.

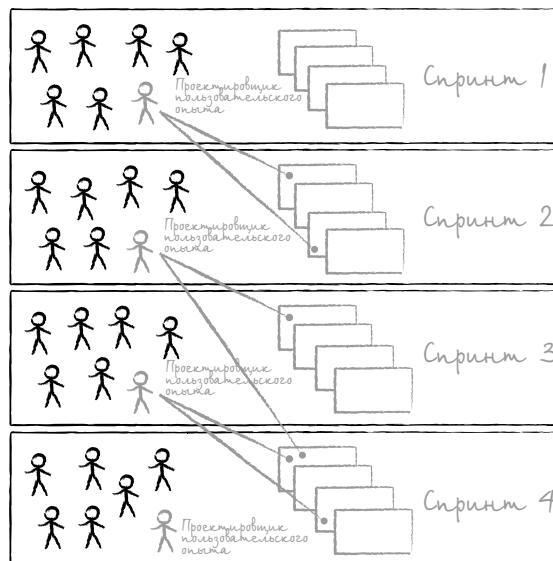


Рис. 14.2. Проектировщики пользовательского интерфейса работают над текущим спринтом, но при этом тратят часть времени на “заглядывание в будущее”

Мыслите целостно, работайте инкрементно

А как же насчет того, что UED (и другая работа такого же типа, например архитектура или проектирование базы данных) должно выполняться глобально, т.е. с использованием целостного взгляда на разрабатываемую систему? Ответ на этот вопрос определяется тем, как выбирается работа из журнала запросов на выполнение работ. На рис. 14.2 показано, как проектировщики пользовательского интерфейса, входящие в состав команды, зачастую посвящают какую-то (иногда даже большую) часть своего времени работе, которую предстоит выполнить в ходе последующих спринтов. Когда речь идет о разработке приложения, существенной составляющей которой является проектирование пользовательского интерфейса, владелец продукта должен (учитывая мнение знающих членов команды) приоритизировать журнал запросов на выполнение работ, помня о необходимости разрешения проблем, связанных с UED.

Владелец продукта и разработчики должны представлять себе разрабатываемую ими систему в целостном виде, решая, приобретение каких новых знаний является для них наиболее актуальным. Эти области становятся фокальными точками для работы, связанной с UED, в следующем спринте. Таким образом, разработчики работают над системой отдельными фрагментами, так называемыми “проектными порциями” (design chunks), по образному выражению Дезире Сай, проектировщика гибкого взаимодействия из компании Autodesk (2007, 120). Проектная порция — это небольшой (размером в один спринт) фрагмент системы, который встраивается в конструкцию системы в целом. Сай говорит, что “проектировщики взаимодействия привыкли рассматривать те или иные явления в их целостном виде, поэтому разбиение конструкции на отдельные фрагменты (особенно на такие, которые поначалу не поддерживают потоки работы) может поначалу показаться очень нелегким делом, но мастерство приходит с опытом. Разбиение конструкции на отдельные фрагменты (проектные порции) таит в себе немало преимуществ” (2007, 120).

Одно из преимуществ итеративного способа работы проектными порциями — возможность сократить объем работы, выполняемой понапрасну, т.е. работы, которая воплощается в функциональных возможностях, впоследствии оказывающихся ненужными. Такой способ работы позволяет команде переключать проектировщиков пользовательского интерфейса (или архитекторов, или разработчиков баз данных) на решение наиболее важных проблем в самые подходящие моменты времени. Дезире Сай пришла к выводу, что проектирование типа “к нужному моменту” в конечном счете приводит к более удачным проектным решениям.

Мы пришли к выводу, что новые способы гибкой методологии разработки, ориентированные на пользователя, приводят к более удачно разработанным продуктам, чем при использовании “водопадных” вариантов тех же методов. Гибкие способы коммуникации позволяют сократить разрыв между выявлением проблем с удобством использования и принятием мер, направленных на решение этих проблем, путем внесения в продукт соответствующих изменений (2007, 112).

Линн Миллер (Lynn Miller), директор разработки пользовательских интерфейсов компании Autodesk, полагает, что разработчикам целесообразно было бы заглянуть несколько дальше в журнал запросов на выполнение работ и заниматься соответствующими пользовательскими историями до того, как за них примется остальная команда. Линн Миллер указывает на следующие три преимущества такого подхода.

Во-первых, не придется понапрасну тратить время “на выполнение работ, которые окажутся невостребованными”. Во-вторых, “тестирование удобства пользования функциональными возможностями и контекстуальное изучение проектного решения могут выполняться одновременно при появлении заказчика в компании (т.е. заказчик сможет реже появляться у исполнителей)”. В-третьих, поскольку разработчики смогут “своевременно получать информацию обратной связи”, в случае “внезапных изменений на рынке (например, при появлении конкурирующего программного обеспечения)” они смогут своевременно “узнать об этом и принять соответствующие меры” (2005, 232).

Эти преимущества подтверждаются Мариссой Мейер (Marissa Mayer), вице-президентом Google по поисковым продуктам и пользовательскому интерфейсу.

В случае Toolbar Beta прототипы нескольких ключевых функциональных возможностей (пользовательские кнопки, совместно используемые закладки) были разработаны менее чем за неделю. По сути, в ходе сеанса мозгового штурма мы обсудили в пять раз больше ключевых функциональных возможностей, от многих из которых мы отказались после того, как в течение недели создавали соответствующие прототипы. Поскольку плодотворной оказалась лишь одна из каждых 5–10 идей, данная стратегия ограничения времени, которое отводится для доказательства плодотворности идей, позволяет нам проверять большие идеи в более сжатые сроки, что повышает наши шансы на успех (Porter, 2006).

Во многих отношениях этот способ работы посредством выработки тех или иных проектных решений ничем не отличается от того, чем хорошие проектировщики занимались всегда. Никто из проектировщиков пользовательского интерфейса не запирает дверь, чтобы потратить несколько недель на поиск “идеального проектного решения” и затем предъявить разработчикам это идеальное решение для реализации. Напротив, хороший проектировщик постоянно держит в голове проект в целом и пытается выявить нерешенные проблемы, способные оказать наибольшее влияние на конечный вариант проекта. Затем хороший проектировщик пытается найти решение этих нерешенных проблем путем обсуждений с пользователями, создания прототипов, выполнения обзоров проектных решений и даже неспешного размышления. Когда удается решить одну совокупность нерешенных проблем (или по крайней мере сузить ее до меньшей совокупности вариантов), проектировщик принимается за следующую совокупность нерешенных проблем. Квалифицированный и опытный проектировщик уже делает то, что я описал: мыслит целостно, но действует итеративно, занимаясь поиском решений.

Архитектура и проектирование баз данных

Я утверждал, что проектирование пользовательского интерфейса, архитектура и проектирование баз данных — это три частных случая одной и той же общей проблемы: итеративное выполнение тех видов деятельности, которые традиционно выполнялись на ранней фазе процесса последовательной разработки. Чтобы убедиться в правильности моего вывода, рассмотрим способ принятия архитектурных решений при разработке коммерческого продукта с помощью гибкой методологии разработки.

Клаус занимал должность архитектора в средней по величине компании, которая приступила к разработке продукта для управления документооборотом научных

данных. Несмотря на то что у владельца продукта, Робин, сформировалось твердое мнение о структуре данного продукта, она знала, насколько важно начать разработку, опираясь на прочный фундамент. Помня об этом, при выполнении приоритизации работ для первых спринтов Робин внимательно рассмотрела предложения Клауса о том, какие пользовательские истории следует реализовать в первую очередь, чтобы создать такой прочный фундамент.

В первом спринте команда создала функциональные возможности для чтения файлов в формате CSV, т.е. файлов, содержащих значения, разделенные запятыми. В конечном счете этот продукт должен был уметь читать много разных форматов файлов, но Клаус и другие предложили начать с самого простого. Работа, запланированная для этого первого спрингта, не устранила многих архитектурных неопределенностей, но позволяла команде гарантировать себе успешный первый спрингт. Кроме того, она позволяла команде легко вводить данные в систему, что, как было понятно каждому участнику проекта, должно было принести немалую пользу при выполнении последующих спринтов.

В ходе второго спрингта команда могла бы продолжать действовать в том же духе, добавив возможность чтения всех остальных форматов файлов, которые могли бы понадобиться со временем. Но Клауса это не особенно волновало. По его мнению, это не было связано с серьезными архитектурными рисками или областями неопределенности. Если данная система могла загружать файлы в формате CSV, то со временем она смогла бы загружать файлы в формате XML и т.д. Что по-настоящему беспокоило Клауса, так это потребности данного продукта в визуализации огромных массивов данных. Вся надежда была на использование какого-либо коммерческого пакета визуализации данных. Но на тот случай, если ни один из таких коммерческих пакетов не обеспечит производительности, необходимой для визуализации чрезвычайно больших объемов данных, предусматривавшихся этим продуктом, был составлен запасной план, который заключался в написании функциональных возможностей визуализации собственными силами. Итак, планируя второй спрингт, команда и владелец продукта согласились задействовать две разные выбранные на тот момент библиотеки визуализации и создать с помощью каждой из них одну простую визуализацию. Это не разрешило бы вопрос о достижении приемлемой производительности, но по крайней мере команда бы знала, к чему может привести использование каждого из этих двух коммерческих продуктов.

Что же касается плана на третий спрингт, то Клаус порекомендовал разработать сложную визуализацию с использованием каждого из коммерческих продуктов (т.е. продуктов, разработанных сторонними компаниями). Это позволило бы команде оценить производительность и пригодность каждого из этих продуктов. К счастью, один из этих коммерческих продуктов функционировал на должном уровне. Поэтому в ходе четвертого спрингта команда воспользовалась именно этим продуктом для разработки двух других сложных, но очень разных визуализаций в качестве дальнейшего теста данного продукта.

Кроме того, в ходе четвертого спрингта команда приступила к работе над реализацией ряда математических возможностей, необходимых для данного продукта. Например, как объяснил мне Клаус в своем письме, отправленном по электронной почте, пользователи хотели, чтобы у них была возможность отдавать системе приказы “добавлять значения в четвертое и девятое поля импортируемого файла, а затем фильтровать этот

набор данных таким образом, чтобы он включал лишь те элементы, сумма которых по крайней мере в два раза превышает значение, содержащееся в первом поле". Эффективность этой работы не заботила Клауса, но он сказал владельцу продукта, что ее следует начать как можно раньше по двум причинам. Во-первых, хотя разработать этот тип функциональных возможностей можно было многими способами, выбранный подход повлиял бы на очень многие последующие проектные решения. Во-вторых, в продукт предстояло добавить сотни разных правил, подобных указанному, так что разработка первых нескольких таких правил позволила бы им сразу же подключить к работе еще две команды, которые сосредоточились бы исключительно на этой теме. В табл. 14.1 указана работа, выполненная в ходе каждого из спринтов, и причины, по которым была выполнена именно эта работа, а не какая-либо другая.

Таблица 14.1. Архитектурный риск и неопределенность могут повлиять на последовательность выполнения работ

Спринт	Цель	Причина
1	Импорт данных из файлов, содержащих значения, разделенные запятыми	Начать с простого. Получить возможность легко вводить данные в систему, поскольку вся последующая работа основывается на использовании этих данных
2	Создание простой визуализации данных с помощью двух отдельных коммерческих пакетов	Уяснить, что влечет за собой использование двух продуктов-кандидатов
3	Разработка сложной визуализации с помощью каждого из пакетов	Понять, может ли один из продуктов обеспечить потребности в сложной визуализации
4	Разработка двух новых очень разных визуализаций. Добавление первоначальных математических функций	Убедиться в пригодности выбранного продукта визуализации. Определить оптимальный способ добавления математических функций и создать базу для быстрого добавления аналогичных правил другой командой

Как видно из этого примера, Клаус держал в голове всю архитектуру и замысел данного продукта, но использовал спринты для итеративной реализации соответствующих проектных решений. Приоритизацией работы всегда занимался владелец продукта, и к окончанию каждого спринта создавалась очередная новая порция работоспособного программного обеспечения. Однако некоторые из функциональных возможностей, разрабатывавшихся в ходе каждого спрингта, реализовывались по предложению архитектора проекта и его коллег. Соображения, связанные с проектом базы данных и проектированием пользовательского интерфейса, могут точно так же учитываться в решениях о том, над чем именно следует работать.

Временные рамки: неизменные и строгие

Когда я только приступил к итеративной и инкрементной разработке (это было еще до того, как я стал использовать гибкую методологию разработки), я совершил ошибку, заключающуюся в том, что не все наши спринты были одинаковой продолжительности. Мы собирались в начале каждого спрингта, чтобы составить план работ на этот спрингт. Одним из пунктов повестки дня таких совещаний, в ходе которых мы составляли планы своих первых спринтов, было планирование продолжительности спрингта. Обычно продолжительность наших спринтов составляла от двух до шести недель.

Решение о том, какой должна быть продолжительность предстоящего спрингта, принималось исходя из оставшегося объема работы; из объема работы, которую нам нужно было выполнить для того, чтобы продемонстрировать пользователям готовые функциональные возможности; из реального количества исполнителей (дело в том, что кто-то из членов нашей команды мог отсутствовать по уважительным причинам на протяжении какой-то части предстоящего спрингта); и из запаса энергии и усталости, которые мы ощущали. В начале проекта у нас заметно преобладали шестинедельные спрингты. (К тому же наши обеденные перерывы были тогда гораздо более продолжительными.) К концу же проекта преобладали двухнедельные спрингты.

В то время нам казалось, что переменная продолжительность спрингтов является удачным решением. Правда, нужно признать, что это решение не было осознанным: мы даже не задумывались тогда, насколько оправданным является такое решение. Зато впоследствии я уяснил преимущества неизменной продолжительности спрингтов. Эти преимущества перечислены ниже.

- **Неизменный ритм работы идет на благо командам.** При переменной продолжительности спрингтов члены команды зачастую даже не помнят календарного плана. В подобных ситуациях часто приходится слышать, например, такие вопросы: “Сейчас последняя неделя спрингта?” или “Работа должна быть готова в этот четверг или в следующий?” В случае неизменной продолжительности спрингтов (от одной до четырех недель) командам легче приспособиться к определенному — наиболее подходящему для них — ритму работы.
- **Становится легче планировать спрингты.** Как планирование спрингтов, так и планирование выпуска готовых частей работы упрощаются, когда команда придерживается одинаковой продолжительности своих спрингтов. Планировать спрингты становится легче, потому что по прошествии нескольких (как правило, двух-пяти) первых спрингтов команда уясняет, какой объем работы можно запланировать на один спрингт.
- **Становится легче планировать выпуск готовой продукции.** Как будет показано в следующей главе, Scrum-команды составляют свои планы выпуска готовой продукции по возможности эмпирическим путем. Они оценивают оставшийся объем работы по данному проекту, а затем измеряют объем работы, выполняемый в течение одного спрингта. Если продолжительность спрингтов является переменной величиной, измерить скорость работы команды сложнее: нет никакой гарантии, что в ходе четырехнедельного спрингта удастся выполнить объем работы, в два раза больший, чем в случае двухнедельного спрингта. Нормализация скорости (т.е. приведение ее к “скорости за неделю”) несколько помогает делу, но оказывается совершенно излишней при использовании фиксированной продолжительности спрингтов.
- **Так поступил бы Ричард Фейнман.** Нобелевский лауреат физик Ричард Фейнман вспоминает, как он понял, что устал каждый вечер выбирать, что бы такое съесть на десерт. Поняв это, он принял решение каждый вечер покупать шоколадное мороженое (1997, 235). Выбирать продолжительность спрингта в начале каждого спрингта — напрасная траты времени и сил. Поэкспериментировав с двумя-тремя вариантами продолжительности спрингтов, выберите подходящий для себя вариант и придерживайтесь этого решения до тех пор, пока у вас не возникнет серьезная причина, чтобы его изменить.

Подчеркивая, что ваши спринты должны иметь одинаковую продолжительность, я вовсе не имею в виду, что вам необходимо на этом “зациклиться”. Выберите определенный день недели, который лучше всего подходит для ваших конкретных обстоятельств, и начинайте все спринты в этот день. Мне нравится начинать спринты в пятницу, что дает нам возможность полностью посвятить этот день обзорам спринтов, ретроспективе и совещаниям по планированию спринтов. Если бы мы выбрали в качестве такого дня понедельник, то возненавидели бы этот день еще больше, чем мы ненавидим его сейчас.

Но бывают ситуации, когда можно несколько отклониться от этого правила. Типичной причиной этого могут быть праздники. В Соединенных Штатах Америки принято, например, брать отгулы ближе к таким праздникам, как День благодарения и Рождество. Команда, которая пользуется двухнедельными спринтами, может оказаться в ситуации, когда в такие предпраздничные (или послепраздничные) дни на работу выходит от силы половина ее состава. В таком случае команда может так заранее спланировать, например, трехнедельный спринт, включающий праздничные, предпраздничные и послепраздничные дни, чтобы в течение этого спринта можно было выполнить обычный объем работы, выполняемой в ходе регулярного двухнедельного спринта (учитывая реальное количество исполнителей этого объема работы).

ВОЗРАЖЕНИЕ

“В некоторых случаях изменение продолжительности спринтов оправдано. Я не хотел бы, чтобы какое-либо жесткое правило препятствовало этому”.

Согласен. Никакую рекомендацию, в том числе и эту, нельзя подавать как абсолютное правило, которому нужно следовать всегда. Допустим, мы пользуемся двухнедельными спринтами и у нас остается три недели до начала крупной отраслевой выставки, на которой мы хотим продемонстрировать свой продукт. Почему бы нам в таком случае не провести один трехнедельный спринт или обычный для нас двухнедельный спринт и однодневный спринт? Было бы очевидной глупостью не воспользоваться целой неделей для встраивания в продукт новых функциональных возможностей только из-за того, что окончание используемого нами двухнедельного спринта не совпадает с датой начала выставки.

Никогда не продлевайте спринт

Вторая ошибка, которую можно допустить при использовании спринтов, заключается в нестрогом отношении к временным рамкам, которыми ограничен спринт. Как бы ни складывались обстоятельства, спринты нужно завершать вовремя. Ни в коем случае не должно быть так, что, подойдя к дате окончания спринта, вы принимаете решение продлить его на пару-тройку дней потому, что не успели завершить запланированную работу.

Допустим, мы выполняем проект продолжительностью в один год. Повредит ли успешной его реализации продление на пару дней первого спринта? Несомненно! Если члены команды решат продлить этот первый спринт, они придут к выводу, что в нарушении сроков нет ничего страшного. Между тем это очень плохо, даже если речь идет о нарушении сроков в самом начале выполнения проекта, когда кажется, что такие изменения вряд ли существенно повлияют на успешную реализацию проекта.

Строгое соблюдение сроков, отведенных для каждого спринта, укрепляет уверенность исполнителей проекта в том, что выполнение проекта идет строго по плану, а это повышает шансы на успешное его завершение. По завершении каждого очередного спринта команда должна получать очередное усовершенствование продукта, которое при необходимости можно было бы поставить заказчику. Если временные рамки спринтов не соблюдаются (“Давайте выполним на этот раз шестинедельный спринт, поскольку нам придется работать над архитектурными элементами”) или расширяются под благовидными предлогами (“Чтобы выполнить запланированную работу, нам нужно еще три дня”), дисциплина утрачивается.

Строгое соблюдение временных рамок спринтов означает, что время от времени командам придется мириться с тем, что им не удалось выполнить всю работу, запланированную на тот или иной спринт. Не исключено, что команде удастся компенсировать этот неприятный, но вполне реальный исход, время от времени добавляя работу в тот или иной спринт. Как будет показано в главе 15, “Планирование”, неспособность выполнить всю работу, запланированную на тот или иной спринт, еще не означает конец света, если работа выполняется в порядке приоритетности.

Если появляются какие-то признаки того, что не удастся выполнить всю работу, запланированную на тот или иной спринт, владелец продукта должен встретиться с командой, чтобы обсудить меры, необходимые для исправления ситуации. Желательно, чтобы такая встреча состоялась как можно ближе к началу спринта, чтобы владелец продукта мог своевременно принять решение, какую работу нужно выполнить обязательно, а какой можно и пожертвовать. Если же команда не обнаружит проблему до 18-го дня 20-дневного спринта, то владельцу продукта уже просто не из чего будет выбирать (имеется в виду выбор работы, которой можно пожертвовать): ему придется отказаться от реализации той функциональной возможности, которую команда ни при каких обстоятельствах не успевает реализовать в данном спринте.

ВОЗРАЖЕНИЕ

“Нам требуется определенная гибкость, чтобы реагировать на пожелания клиента или заказчика”.

Оперативная реакция на сообщения об ошибках или на запросы о реализации новых функциональных возможностей является целью, которую ставят перед собой многие организации. Это вполне достойная цель, но важно понимать, что подлинной целью является оперативная реакция в целом и на протяжении длительного времени, а не на один конкретный запрос (понимая, конечно, что подчас один конкретный запрос может исходить от клиента, на которого приходится 80% вашего бизнеса). Вообще говоря, для большинства клиентов более важна не наша способность время от времени быстро исправить ошибку, выявленную клиентом, а уверенность в том, что выявленная ошибка будет исправлена к обещанному нами сроку. Ограничиваая себя четкими временными рамками, мы повышаем собственную предсказуемость в глазах наших клиентов: нам заранее известны даты, когда будут начинаться и заканчиваться спринты, и поэтому срок выполнения запроса мы можем четко увязать с датой завершения того или иного спринта.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если вы используете спринты переменной длины, выберите продолжительность спрингта, которая кажется вам оптимальной, и постарайтесь четко придерживаться этой продолжительности в течение следующих двух месяцев. После этого примите решение, следует ли изменить продолжительность спрингта.
- Если вы используете спринты четырехнедельной, или месячной, продолжительности, попытайтесь перейти к двухнедельным спринтам. Поначалу они могут показаться вам слишком короткими, поэтому, прежде чем решить, будет ли ваш переход к двухнедельным спринтам окончательным, выполните три таких спрингта.
- Проигнорируйте мою рекомендацию, касающуюся нежелательности частого изменения продолжительности спрингтов, выполнив два однодневных спрингта. Бегуны, специализирующиеся на марафонской дистанции, во время тренировок нередко совершают забеги со скоростью, существенно превышающей марафонскую. Такие ускоренные забеги помогают повысить скорость бега, которую спортсмен способен поддерживать на протяжении длительного времени. Аналогия с разработчиками программного обеспечения в этом случае представляется вполне уместной. В ходе ретроспективы обсудите ход и результаты выполнения однодневных спрингтов. Проанализируйте, нельзя ли заимствовать из опыта выполнения этих однодневных спрингтов что-либо полезное, чем можно было бы воспользоваться после того, как ваша команда вернется к спрингтам обычной продолжительности.

Не меняйте цель

Когда я был еще начинающим программистом, мне пришлось работать в отделе юридического консалтинга одной крупной консалтинговой компании. Характерной особенностью работы в этом отделе была неизбежность изменений. Наше начальство — адвокаты, которые занимались какими-то судебными тяжбами — запускало проект, выполнение которого должно было продолжаться две-три недели. Когда проект был уже наполовину готов, адвокаты вбегали в комнату, где работали программисты, и командовали: “Прекращайте работу! Другая сторона запросила то-то и то-то. Теперь от вас требуется...” После этого они “перенацеливали” нас на новый проект, по завершении которого нам нужно было вернуться к прежнему, приостановленному, проекту.

Учитывая этот печальный опыт, неудивительно, что одной из самых привлекательных — по крайней мере для меня — особенностей методологии Scrum стала неизменность цели, преследуемой командой, на протяжении всего спрингта. Подход Scrum к изменениям носит двоякий характер. В течение спрингта ничто не должно меняться. В первый день спрингта команда составляет план работ на весь спрингт и исходит из того, что выбранные ею приоритеты останутся неизменными на протяжении всего спрингта. Однако несмотря на недопустимость каких-либо изменений в течение спрингта, окружающий мир может измениться за это время весьма существенно.

Может показаться, что установка Scrum на недопустимость каких-либо изменений в течение спрингта способна повредить успеху проекта. В конце концов, изменения бывают настолько важными, что обойтись без них невозможно. В других случаях появление новой информации может сделать бессмысленной работу, которой команда

занимается в настоящее время. В любом случае я призываю вас придерживаться (по крайней мере, поначалу) категорического неприятия методологией Scrum каких-либо изменений по ходу спринта.

Чтобы обосновать свою точку зрения, предлагаю рассмотреть примеры обоих типов этих обоснованных, на первый взгляд, причин внесения изменений по ходу спринта. Сначала рассмотрим случай обнаружения владельцем продукта какого-то важного нового требования, которое, по его мнению, нужно выполнить вместо работы, которой команда занимается в настоящее время. Иногда так случается. Когда это происходит в моей практике, я предлагаю сделать соответствующее изменение цели спринта видимым. Scrum позволяет осуществить это следующим образом: команда объявляет об *аварийном завершении спринта*, которое сопровождается немедленным планированием нового спринта, включающего реализацию только что выявленной высокоприоритетной функциональной возможности. Обеспечение видимости изменений в целях спринта важно потому, что это позволяет снизить вероятность изменений. В очень многих организациях единственными, кто ощущают на себе постоянное перенацеливание команды, являются сами члены команды. Неприятие методологией Scrum каких-либо изменений по ходу спринта и ее готовность к аварийному завершению текущего спринта и началу нового спринта обеспечивают видимость цены и частоты изменений. Это позволяет сократить до минимума изменения, которые приходится вносить команде. Лишь самые важные и абсолютно необходимые изменения могут оправдать аварийное завершение текущего спринта.

Теперь рассмотрим случай поступления новой информации, которая делает менее желательным выполнение работы, запланированной на текущий спринт. Как и в предыдущей ситуации, команде действительно может стать известно нечто такое, что должно заставить ее прекратить выполнение одной из работ, запланированных на текущий спринт. Например, цель текущего спринта может заключаться в добавлении определенной функциональной возможности, что поможет совершить продажу крупному клиенту. В ходе выполнения спринта этот клиент сообщает вам, что его компания испытывает финансовые затруднения и не может купить ваш продукт даже при наличии в нем указанной функциональной возможности или что его компанию поглотила другая компания и теперь им придется воспользоваться продуктом вашего конкурента, или называет вам какие-то другие причины, препятствующие покупке вашего продукта. В подобных случаях нужно, конечно же, прекращать работу по реализации соответствующей функциональной возможности (правда, при этом следует учитывать желательность данной функциональной возможности для других клиентов и то, насколько далеко продвинулась работа по ее реализации). Однако ситуации, похожие этой, случаются не так часто, как может показаться большинству из тех, кто лишь приступает к освоению Scrum.

С гораздо более типичной ситуацией столкнулась Дженис. Исполняя роль владельца продукта по отношению к некоторому приложению из области биоинформатики, она работала вместе с командой над проектированием весьма запутанного экрана для поиска сложных данных. Предложенный вариант оказался далеким от идеального, но никто не смог предложить ничего лучшего, и в целом этот экран устраивал всех. Во время очередного совещания, посвященного планированию спринта, все согласились с тем, что нужно разрабатывать экран поиска в соответствии с предложенным прототипом.

Работа продвигалась успешно в течение первой половины двухнедельного спринта этой команды. На седьмой день утром Дженис объявила, что накануне вечером на нее снизошло озарение. Она продемонстрировала разработчикам набросок более удачного, на ее взгляд, экрана поиска. Он не имел практически ничего общего с вариантом, который Дженис предложила ранее. Безусловно, новый вариант был лучше прежнего. Никто даже не пытался это оспорить. Дженис и команде оставалось лишь решиться на одно из двух.

- Остановить текущий спринт и начать новый спринт, полностью сосредоточившись на новом варианте экрана поиска, признанном более удачным, чем предыдущий
- Продолжить работу над прежним вариантом экрана поиска, признанным вполне удовлетворительным семь дней назад

Кому-то может показаться, что выбор между этими двумя вариантами не так уж сложен. Новый вариант экрана поиска был заведомо лучше, поэтому разработчикам следовало немедленно переключиться на его реализацию. Однако методология Scrum помогла им сформулировать этот вопрос несколько по-другому: поставить заказчику продукт с вполне удовлетворительным вариантом экрана поиска, имея в распоряжении семь дней для разработки дополнительных функциональных возможностей, или поставить заказчику продукт только с более удачным вариантом экрана поиска? Окончательное решение оставалось за Дженис, но в выработке этого решения принимала участие вся команда. Коллективно команда решила завершить реализацию вполне удовлетворительного варианта экрана поиска. Усовершенствованный вариант экрана поиска был реализован девять месяцев спустя в следующей версии продукта.

ВОЗРАЖЕНИЕ

“Особенностью нашего бизнеса являются постоянные изменения, поэтому нашим разработчикам не остается ничего другого, как время от времени бросать наполовину сделанную работу и приступать к новой”.

Типичной организацией, которой постоянно приходится переходить от одной работы к другой, является отделение экстренной (неотложной) медицинской помощи в больнице. Изменения происходят буквально на каждом шагу, и главная забота для руководства отделения — правильно определить очередность оказания помощи пациентам в зависимости от тяжести каждого конкретного заболевания или полученной травмы. Однако организации со столь стремительно меняющейся обстановкой встречаются сравнительно редко. Во всяком случае, большинство организаций имеет возможность расставить приоритеты в начале каждого двухнедельного спринта и неукоснительно их соблюдать. Многим организациям кажется, что они действуют в условиях, близких к тем, в которых приходится действовать отделениям экстренной медицинской помощи. Между тем это не соответствует действительности. Просто нужно привыкнуть продумывать ситуацию хотя бы на два-три хода вперед. Если же ваша организация относится к числу тех, для которых внезапные, практически постоянные изменения являются фактом жизни (подобно отделу юридического консалтинга, о котором рассказывалось в начале этого раздела), то вам, возможно, следует использовать более короткие спринты.

Откажитесь от перенацеливания команды

Выработав на протяжении многих лет привычку постоянного перенацеливания своих команд с одной задачи на другую, организациям бывает очень нелегко с ней расстаться. Зачастую командам приходится бросать на полпути одну работу и приступить к другой не потому, например, что владелец продукта выявил у заказчика какую-то важную и неожиданную потребность, а потому, что владелец продукта или другие заинтересованные лица попросту не удосужились продумать ситуацию на несколько ходов вперед. Они привыкли работать по старинке и даже не дают себе отчета в негативных последствиях такого подхода для команд разработчиков.

Я заметил это в одной компании, которая специализировалась на оказании услуг другим компаниям. Ее бизнес строился в основном на налаживании отношений с партнерами. Добавление нового партнера требовало пяти-десяти часов работы со стороны команды разработчиков. Эта команда привыкла рассматривать свою работу как не поддающуюся планированию, поскольку она никогда не знала, какое количество партнеров добавит торговый представитель в течение одного из их двухнедельных спринтов.

Я счел, что это снижает производительность команды, и решил поговорить с торговым представителем. Я сказал ему, что отныне команда будет добавлять только тех партнеров, которые подписали контракты до начала соответствующего спринта. Он удивил меня, сказав, что с ним не будет никаких проблем. Он рассказал мне, что важным преимуществом для партнеров является то, что свой веб-сайт компания обновляет каждые две недели. Потенциальные партнеры поймут, что “запуски” партнеров будут увязываться с обновлениями веб-сайта и что, если они хотят, чтобы “запуск” состоялся к определенной дате, им придется подписать контракт к началу соответствующего спринта. Этот торговый представитель не видел никаких проблем в том, что ему придется работать так, как выгоднее команде, а раньше он не делал так лишь потому, что никто не просил его об этом или не указывал ему на невыгодность нынешнего способа работы.

Впоследствии свои требования можно ослабить

Обычно я советую Scrum-командам начать с выражения своего категорического неприятия внесения каких-либо изменений в середине спринта. Я делаю это вовсе не потому, что категорически не приемлю перенацеливание команды с одной задачи на другую, и не потому, что слепо подчиняюсь тем или иным правилам Scrum. Это объясняется тем, что я хочу помочь понять не членам команды, что за такое перенацеливание приходится платить определенную цену. Разумеется, бывают ситуации, когда без перенацеливания команды в середине спринта просто не обойтись. Но слишком часто команды перебрасывают с одних задач на другие потому, что это очень легко сделать, а также потому, что никто не желает продумывать ситуацию хотя бы на два-три хода вперед. Я ослабляю это категорическое неприятие внесения каких-либо изменений в середине спринта после того, как вижу, что организация уже не рассматривает поступление каждого нового запроса как чрезвычайное обстоятельство, стоящее того, чтобы вносить те или иные изменения в середине спринта.

ВОЗРАЖЕНИЕ

“Умение оперативно реагировать на запросы клиента — вот что приносит нам успех; пользователи ожидают этого от нас”.

Действительно, многие организации наладили выгодные отношения со своими пользователями или клиентами за счет того, что им удается оперативно реагировать на их запросы. Однако многие из таких организаций испытывают определенные трудности, будучи вынужденными постоянно демонстрировать свою сверхоперативность. Мой собственный опыт свидетельствует о том, что на самом деле клиентов интересует предсказуемость. Большинство из них понимают, что некритичные ошибки невозмож но исправить немедленно. Они понимают, когда им говорят: “Это, несомненно, важный вопрос, и мы хотим взяться за его решение как можно быстрее. Мы вносим в систему исправления каждые две недели. Мы уже не в состоянии запланировать внесение этого изменения в выпуск, который появится в ближайшую пятницу, но мы обязательно внесем его в выпуск, который появится через две недели”. Поскольку организации, занимающиеся разработкой программного обеспечения, не обеспечивали им такого уровня предсказуемости в прошлом, они привыкли к тому, что лучше всего требовать внесения всех исправлений **прямо сейчас**.

ВОЗРАЖЕНИЕ

“Scrum — это, прежде всего, гибкость. Если необходимость в изменении возникает в середине спрингта, мы должны уметь внести это изменение”.

Scrum — это не только гибкость, но и максимизация командой ценности, которую она обеспечивает заказчику на протяжении длительного времени. Хороший способ добиться этого — предоставить команде возможность сосредоточиться на какой-то одной цели и, лишь добившись ее, переориентировать ее на другую цель. Повышение гибкости организации в ее способности реагировать на стратегические изменения — это не то же самое, что умение сверхоперативно реагировать буквально на каждый очередной чих клиента — тем более что такой подход приводит к неудовлетворительным результатам в долгосрочной перспективе.

Получайте отзывы, учитесь и приспосабливайтесь

Каждый спринт можно рассматривать как эксперимент. Владелец продукта и команда собираются в начале спрингта, чтобы определить наиболее ценный эксперимент, который они могут провести. Такой эксперимент предполагает создание какого-то объема новой функциональности в форме работоспособного программного обеспечения. Это новое расширение возможностей продукта реализуется таким образом, чтобы его можно было поставить заказчику и получить максимальный объем информации от потребителя по данному эксперименту. В конце спрингта выполняется оценка эксперимента, и команда в целом извлекает из него определенные уроки.

Большинство этих уроков касается продукта, создаваемого командой. Что нравится пользователям? Что им не нравится? Что вызывает у них путаницу? Что они хотят получить в следующий раз? О каких функциональных возможностях (тех, о которых они раньше даже не думали) это новое расширение позволяет им думать? Возможно, не меньше уроков, полученных командой в результате очередного эксперимента, касается использования командой самой методологии Scrum. Какой объем работы мы можем выполнить в течение спринта? Что мешает успешному выполнению работы? Что могло бы помочь нам работать быстрее? Удаётся ли нам в результате каждого спринта создавать работоспособное программное обеспечение?

Большинство новых знаний оказалось бы бесполезным, если бы разработка по методологии Scrum не обладала свойством итеративности. Спринт за спринтом владелец продукта, команда и Scrum-мастер могут возвращаться к не вполне удовлетворительным, но работоспособным реализациям и совершенствовать их. К экрану поиска данных могут быть добавлены дополнительные поля. На основе отзывов клиентов пользовательский интерфейс переделывается из приемлемого в отличный. Основываясь на собранных данных, выполняется настройка параметров в самых важных частях системы. Исходя из уточненного понимания объема работы, который может быть выполнен в течение трех месяцев, оставшихся до завершения проекта, корректируются планы и реприоритизируется работа.

Преимущества итеративной и инкрементной разработки заключаются в обратной связи, извлечении определенных уроков на основе полученной информации и последующем внесении корректировок в то, что мы создаем и как мы создаем. Спринты обеспечивают команды механизмами для успешного решения этих задач.

Дополнительная литература

Appelo, Jurgen. 2008. We increment to adapt, we iterate to improve. *Methods & Tools*, Summer, 9–22.

В статье Аппело содержатся превосходные описания итеративной и инкрементной разработок. В ней показано также, как то и другое привносят в процесс гибкой разработки что-то важное, но свое, особенное.

Cockburn, Alistair. 2008. Using both incremental and iterative development. *Crosstalk*, May, 27–30.

В этой статье содержатся превосходные определения итеративной и инкрементной разработок и приводятся убедительные аргументы в пользу их совместного использования.

Larman, Craig, and Victor R. Basili. 2003. Iterative and incremental development: A brief history. *IEEE Computer*, June, 47–56.

Это обзор итеративной и инкрементной разработок, в котором их корни прослеживаются начиная с 1950-х годов. Авторы статьи доказывают, что итеративная и инкрементная разработка не относятся к числу модных и быстропреходящих увлечений.

Sy, Desirée. 2007. Adapting usability investigations for agile user-centered design. *Journal of Usability Studies* 2 (3): 112–132.

Это наилучшее из имеющихся описаний того, как включить проектирование пользовательского интерфейса в процесс гибкой разработки.

Глава 15

Планирование

“Мы придерживаемся гибкой методологии разработки. Зачем нам планирование?” и “Мы выполним свою работу тогда, когда выполним” — вот типичные высказывания, которые мне приходилось слышать в первые годы после опубликования Agile Manifesto. Я подозреваю, что многие из тех, кто были членами первых команд, внедривших у себя гибкую методологию разработки и придерживавшихся подобных взглядов, понимали, что лишаются чего-то ценного, когда отказываются от планирования. Но это было естественной реакцией людей на предшествующие культурные среды, в которых им приходилось работать. Очень многие разработчики ненавидели планирование, поскольку в планировании они не усматривали лично для себя никакой выгоды. Более того, зачастую планы были оружием, которое использовалось против разработчиков: “Вы сказали, что выполните эту работу к июню. Сейчас июнь. Покажите мне готовую работу”.

Использовать планы в качестве оружия против разработчиков было неразумно. Столь же неразумно было бы и полностью отказаться от планирования. Как бывший вице-президент по новым разработкам в ряде компаний, в которых гибкая методология разработки являлась решающим фактором успеха, могу с уверенностью заявить, что Scrum-команды могут и должны заниматься планированием. Более того, согласно результатам исследования, проведенного Кьетил Молоккен-Оствольд (Kjetil Moløkken-Østvold) и Магне Йоргенсен (Magne Jørgensen), Scrum-команды (и вообще команды, пользующиеся гибкой методологией разработки) не только имеют все основания заниматься планированием, но и осуществляют такое планирование тщательнее, чем команды, придерживающиеся последовательного метода разработки (2005).

Планирование является фундаментальным аспектом Scrum. Scrum-команды всегда стремятся работать над реализацией функциональных возможностей, имеющих самую высокую ценность. Для этого команда совместно с владельцем продукта должна оценить стоимость разработки той или иной функциональной возможности; в противном случае приоритизация работ осуществляется лишь по критерию желательности. Аналогично важно оценить продолжительность разработки той или иной

функциональной возможности: функциональная возможность, которая не попадает в критичное “рыночное окно”, будет обладать гораздо меньшей ценностью. Очевидно, что для Scrum-команды, которая взяла на себя обязательство работать в порядке приоритетности функциональных возможностей, планирование является важной составляющей деятельности.

В этой главе мы выйдем за рамки фундаментальных основ Scrum и рассмотрим некоторые проблемы планирования, с которыми, как свидетельствует мой собственный опыт, по-прежнему сталкиваются многие организации, внедрившие у себя Scrum. Материал этой главы начинается с рассмотрения необходимости постепенного уточнения планов (вместо того чтобы начинать проект с составления полностью детализированного плана). Затем я расскажу о том, почему сверхурочная работа не является разумным способом решения проблем, вызванных ошибками в календарном планировании. Я попытаюсь доказать, почему организациям следует отдавать предпочтение изменению масштаба, а не других важных параметров планирования проекта, таких как календарный план, ресурсы или качество. Завершается глава советом не смешивать оценки и обязательства.

Постепенно уточняйте свои планы

В главе 13, “Журнал запросов на выполнение работ”, говорилось о том, что журнал запросов на выполнение работ нуждается в постепенном уточнении. Функциональные возможности, которые будут добавлены в отдаленном будущем, поначалу заносятся в журнал запросов на выполнение работ как эпические истории, которые впоследствии расчленяются на все меньшие имена пользовательские истории. Постепенно эти истории становятся настолько малы, что дальнейшее их расчленение утрачивает смысл. После этого они в последний раз уточняются путем добавления условий удовлетворенности, которые описывают тесты верхнего уровня, используемые впоследствии для определения, была ли полностью реализована соответствующая история.

Хорошая Scrum-команда применяет аналогичный подход и к планированию. Точно так же, как эпическая история описывает сущность определенной функциональной возможности, игнорируя при этом детали, первоначальный вариант плана отражает сущность того, что должна сделать команда, оставляя подробности “на потом”. Последующие варианты плана добавляют необходимые подробности, но лишь тогда, когда эти подробности могут подкрепляться знанием, полученным на тот момент в результате реализации проекта. Исключение подробностей из первоначального варианта плана не означает, что мы не можем взять на себя обязательства о том, что будет включено в проект к его завершению. Мы по-прежнему можем брать на себя обязательства, но в этих обязательствах должно оставаться место для изменений, соответствующих степени неопределенности данного проекта.

Рассмотрим, например, ситуацию команды, которая разрабатывает новый веб-продукт, предназначенный для построения генеалогического дерева. Работа должна быть завершена не позже, чем через шесть месяцев, и должна быть способна реализовать именно то, что может быть создано к этому времени. Команда может указать множество подробностей о самых важных функциональных возможностях (которым должен быть назначен самый высокий приоритет в журнале запросов на выполнение работ). Если более высоким приоритетом обладает составление генеалогических деревьев

вручную, то первоначальный вариант плана будет включать значительное число подробностей, касающихся этой функциональной возможности. В журнале запросов на выполнение работ может упоминаться монтажная сетка (layout grid), привязка к ней элементов, отображение масштабных линеек, ручная вставка разделителей страниц и т.д.

Функциональная возможность, расположенная в журнале запросов на выполнение работ ниже, будет включать меньше подробностей. Мы могли бы написать: “*Будучи пользователем, я хочу загружать фотографии таким образом, чтобы я мог связать их с тем или иным человеком, представленным в генеалогическом дереве*”. Впоследствии это обеспечивает команде и владельцу продукта гибкость, необходимую для поддержки только JPG- или GIF-файлов, несмотря на то что поначалу была надежда обеспечить поддержку семи или восьми форматов изображений.

ВОЗРАЖЕНИЕ

“Мы выполняем разработку на основе договоров с привлечением сторонних разработчиков. Прежде чем подписать договор, наши заказчики желают знать, работа с какими форматами файлов будет реализована. Я не могу игнорировать подобные детали”.

Несмотря на то что откладывание подробностей реализации “на потом” предоставляет команде большую гибкость в вопросе поиска наилучшего решения, вы не обязаны постепенно уточнять описания всех функциональных возможностей. Если форматы файлов с изображениями так важны для ваших заказчиков, то вы правы. Их нужно тут же детализировать в журнале запросов на выполнение работ и в планах, которые прилагаются к соответствующему договору. Другие, менее критичные, функциональные возможности могут уточняться по мере того, как они будут подниматься к верхней части журнала запросов на выполнение работ.

Постепенное уточнение плана обладает многими преимуществами. Главные из них перечислены ниже.

- **Минимизирует затраты времени.** Планирование необходимо, но оно связано с определенными затратами времени. Время, потраченное на оценивание и планирование, целесообразно рассматривать как инвестицию: мы согласны инвестировать в планирование лишь в той мере, в какой вознаграждаются наши усилия. Если мы составляем подробный план проекта в самом начале выполнения этого проекта, то такой план неизбежно будет основываться на массе предположений. В ходе выполнения проекта мы придем к выводу, что какие-то из наших предположений оказались ложными, и это обесценит основанные на них планы.
- **Позволяет принимать решения в нужное время.** Постепенное уточнение плана помогает команде избежать соблазна принятия слишком большого числа решений в самом начале проекта. Участники проекта день ото дня узнают о своем проекте все больше и больше. Если нет необходимости принимать какое-то решение именно сегодня, т.е. если его принятие можно без ущерба для дела отложить на будущее, то с ним не следует торопиться, тем более что информированность команды день ото дня повышается, а это способствует принятию более обоснованных и разумных решений.

- **Позволяет команде уточнять направление своих действий.** Никогда не следует забывать о том, что “все течет, все меняется”. Планирование, достаточное для того, чтобы команда могла уяснить общее направление своих действий, но предусматривающее отказ от излишней детализации, предоставляет ей определенную свободу маневра, что дает возможность уточнять курс по мере получения новой информации. Обратите внимание: я тщательно избегаю расхожего выражения *корректировка курса*. Это объясняется тем, что, на мой взгляд, вообще невозможно говорить о каком-то заранее известном “правильном курсе”.
- **Помогает команде не оказаться в плену собственного плана.** Как бы хорошо мы ни понимали возможность возникновения любых неожиданностей и невозможность составления такого плана, который не требовал бы внесения в него тех или иных изменений по ходу реализации проекта, тщательно составленный и хорошо документированный план может заставить нас поверить в то, что все можно продумать и предусмотреть заранее. Постепенное уточнение плана помогает команде всегда помнить о том, что даже хорошо продуманный план не может обойтись без внесения изменений.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Проанализируйте текущий план выпуска продукта. Выявите части этого плана, которые кажутся вам чрезмерно детализированными (с точки зрения текущего этапа реализации вашего проекта).
- ❑ Составьте список причин, заставляющих вашу организацию составлять слишком детализированные планы, несмотря на очевидность того, что эти планы впоследствии придется уточнять. Может быть, в вашей организации есть конкретные люди или группы людей, которые требуют составления таких планов? Можно ли убедить их в том, что начинать нужно с составления менее детализированных планов? Если вы можете положительно ответить на этот вопрос, встретьтесь с этими людьми и попытайтесь убедить их в целесообразности составления именно таких планов.

Ниаких сверхурочных для спасения плана

Давным-давно, когда мне только предложили стать руководителем команды разработчиков программного обеспечения, мне казалось, что нет ничего проще. Мой собственный программистский опыт свидетельствовал о том, что программисты, как правило, недооценивают объем работы, который им предстоит выполнить. Мне казалось, что все, что требуется от меня как от руководителя, — это сначала попросить каждого из разработчиков оценить объем предстоящей ему работы, а затем настоять на том, чтобы он уложился в указанные им самим сроки. Поскольку, как я уже сказал, программисты склонны к недооценке предстоящего им объема работы, мне казалось, что мы справимся с работой быстрее, чем в случае, если бы я самостоятельно составил календарный план для этой команды.

В течение первых нескольких месяцев все шло хорошо. Поскольку речь идет о 1980-х годах, наша команда, конечно же, не была Scrum-командой. Для многих из первых задач, включенных в календарный план, были указаны недостаточно четкие

результаты на выходе. Анализ был выполнен тогда, когда мы сами назвали его выполненным. Проектирование было выполнено, когда подошел срок его окончания. Первые несколько функциональных возможностей, которые предстояло запрограммировать, были завершены согласно календарному плану. Чтобы уложиться в запланированные сроки, я заставил нескольких членов команды работать сверхурочно — в конце концов, именно они сами, а не я, оценивали объемы работы. Работать сверхурочно пришлось не так уж много: несколько дополнительных часов на этой неделе и, возможно, полдня в следующую субботу. Но после нескольких месяцев такой работы я заметил, что нам приходится работать сверхурочно все больше, а толку от этого становилось все меньше. Углы, которые мы срезали во время предыдущих периодов авральной работы, досаждали нам впоследствии. К тому же, чем дальше, тем больше ошибок мы находили (или совершали).

Я не придумал тогда ничего лучшего, чем увеличение объемов сверхурочных работ.

Разумеется, это нам не помогло. Это не помогло и при выполнении нескольких последующих проектов, когда я использовал тот же подход. Со временем я понял, что невозможно до бесконечности понукать свою команду и что по достижении некоторого критического уровня дальнейшее наращивание объема сверхурочных работ приводит к снижению производительности команды, вместо того чтобы повышать ее.

В первые дни существования экстремального программирования это называлось *сорокачасовой рабочей неделей* (исходя из восьмичасового рабочего дня, принятого в Соединенных Штатах Америки). Однако вскоре этот принцип был переименован в *приемлемый темп* (*sustainable pace*). Такое переименование явилось отражением того, что стандарты, принятые в других странах, отличались от 40 часов и что иногда бывает целесообразно работать больше, чем 40 часов в неделю. Понаблюдайте за марафонским бегом — и вы увидите, что каждый марафонец бежит в приемлемом лично для него темпе. Однако приглядитесь повнимательнее — и вы заметите, что темп бега каждого марафонца не остается неизменным: он меняется от километра к километру. Каждый из них работает несколько напряженнее на подъемах и, возможно, несколько расслабляется на спусках. На финишной прямой большинство бегунов наращивают темп и бегут на пределе своих возможностей перед самой финишной чертой.

Примерно то же самое термин “приемлемый темп” означает и для Scrum-команды. Большую часть времени команда действует в умеренном, ровном темпе, но время от времени (например, при выходе на финишную прямую или, возможно, когда приходится исправлять серьезную ошибку, обнаруженную пользователями) членам команды приходится включать более высокую скорость. Сверхурочные работы, если ими не злоупотреблять, никоим образом не противоречат цели “работать в приемлемом темпе”. Авторы книги *Extreme Programming Explained* Кент Бек (Kent Beck) и Синтия Andres (Cynthia Andres) согласны с таким выводом.

Сверхурочные работы — это симптом неблагополучной ситуации с проектом.

Правило экстремального программирования формулируется достаточно просто: вы не можете работать вторую неделю сверхурочно. Если речь идет об одной неделе, то несколько часов сверхурочной работы вполне допустимы. Если же вы приходите на работу в следующий понедельник и заявляете: “Чтобы выполнить поставленные цели, нам опять придется поработать сверхурочно”, это говорит о том, что у вас возникла серьезная проблема, которую вряд ли удастся решить, поработав несколько дополнительных часов (2004, 60).

Обходиться без сверхурочной работы трудно

Когда Клинтон Кейт (Clinton Keith) занимал должность главного технического директора High Moon Studios, разработчика видеоигр Triple-A, ему стоило больших усилий уяснить всю важность предостережения насчет чрезмерного увлечения сверхурочными работами. Индустрия видеоигр относится к числу немногих, ежегодные отраслевые выставки которых привлекают к себе повышенное внимание. Отраслевые выставки индустрии видеоигр называются Electronic Entertainment Expo, или Е3. На Е3 демонстрируются важные продукты индустрии видеоигр, которые должны в ближайшее время появиться на рынке. Сделки обычно заключаются между студиями-разработчиками и издателями. Естественно, что команды, стремясь успеть с выпуском своих новых игр к началу такого крупнейшего события года, как Е3, работают ударными темпами, в том числе и сверхурочно. Члены команд хотят не только, чтобы их игры предстали в лучшем виде перед представителями прессы и потенциальными бизнес-партнерами, но и чтобы эти игры впечатлили их друзей из других компаний, которые планируют посетить выставку.

Многие годы Клинтон Кейт заставлял свои команды в преддверии очередной выставки Е3 месяцами работать сверхурочно. Но после того как в High Moon Studios внедрили Scrum, команды работают в постоянном, приемлемом для себя темпе. Тем не менее старые привычки дают о себе знать. За несколько недель до открытия очередной отраслевой выставки Клинтон Кейт попросил свои команды, в добровольно-принудительном порядке, поработать сверхурочно. Как и следовало ожидать, в первую неделю производительность команд возросла (рис. 15.1). Но уже во вторую неделю, несмотря на то что производительность команд оставалась выше, чем без сверхурочных работ, она снизилась по сравнению с первой неделей. На третьей неделе производительность оказалась лишь незначительно выше, чем при отсутствии сверхурочных работ. На четвертой неделе сверхурочной работы производительность оказалась даже ниже той, которую команды обычно демонстрировали, работая в постоянном, приемлемом для себя темпе (Keith, 2006).

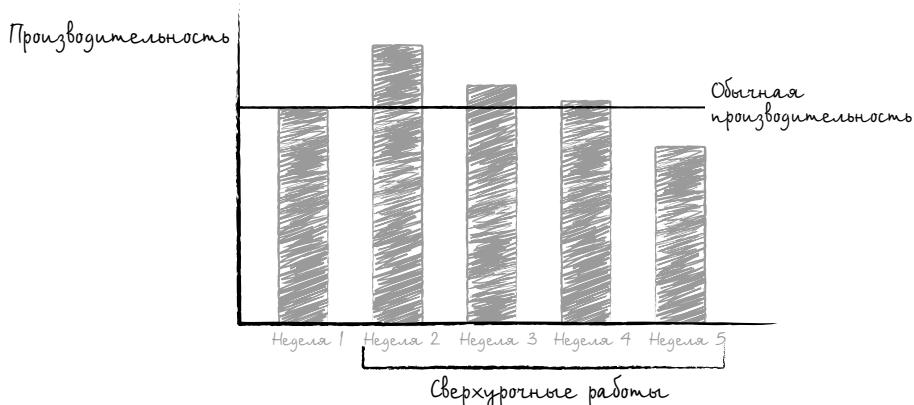


Рис. 15.1. High Moon Studios пришла к выводу, что последующие недели сверхурочной работы фактически вели к снижению производительности команд (публикуется с разрешения Клинтона Кейта, отдел гибкой методологии разработки игр)

Несмотря на то что иногда избавиться от прочно укоренившихся привычек очень нелегко, после того как руководитель на практике столкнется с фактом, подобным описанному выше, и его команды наглядно продемонстрируют ему, что чрезмерное увлечение сверхурочными работами не приносит желаемых результатов, полученный урок наверняка пойдет любителю сверхурочных работ впрок.

Как прийти к финишу с наилучшим результатом

Довод в пользу работы в приемлемом темпе гласит, что, работая таким способом, команды добиваются более высоких результатов, чем в случае, когда периоды работы в неприемлемом темпе сменяются периодами расслабления. В графическом виде это представлено на рис. 15.2. Команда, работающая в приемлемом для себя темпе, выполняет один и тот же объем работы в каждый период времени. Команда, работающая в неприемлемом для себя темпе, в какие-то периоды превышает этот объем работы, но в другие периоды ей приходится восстанавливаться после чрезмерного напряжения, и поэтому объем выполняемой ею работы снижается. После анализа рис. 15.2 возникает естественный вопрос: “Превышает ли область, расположенная под кривой приемлемого темпа работы (представляющая совокупный объем работы, выполненной командой), область, расположенную под кривой неприемлемого темпа работы?” Эту ситуацию можно представлять себе по-другому. Допустим, вам нужно пробежать пять километров. В каком случае вы придете к финишу быстрее: если будете бежать всю дистанцию с одинаковой скоростью или если какие-то отрезки этой дистанции будете бежать очень быстро, а на других будете переходить на шаг?

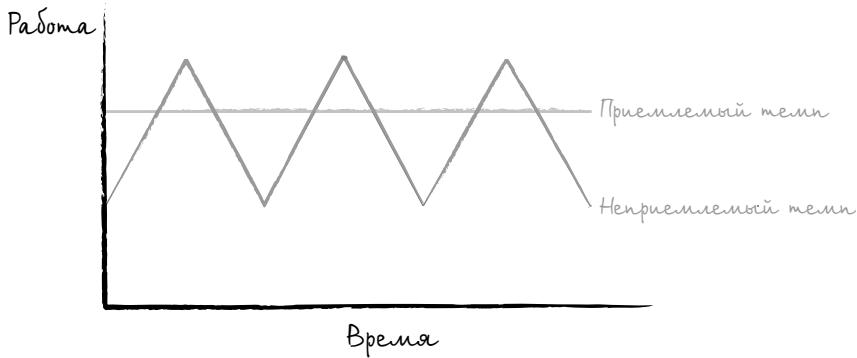


Рис. 15.2. Выполненный объем работы представлен областью, расположенной под кривой

Доводы в пользу того, что работа в постоянном, приемлемом темпе является наиболее продуктивной, вряд ли убедят большинство сомневающихся. В конце концов, вы действительно полагаете, что черепаха (предпочитающая двигаться с постоянной, приемлемой для себя скоростью) способна обогнать зайца (который предпочитает стратегию рывков и остановок)? Я согласен с тем, что басни Эзопа призваны подкрепить великие истины, но, чтобы поверить во что-то, я должен увидеть это “что-то” собственными глазами. То же самое можно сказать об утверждении, касающемся приемлемого темпа: чтобы убедиться в том, что сверхурочные работы не могут быть решением долгосрочных проблем, связанных с календарным планированием,

большинству организаций нужно собрать собственные данные. После того как вы проанализируете данные для своих команд (как это сделал Клинтон Кейт, построив график, представленный на рис. 15.1), вы убедитесь, что чрезмерное увлечение сверхурочными работами не ведет к повышению производительности.

К сожалению, заставить людей работать с постоянной, приемлемой для них скоростью (хотя бы для того, чтобы вы могли собрать соответствующие данные) — не так-то легко. Я нашел следующие доводы в защиту работы с постоянной, приемлемой скоростью.

- **Работа с постоянной приемлемой скоростью позволяет оставить дополнительный потенциал вашей команды на случай, когда этот потенциал действительно понадобится.** Если команда все время работает на пределе своих возможностей, у нее не останется дополнительного резерва энергии на случай, когда этот дополнительный резерв ей действительно понадобится.
- **Работа с постоянной приемлемой скоростью оставляет больший простор для проявления творческих способностей.** Подлинная производительность является не только следствием более продолжительной работы, но и результатом появления времени от времени творческих решений, которые позволяют либо резко сократить сроки выполнения проекта, либо существенно повысить качество продукта. У команд, работающих с постоянной приемлемой для себя скоростью, наверняка останется запас интеллектуальной энергии для того, чтобы генерировать такие идеи.
- **Работа с постоянной приемлемой скоростью позволяет прекратить всякие разговоры о том, что после шестичасового рабочего дня “мозги перестают работать”.** Командам нужно прекратить рассказывать руководству о том, что после шести часов напряженного умственного труда они чувствуют себя совершенно измочаленными и что у них не остается никаких сил для работы сверх шести часов. Многие руководители трудятся по 12 и более часов в день. Возможно, эта работа требует меньшего напряжения умственных сил, но разработчики ничего не добьются, если будут рассказывать таким руководителям, что заниматься программированием более шести часов невозможно. К тому же многие из разработчиков, делающих подобные заявления своим руководителям, приходят домой и посвящают почти все свободное время участию в каком-нибудь проекте с открытым кодом, занимаясь этим исключительно для собственного удовольствия. Повышается заинтересованность — повышается и производительность.
- **Это стоит попробовать.** Если какой-либо эксперимент приводит к появлению полезных, объективных данных, то большинство лиц, принимающих решения, оказывают такому эксперименту поддержку (если он проводится в надлежащее время). В самом начале проекта, когда время еще не очень поджимает, приступайте к сбору данных о скорости работы команды, когда она работает в приемлемом для себя темпе. Впоследствии, когда почувствуете, что без сверхурочных работ не обойтись, не возражайте против них. Напротив, пообещайте команде, что не будете иметь ничего против того, чтобы сверхурочные работы продолжались более двух-трех недель лишь в случае, если собранные данные будут свидетельствовать о том, что скорость работы команды повысилась.

Если не сверхурочные, то что?

Неумеренное использование сверхурочных работ объясняется тем, что это достаточно дешевый, удобный в применении и иногда достаточно эффективный способ решения проблем, вызванных просчетами в календарном планировании. Руководителю ничего не стоит сказать: “Я предлагаю вам поработать в субботу”. Руководителю не придется платить за это слишком большую цену. Если же мы хотим сформировать культуру, лишенную столь привлекательного на первый взгляд и сравнительно дешевого инструмента, как сверхурочные работы, нам нужно предложить что-то взамен него.

Тони Шварц (Tony Schwartz) и Кэтрин Маккарти (Catherine McCarthy) из Energy Project полагают, что им удалось найти подходящее решение. Они указывают, что время является конечным ресурсом: мы не можем наращивать количество часов нашего рабочего дня. Другое дело — энергия, как утверждают они. Ее можно наращивать. Мы понимаем это на интуитивном уровне. Бывают дни, когда мы приходим на работу, словно заряженные энергией, и демонстрируем сверхпроизводительность. Но бывают дни, когда мы только и делаем, что поглядываем на часы в ожидании окончания рабочего дня. Если мы сможем “накачивать” команду энергией, то количество сверхпроизводительных дней увеличится, а количество дней пониженной производительности уменьшится (2007).

См. также Предложения о том, как довести до сведения людей эти представления, обсуждались в главе 12, “Управление самоорганизующимся коллективом”.

Одним из наиболее эффективных способов наращивания энергии является повышение заинтересованности людей в выполняемой ими работе. Чем большую заинтересованность в успехе проекта проявляют его исполнители, тем вероятнее, что они будут трудиться с полной самоотдачей каждый день. Ключевой фигурой в этом отношении является владелец продукта. Владельцы продукта должны доводить до сведения исполнителей проекта убедительные представления о разрабатываемом продукте, заряжая членов команды энтузиазмом и повышая их заинтересованность в работе над этим продуктом.

Еще один хороший метод, сторонниками которого являются Тони Шварц и Кэтрин Маккарти, — использование кратковременных, но регулярных перерывов (2007). Двадцатиминутная прогулка на свежем воздухе или непродолжительная беседа с товарищем по работе способствует восстановлению внимания и энергии, необходимых для решения основной задачи. Приведу пример из собственной практики: при написании этой книги я использовал 30-минутные спринты. В начале каждого такого спрингта я полностью отмежевывался от всех отвлекающих факторов, таких как электронная почта и телефон. Для этого я пользовался 30-минутными песочными часами. В конце каждого получаса я либо переворачивал песочные часы и продолжал работать (если работа спорилась), либо отводил 5–10 минут на то, чтобы проверить электронную почту, ответить на телефонный звонок или просто прогуляться на свежем воздухе.

Франческо Сирилло (Francesco Cirillo) уже давно отстаивает аналогичный подход, который он называет “*tomodoro*”, что в переводе с итальянского означает “помидор”. Суть этого подхода заключается в том, что члены команды работают 30-минутными

“приращениями”. В начале каждого такого приращения кухонный таймер (изготовленный в виде помидора) устанавливается на 25-минутный интервал. В течение этого времени команда работает не покладая рук и не отвлекаясь на такие внешние факторы, как электронная почта, телефон и т.п. Когда таймер срабатывает, члены команды устраивают 5-минутный перерыв. В течение этих пяти минут они могут выйти в коридор, сделать несколько физических упражнений, поболтать о том, о сем и т.п. Запрещается лишь говорить о работе и проверять электронную почту. После каждого четвертого “помидора” Сирилло разрешает своей команде устроить более продолжительный (15–30 минут) перерыв (2007).

Рекомендация Сирилло устраивать дважды в день более продолжительный перерыв в умственной работе соответствует так называемым ультрадианным ритмам человека. Существуют циклы продолжительностью от 90 до 120 минут, в течение которых организм человека переходит из состояния с высокой энергией в состояние с низкой энергией и наоборот. Психолог Эрнест Росси (Ernest Rossi) утверждает: “Основная идея заключается в том, что примерно через каждые полтора часа нужно делать перерывы в работе. В противном случае вы устаете, а ваша мысль рассеивается. Вы начинаете совершать ошибки, раздражаетесь и попадаете во всякие неприятные ситуации” (2002).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Дайте себе слово выполнить следующие два-три спринта без сверхурочных работ. После этого оцените объем выполненной работы, ее качество, а также творческие способности и энергию, продемонстрированные членами команды в течение этих спринтов.
- Рассмотрите возможность использования “помидорного” подхода Франческо Сирилло или какой-либо менее жесткой его разновидности. Поработайте непрерывно в течение 30–60 минут. Затем устройте перерыв продолжительностью 5–10 минут, походите по коридору или побеседуйте с товарищами по работе.

По возможностям предпочитайте изменение масштаба

Институт управления проектами (Project Management Institute — PMI) уже давно нарисовал “железный треугольник”, показанный на рис. 15.3. Этот железный треугольник демонстрирует отношения взаимозависимости между масштабом, затратами (ресурсами) и календарным планом. Этот треугольник рисуется руководителем проекта и вручается заказчику со словами: “Выберите любые две стороны”. Говоря так, руководитель проекта подразумевает: до тех пор, пока у меня есть свобода действий в отношении одного из трех этих измерений, я могу удовлетворить ожидания заказчика в отношении двух других.

В центре железного треугольника, представленного на рис. 15.3, показано качество. Это объясняется тем, что качество — подобно федеральным агентам, арестовавшим Аль Капоне — считается неприкосновенным. К сожалению, в действительности так бывает не часто: качество нередко используется как четвертая сторона “железного треугольника”.



Рис. 15.3. «Железный треугольник» иллюстрирует взаимосвязь между масштабом, ресурсами и календарным планом

При переходе к использованию Scrum ключевые участники проекта, разработчики и владельцы продукта должны научиться отдавать предпочтение изменению масштаба проекта. Нет ничего проще фиксирования календарного плана, ресурсов и качества проекта. Это не означает, что время от времени мы не можем фиксировать масштаб проекта и варьировать календарный план или ресурсы. Однако мы должны отдавать предпочтение изменению масштаба проекта таким образом, чтобы он соответствовал имеющимся ресурсам и календарному плану.

Альтернативные варианты

Чтобы понять, почему следует отдавать предпочтение изменению масштаба проекта по сравнению с другими вариантами, рассмотрим следующий пример. Допустим, мы входим в состав команды, которая только что завершила девятый месяц проекта, выполнение которого, согласно нашему календарному плану, должно продолжаться 12 месяцев. В этот момент каждому становится понятно, что к окончанию 12 месяцев данный проект в полном объеме реализовать невозможно, если он по-прежнему будет выполняться силами лишь этой команды. Какие варианты есть в нашем распоряжении?

Пожертвовать качеством

Возможно, нам следует кое в чем пожертвовать качеством: сократить тестирование, оставить неисправленными несколько ошибок и т.п. Снижение качества в явном виде как вариант рассматривается редко, тем не менее зачастую этот вариант используется «по умолчанию», когда становится ясно, что исполнители проекта не укладываются в сроки, предусмотренные календарным планом. Если снизить качество (возможно, изменив определение того, какие ошибки должны быть исправлены до отправки продукта заказчику, и, возможно, исключив стресс-тестирование), то можно завершить проект за оставшиеся три месяца. Проблема, однако, в том, что снижение качества — «близорукий» подход. Если такие решения будут приняты, они непременно «аукнутся» команде при следующем выпуске. Тогда команда, вероятно, попадет в такой же цейтнот, и, кроме того, ей придется найти способ выплаты «технического долга», который образовался, когда она пыталась уложиться в срок при выпуске предыдущей версии продукта.

Scrum-команды усвоили, что наилучший способ ускорения реализации проекта — это поддержание высокого качества системы на протяжении всего времени ее разработки. В 1979 году Филип Кросби (Philip Crosby) писал, что “нет ничего дешевле качества” и что “по-настоящему дорого обходятся нам лишь некачественные действия, т.е. действия, которые не позволяют нам выполнить работу надлежащим образом с первого раза” (1). Поэтому, если мы попытаемся уложиться в сроки, предусмотренные календарным планом, за счет снижения качества, то, скорее всего, мы лишь затормозим процесс разработки из-за необходимости переделывать некачественно выполненную работу и из-за неустойчивого функционирования системы (даже на сравнительно коротком отрезке времени).

Еще одной проблемой, связанной со снижением качества, является необходимость решить, насколько оно будет снижено и каких именно компонентов системы это снижение будет касаться. Вообще говоря, последствия снижения качества прогнозировать довольно трудно. И если Кросби прав, то попытки снизить качество могут привести лишь к затягиванию процесса разработки системы. Представьте, что вас попросили сократить продолжительность разработки системы с шести до пяти месяцев и что в силу тех или иных причин все участники разработки (в том числе и вы) пришли к выводу, что решить эту задачу лучше всего путем снижения качества. Как именно нужно снизить качество, чтобы сократить продолжительность разработки системы на один месяц? Точнее говоря, качеством каких компонентов системы можно в данном случае пожертвовать (например, путем сокращения тестирования) и в какой степени можно снизить это качество (т.е. от выполнения каких именно тестов можно отказаться)? От каких этапов проверки функционирования системы можно было бы отказаться в данном случае? Трудность получения ответов на эти вопросы показывает, насколько непредсказуемыми могут быть последствия снижения качества, когда мы пытаемся за счет этого сократить время разработки системы.

Наращивать ресурсы

А что если попытаться уложиться в календарный план за счет наращивания ресурсов? Многим кажется, что, включив в команду еще нескольких (или несколько десятков) разработчиков, можно уложиться в календарный план. К сожалению, все не так просто, как может показаться на первый взгляд. В книге *The Mythical Man-Month* (“Мифический человеко-месяц”) Фред Брукс (Fred Brooks) пишет, что “наращивание числа исполнителей на поздних стадиях выполнения проекта, связанного с разработкой программного обеспечения, приводит лишь к еще большему затягиванию выполнения проекта” (1995, 25). Если до завершения нашего 12-месячного проекта остается три месяца, то вполне возможно, что время, которое придется потратить на обучение новых членов команды, издержки, обусловленные дополнительными коммуникациями, и тому подобное сведут на нет выгоды от подключения к проекту дополнительных разработчиков, которые можно получить за столь короткий период. Если бы мы, осознав, что не укладываемся в запланированные сроки, подключили дополнительных разработчиков после первого месяца работы над 12-месячным проектом, то такое наращивание человеческих ресурсов имело бы больше смысла, поскольку у дополнительных разработчиков оставалось бы значительно больше времени для того, чтобы внести свой вклад в реализацию этого проекта.

Даже если бы нам пришлось обсуждать относительные достоинства наращивания человеческих ресурсов и определения момента, когда делать это будет уже слишком поздно, несомненно и не подлежит обсуждению одно: последствия наращивания человеческих ресурсов точно предсказать невозможно, а раз так, то такое наращивание таит в себе немалый риск.

Откорректировать календарный план

Таким образом, если невозможно гарантировать успех нашего гипотетического проекта за счет снижения качества или наращивания ресурсов, остается только одно: изменить масштаб проекта или его календарный план. Рассмотрим сначала корректирование календарного плана. С точки зрения команды разработчиков, корректирование календарного плана — замечательный вариант. Если наш проект невозмож но завершить за оставшихся три месяца, как предполагалось вначале, остается лишь прикинуть, сколько понадобится дополнительного времени, а затем объявить, что это и есть наш новый календарный план. Помимо трудности в правильном определении срока завершения проекта, никаких других проблем у разработчиков в этом случае не должно возникнуть.

К сожалению, корректирование календарного плана может привести к серьезным проблемам для вашего бизнеса. Составив первоначальный вариант календарного плана, вы взяли на себя определенные обязательства перед заказчиками или инвесторами. Планы рекламы (в том числе долгостоящая реклама на телевидении перед матчами за Суперкубок по бейсболу) составляются с учетом даты выпуска нового продукта. Возможно, ваша компания уже приняла на работу новых сотрудников, которым предстоит отвечать на возросшее число запросов, касающихся продажи или сопровождения нового продукта. Возможно, ваша компания уже запланировала обучение этих новых сотрудников. И так далее. Несмотря на внешнюю привлекательность корректирования календарного плана и его удобство с точки зрения команды разработчиков, такой вариант далеко не всегда можно осуществить на практике. Однако если такая возможность существует, то рассмотреть ее, безусловно, стоит.

Откорректировать масштаб проекта

А как насчет изменения масштаба выпуска? Да-да, “изменение масштаба” — это такой политкорректный синоним отказа от реализации части функциональных возможностей. Для начала разберемся, всегда ли так уж плох отказ от реализации части функциональных возможностей. Впервые планируя свой проект, мы подвели черту под некоторой совокупностью функциональных возможностей и сказали заказчику: “Вот это вы получите”. Допустим, мы подвели черту под сотней функциональных возможностей. Я вполне допускаю, что владелец продукта испытал большое разочарование, когда узнал, что команда не намерена реализовать 101-ю функциональную возможность, предусмотренную в журнале запросов на выполнение работ. От реализации этой функциональной возможности было решено отказаться еще до того, как мы приступили к выполнению проекта.

Теперь, когда мы пришли к выводу, что сможем реализовать лишь первые 95 из 100 первоначально запланированных функциональных возможностей, владелец продукта имеет еще больше оснований для разочарования. Однако это еще не конец света, особенно если принять во внимание, что команда реализует функциональные

возможности строго в порядке их приоритетности. Если пять функциональных возможностей, от реализации которых мы решили отказаться, имеют самый низкий приоритет среди ста первоначально запланированных функциональных возможностей, и если мы исходим, как это обычно и бывает, из того, что команда делает лучшее, на что она способна в конкретных обстоятельствах, то владелец продукта получит наилучший возможный продукт за отведенное для этого время и при данном составе команды.

Отказ от реализации части функциональных возможностей удручет? Несомненно! Было бы лучше, если бы мы могли точнее спрогнозировать, какой объем работы может быть выполнен к указанному сроку? Несомненно! Есть ли у нас основания на-деяться на абсолютную точность таких прогнозов? К сожалению, нет.

См. также Конкретные рекомендации по оценке объема функциональных возможностей, которые могут быть реализованы, и сроков их реализации, приводятся в следую-щем разделе, — “Оценки и обязательства — это не одно и то же”.

Итак, вернемся к нашему примеру, в котором мы пришли к выводу, что не успеваем реализовать все функциональные возможности за три месяца, оставшиеся до завершения нашего гипотетического проекта. Является ли сокращение масштаба разумной реакцией в данной ситуации? С точки зрения команды разработчиков, несомненно, является. Если команда поняла, что не успевает завершить всю запланированную работу, остается лишь определить, какую именно работу удастся завершить к указанному сроку, отказавшись от всего остального. Если команда придерживается гибкой методологии разработки (и, в частности, доводит систему до состояния потенциальной готовности к поставке заказчику к концу каждого спринта), то у нее не возникнет никаких проблем с отказом от реализации части функциональных возможностей.

См. также Подробнее о важности доведения системы до состояния потенциальной готовности к поставке заказчику к концу каждого спрингта рассказывается в главе 14, “Спринты”.

С точки зрения компании, сокращение масштаба проекта в любом случае нежелательно. Но какие варианты у нее есть? Мы пришли к выводу, что снижение качества во имя соблюдения сроков, предусмотренных календарным планом, является плохим выходом. Мы также пришли к выводу, что последствия наращивания человеческих ресурсов предсказать невозможно. В таком случае компании остается лишь изменить календарный план или сократить масштаб проекта. С учетом вероятных проблем, которые может вызвать корректирование календарного плана, предпочтительным вариантом зачастую оказывается сокращение масштаба проекта (в предположении, что функциональные возможности реализуются в порядке их приоритетности).

Важность контекста конкретного проекта

Правильный выбор компромиссов между сторонами “железного треугольни-ка” сводится к принятию правильных решений в контексте выполняемого проекта.

Я вовсе не настаиваю на том, что первое, с чего нужно начинать, — это сокращение масштаба проекта. Я вовсе не настаиваю на том, что к сокращению масштаба проекта всегда можно относиться, как к само собой разумеющемуся варианту. Я просто хочу, чтобы организации усвоили, что сокращение масштаба проекта зачастую оказывается более осуществимым вариантом, чем нам представлялось в прошлом, и что зачастую масштаб проекта оказывается именно той стороной “железного треугольника”, корректировать которую целесообразнее всего.

ВОЗРАЖЕНИЕ

“Этот продукт напоминает автомобиль: автомобиль ни на что не годится, если в нем есть двигатель, но нет тормозов. Мне нужно и то, и другое”.

Совершенно верно: любой автомобиль должен обладать неким обязательным набором функциональных возможностей. Я полностью согласен с тем, что во всех автомобилях, изготовленных с тех пор, когда их водил Фред Flintстоун (Fred Flintstone)¹, должны присутствовать и двигатель, и тормоза. Однако есть много других функциональных возможностей, даже в автомобиле, без которых, в принципе, можно обойтись: прозрачная крыша, кондиционер, сенсоры сцепления и т.п. Главное, чтобы функциональные возможности реализовались в порядке их приоритетности. Это означает, что программные эквиваленты двигателя и тормозов должны создаваться в первую очередь. Впоследствии, когда мы поймем, что не в состоянии реализовать все запланированные функциональные возможности в сроки, предусмотренные календарным планом, возможно, придется пожертвовать даже безусловно привлекательными функциональными возможностями, которые, тем не менее, не относятся к числу действительно фундаментальных для разрабатываемой системы. Если же проект окажется в ситуации, когда в сроки, предусмотренные календарным планом, можно уложиться, лишь отказавшись от реализации функциональных возможностей, по-настоящему важных для разрабатываемой системы (двигатель и тормоза и т.п.), то можно рассмотреть и другие альтернативы — в том числе отказ от дальнейшего выполнения проекта.

“Если продукт включает меньше функциональных возможностей, чем планировалось, его никто не купит”.

Здесь та же ситуация, что и с автомобилем. Реальная проблема в этом случае заключается в том, что соответствующий план создавался без надлежащего “запаса прочности”. Чаще всего это возражение приходится слышать, когда процесс планирования проекта включал создание журнала запросов на выполнение работ, определение наиболее ранней возможной даты завершения всех работ и оповещение об этой дате заказчиков или пользователей в качестве официальной даты завершения проекта. Если соответствующий продукт действительно нуждается в функциональных возможностях, которые не могут быть реализованы к указанной дате, то мы имеем дело с одним из тех случаев, когда дату завершения проекта нужно перенести на более отдаленный срок. Проблема в таком случае заключается главным образом в ненадлежащем планировании проекта, а не в чем-либо другом.

¹ Фред Flintстоун — главный персонаж анимированного ситкома The Flintstones, который демонстрировался телекомпанией ABC с 1960 по 1966 год. — Примеч. пер.

Оценки и обязательства — это не одно и то же

Фундаментальная и типичная проблема во многих организациях заключается в том, что оценки, выполняемые организацией, и обязательства, которые она принимает на себя, считаются одним и тем же. Команда разработчиков, какой бы методологии разработки (последовательной или гибкой) она ни придерживалась, прикидывает, что реализация желаемой совокупности функциональных возможностей — при наличии у команды определенных ресурсов — займет семь месяцев. Члены команды сообщают эту оценку своему руководителю, который передает ее вице-президенту. Тот, в свою очередь, информирует заказчика. В некоторых случаях в процессе таких передач информации от одних лиц к другим первоначальная оценка изменяется до неузнаваемости, в результате чего перед командой ставится нереальная цель.

Проблема в данном случае заключается не в том, что оценка командой продолжительности выполнения проекта (семь месяцев) оказалась правильной или неправильной. Проблема в том, что эта оценка превратилась в обязательства, которые приходится брать на себя команде. Оценка (“по нашему мнению, продолжительность выполнения данного проекта составит семь месяцев”) превратилась в обязательство (“мы обязуемся выполнить данный проект за семь месяцев”). Важны и оценка продолжительности выполнения проекта, и обязательство, принятое на себя командой, однако их следует рассматривать как разные виды деятельности.

Я должен забрать мою дочь с занятий плаванием, которые предстоят ей сегодня вечером. Я спросил ее, когда именно закончатся занятия (т.е. когда закончится ее тренировка в бассейне, после чего она еще должна будет принять душ и переодеться). Она ответила: “Я буду готова в 17:05”. Это была ее оценка. Если бы я попросил ее взять на себя какое-то твердое обязательство (“Ты должна ожидать меня на улице к такому-то времени, в противном случае я уезжаю домой, не дожидаясь тебя”), то она, наверное, указала бы другое время, например 17:25, чтобы исключить любые случайности (например, необходимость немного задержаться на тренировке, отставание наручных часов тренера на пять минут, очередь в душевую и т.п.). Чтобы определить время, к которому она обязуется быть на улице, моя дочь наверняка по-прежнему прибегла бы к оценке. Но вместо того чтобы сообщить мне свою оценку напрямую, она преобразовала бы ее в крайний срок, к которому она обязуется ждать меня на улице.

Какими данными нужно располагать

Хорошая организация должна уметь разделять получение оценки и принятие определенных обязательств. Сначала мы выполняем оценку, а затем, основываясь на степени своей уверенности в такой оценке, преобразуем ее в обязательство. Но без надежной первоначальной оценки любые обязательства команды будут лишены смысла. Чтобы получить надежную оценку, владелец продукта и команда должны располагать всеми необходимыми данными. Прежде всего они должны знать две важнейшие вещи.

- Объем работы, которую предстоит выполнить
- Ожидаемый темп выполнения этой работы командой

Чтобы оценить объем пользовательских историй, представленных в журнале запросов на выполнение работ, большинство команд использует либо баллы историй,

либо идеальные дни, как описано в книге *Agile Estimating and Planning* (Cohn, 2005). Темп реализации элементов журнала запросов на выполнение работ называется скоростью. Скорость — это просто сумма оценок (выраженных в баллах историй или идеальных днях) элементов журнала запросов на выполнение работ, реализованных в каждом спринте. Работая с этими значениями, владелец продукта может видеть, какой объем функциональных возможностей может быть реализован к тому или иному сроку. Давайте посмотрим, как владелец продукта может воспользоваться этой информацией для принятия обоснованных решений, позволяющих обеспечить оптимальный баланс между масштабом проекта и его календарным планом.

Пример

Рассмотрим табл. 15.1, в которой указаны фактические скорости одной из команд, с которыми мне приходилось работать. Первое, на что следует обратить внимание, — это изменчивость скорости: она может колебаться в значительных пределах от одного спринта к другому. Это объясняется невозможностью идеально оценить объем каждой из пользовательских историй, представленных в журнале запросов на выполнение работ: для некоторых из них оценка оказывается завышенной, для остальных — заниженной. Аналогично иногда в ходе одних спринтов команды вынуждены чаще прерывать свою работу, чем в других спринтах. Я приравниваю скорость команды к количеству баллов (очков), набранных спортивной командой в определенной серии игр. Моей любимой спортивной командой является баскетбольная команда Los Angeles Lakers. В последней серии из девяти игр они набирали 101, 94, 102, 102, 107, 93, 114, 117 и 97 очков. Сравните изменчивость количества набранных за матч очков с изменчивостью, представленной в табл. 15.1.

Таблица 15.1. Скорость команды изменяется от одного спрингта к другому

Номер спрингта	Скорость
1	34
2	41
3	27
4	45
5	35
6	38
7	40
8	39
9	40

Поскольку скорость команды изменяется (возможно, в достаточно больших пределах) от одного спрингта к другому, я не очень-то полагаюсь на отдельно взятые значения. В большей степени меня интересует вероятный диапазон будущих скоростей, или то, что статистик назвал бы *доверительным интервалом*. Пример доверительного интервала: по некоторым оценкам, глобальное потепление за период с 2000 по 2030 год составит от 0,1 до 0,3°C за каждые десять лет. В 2030 году, оглядываясь назад, мы сможем вычислить точное значение (например, 0,20°C), но при составлении прогноза мы

используем некий диапазон. Ученые, исследовав процесс глобального потепления, с 90-процентной уверенностью заявляют, что фактическая величина потепления (когда мы сможем вычислить ее в 2030 году) составит от 0,1 до 0,3°C за каждые десять лет.

То же самое я хотел бы знать и в отношении скорости команды. Я хотел бы, например, сказать, что в течение оставшихся пяти спринтов проекта моя команда с 90-процентной вероятностью продемонстрирует скорость в диапазоне от 18 до 26. К счастью, применить понятие доверительного интервала к скорости команды не так уж сложно.

Начните со сбора данных о скорости, постаравшись охватить как можно большее число прошлых спринтов. Чтобы вычислить 90-процентный доверительный интервал, вам понадобится по меньшей мере пять спринтов. Не учитывайте данные спринтов, которые неадекватно, на ваш взгляд, отражают возможную скорость команды в будущем. Если, например, восемь спринтов назад в команду было принято несколько новых сотрудников, то нужно собрать данные лишь за эти прошлые восемь спринтов. Но если на протяжении последних 13 спринтов численность команды колебалась от 5 до 7 человек и я рассчитываю, что численность команды продолжит колебаться в таких же пределах, то мне следует собрать данные за последние 13 спринтов. Анализируя конкретную ситуацию, старайтесь избегать игнорирования значений, которые могут помочь получить требующийся диапазон.

Получив прошлые величины скорости, отсортируйте их в порядке возрастания. Сортировка значений, приведенных в табл. 15.1, приводит к последовательности

27, 34, 35, 38, 39, 40, 40, 41, 45

Затем мы хотим воспользоваться этими отсортированными значениями скорости для нахождения диапазона, который, как мы можем утверждать с 90-процентной уверенностью, содержит скорость, которую команда станет демонстрировать в будущем. Воспользуемся табл. 15.2, в которой показано, какими двумя точками данных в нашей отсортированной последовательности скоростей нужно воспользоваться, чтобы определить этот 90-процентный доверительный интервал. Так, в табл. 15.1 было представлено 9 наблюдавшихся нами скоростей. Просматривая первый столбец табл. 15.2, мы видим два варианта, близких к 9: 8 и 11. Округляя в меньшую сторону, получаем 8. Напротив числа 8 мы видим во втором столбце значение 2. Это означает, что мы создаем доверительный интервал, воспользовавшись вторым наблюдением снизу и вторым наблюдением сверху в нашей отсортированной последовательности скоростей. Этими величинами являются 34 и 41. Следовательно, мы на 90% уверены в том, что средняя скорость этой команды окажется в диапазоне от 34 до 41.

Таблица 15.2. *n*-е нижнее и *n*-е верхнее наблюдения в отсортированной последовательности скоростей, которые можно использовать для нахождения 90-процентного доверительного интервала

Количество наблюдений скорости	<i>n</i> -е наблюдение скорости
5	1
8	2
11	3
13	4
16	5

Окончание табл. 15.2

Количество наблюдений скорости	<i>n</i> -е наблюдение скорости
18	6
21	7
23	8
26	9

Теперь можно воспользоваться этим доверительным интервалом для прогнозирования объема функциональных возможностей, которые могут быть реализованы к тому или иному сроку. Затем можно воспользоваться этим прогнозом, чтобы решить, какие обязательства можно принять в отношении масштаба проекта и календарного плана проекта. Допустим, что команде, речь о которой шла в табл. 15.1, до выпуска осталось пять спринтов. Чтобы увидеть, какой объем работы эта команда сможет выполнить за оставшееся время, мы можем умножить количество спринтов (5) на значения доверительного интервала (34 и 41). Затем отсчитаем в обратном порядке именно такое количество баллов историй в журнале запросов на выполнение работ и укажем диапазон функциональных возможностей, которые команда смогла бы реализовать. Этот прогноз представлен на рис. 15.4. Здесь же показана стрелка, указывающая среднюю величину скорости данной команды (39).

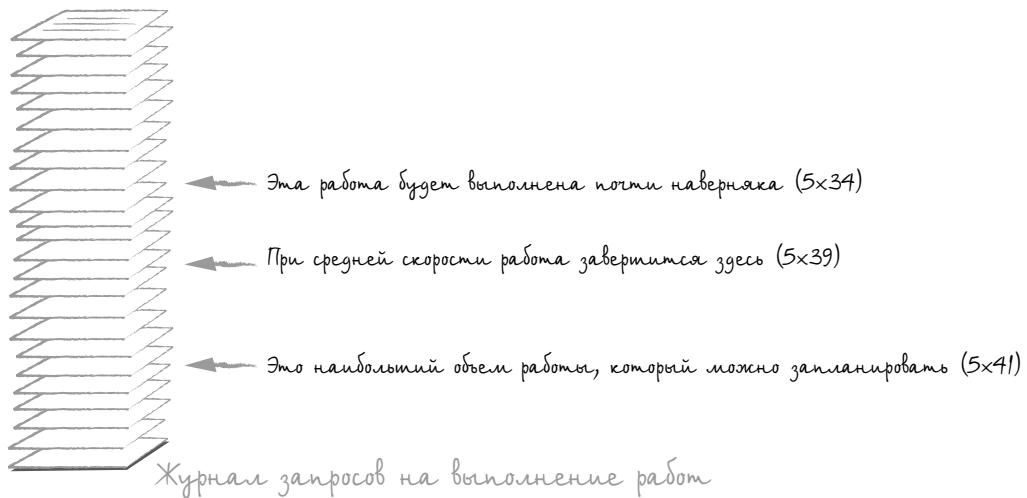


Рис. 15.4. Объем работы, которая может быть выполнена в течение пяти спринтов командой, скорость работы которой указана в табл. 15.1

Переход от оценки к обязательству

Три стрелки, которые вы видите на рис. 15.4, по-прежнему отражают лишь оценки. Для многих проектов эти оценки нужно преобразовать в обязательства. В идеальном случае я предпочел бы преобразовывать эти оценки в обязательства, сказав примерно следующее: “Мы полагаем, что в течение следующих пяти спринтов мы могли бы взять на себя обязательство реализовать от 170 [5×34] до 205 [5×41] баллов историй;

это означает, что мы реализуем истории от сих [верхняя стрелка] и до сих [нижняя стрелка] в журнале запросов на выполнение работ". Это было бы наиболее реалистичным и точным обязательством. Однако во многих случаях владельцам продукта и их командам приходится давать более точную оценку: "Мы обязуемся выполнить строго вот такой объем работы". Именно так часто приходится поступать организациям, выполняющим разработки по контракту с привлечением в качестве субподрядчиков сторонних организаций. В таких случаях организации приходится брать на себя обязательство реализовать вполне определенный объем функциональных возможностей ко вполне определенному сроку.

Когда организацию просят перейти от диапазона (например, 170–205 баллов историй) к точечной оценке, которая будет использоваться в качестве обязательства, всегда возникает соблазн ответить примерно так: "Хорошо, если вы хотите, чтобы мы взяли на себя твердое и вполне определенное обязательство, то мы обязуемся реализовать 170 баллов историй". Если указанный объем работ и есть то, что вы обязуетесь выполнить, то вы почти наверняка выполните его, но вместе с тем вы рискуете прямо сейчас навлечь на себя гнев человека, перед которым вы принимаете на себя это обязательство. Такое решение может рассматриваться как попытка, создавая для себя определенные проблемы в краткосрочной перспективе (ваш клиент, заказчик или начальник может рассердиться на вас прямо сейчас, если 170 баллов историй — это все, что вы обязуетесь выполнить), максимально снизить риск в долгосрочной перспективе (риск реализовать объем функциональных возможностей, меньший указанного, не так уж велик).

Альтернативный вариант — обязательство выполнить объем функциональных возможностей, соответствующий верхнему уровню доверительного интервала — вынуждает прибегнуть к противоположному компромиссу: риск в долгосрочной перспективе достаточно высок (вполне возможно, что вам не удастся реализовать столь большой объем функциональных возможностей), тогда как риск в краткосрочной перспективе очень мал (в данный момент все подумают, что вы — крутой парень, если берете на себя такие серьезные обязательства).

Итак, три стрелки на рис. 15.4 не говорят нам о том, какое именно обязательство нужно принимать. Они свидетельствуют лишь о наиболее вероятном диапазоне наших обязательств. Например, компания, выполняющая разработки по договорам и имеющая значительный резерв рабочей силы, вполне может принимать на себя обязательства вблизи нижней стрелки; цель этой компании, вероятнее всего, заключается в том, чтобы в максимальной степени задействовать неиспользуемую в данный момент рабочую силу, даже если компания берет на себя риск не справиться с исполнением этого обязательства. И наоборот, компания, выполняющая разработки по договорам и задействовавшая на выполнении этих договоров всю свою рабочую силу, приступая к реализации очередного проекта, вероятнее всего, возьмет на себя обязательство вблизи верхней стрелки.

ПРИМЕЧАНИЕ

В случае проекта с фиксированным масштабом аналогичный анализ может быть выполнен с целью определения вероятного диапазона количества спринтов, которые потребуются для реализации этого проекта. Для этого нужно просуммировать оценки требуемого диапазона и разделить полученный результат на оба значения, которые составляют ваш доверительный интервал.

Исторически сложившаяся скорость создает основу для обязательств

Когда я излагаю подобную информацию своим клиентам, они обычно жалуются, что с удовольствием выполняли бы такой анализ, но не располагают необходимыми для этого данными. У этой проблемы есть очень простое решение: получите нужные вам данные. Сделать это гораздо проще, чем может показаться на первый взгляд. Посмотрим, как это можно сделать, в двух типичных “проблематических” ситуациях: когда сформирована совершенно новая команда, члены которой никогда не работали вместе, и когда в ходе выполнения проекта изменяется (или изменится) численность команды.

Члены команды никогда раньше не работали вместе

Если члены команды никогда раньше не работали вместе, то такая команда не располагает данными об исторически сложившейся у нее скорости. Наилучшее решение в этом случае — предоставить членам команды возможность выполнить по меньшей мере один спринт и лишь после этого принять на себя какие-то обязательства. Разумеется, было бы гораздо лучше, если бы такой команде предоставили возможность принять на себя определенные обязательства лишь после выполнения двух-трех спринтов. Я понимаю, что принятие обязательств лишь после выполнения одного спринта возможно далеко не всегда, поэтому могу предложить альтернативный подход.

Составив журнал запросов на выполнение работ с учетом оценок, выраженных в баллах историй или идеальных днях, соберите свою команду. Предложите членам команды провести совещание, посвященное планированию спринта. Предложите им выбирать каждый раз из журнала запросов на выполнение работ одну пользовательскую историю, определять задачи, необходимые для реализации этой пользовательской истории, оценивать объем каждой такой задачи в часах, а затем принимать решение, могут ли они взять на себя обязательство реализовать эту пользовательскую историю в течение одного спринта. Они могут выбирать пользовательские истории в любой последовательности: мы не планируем реальный спринт, а лишь пытаемся уяснить, какой объем работы команда сможет выполнить в течение одного спринта. По сути, такой подход оказывается наиболее эффективным, если команда выбирает пользовательские истории из журнала запросов на выполнение работ в более или менее случайной последовательности. Если члены команды выберут историю, слишком сложную технически для реализации в одном из первых спринтов, — предложите им выдвигать любые предположения о состоянии разрабатываемой системы, которые кажутся им разумными. Когда члены команды становятся на определенной совокупности пользовательских историй и скажут, что не могут взять на себя обязательство выполнить что-либо сверх этого, просуммируйте оценки, выраженные в баллах историй или идеальных днях, которые были ранее назначены выбранным пользовательским историям. Таким образом мы получим одну оценку скорости данной команды. Если вам удастся подвигнуть команду на этот эксперимент, предложите ей спланировать таким же образом второй спринт и усредните полученные результаты. Это снизит влияние одной или двух некачественных оценок в первом спринте.

ВОЗРАЖЕНИЕ

“У нас еще нет команды. Мы сформируем ее, если придем к выводу, что сможем выполнить данный проект в приемлемые для нас сроки”.

Найдите в своей организации других сотрудников, которые обладают примерно такими же опытом и квалификацией, как и те, из кого вам хотелось бы сформировать свою команду. Если вам кажется, что со временем вам удастся сформировать свою команду из двух хороших программистов, одного великолепного программиста, двух надежных тестеров, одного превосходного проектировщика пользовательского интерфейса и одного конструктора баз данных, имеющего пятилетний опыт работы, то постарайтесь найти в своей организации похожую группу специалистов. Пригласите их на собрание и попросите их представить, что они являются членами команды, которой предстоит выполнить новый проект. Попросите их отнестись к предложенной вами задаче оценивания как можно серьезнее: кто-то из них, возможно, в конце концов окажется в этой команде.

По завершении собрания вам, возможно, понадобится откорректировать (в ту или иную сторону) полученную вами ранее оценку скорости работы исходя из того, каким, по вашему мнению, окажется окончательный состав команды в сравнении с этими “оценщиками”, а также из того, насколько серьезно, на ваш взгляд, эти “оценщики” отнеслись к выполнению предложенного вами задания.

Вычисление первоначальной скорости таким способом является лишь первым шагом; полученную оценку вам нужно преобразовать в диапазон, как в случае, когда на основе “исторических” данных мы вычисляли доверительный интервал. Одним из способов формирования диапазона на основании оценки скорости является включение вашей интуиции, руководствуясь которой, вы можете повысить и понизить эту оценку на 25% (или больше), если вам кажется, что команда отнеслась к выполнению предложенного вами задания не столь серьезно, как следовало бы.

Другим способом формирования диапазона на основании оценки скорости является изменение этой оценки на основе относительного среднеквадратического отклонения, вычисленного исходя из скоростей других команд. Относительное среднеквадратическое отклонение — это просто среднеквадратическое отклонение, выраженное в процентах. Возвращаясь к табл. 15.1, можно вычислить среднеквадратическое отклонение для указанных там величин скорости, которое оказывается равным 5,1. Средняя скорость для этих данных составила 37,6. Разделив 5,1 на 37,6 и округлив результат деления, получаем 14%. Это и есть относительное среднеквадратическое отклонение. Если вы располагаете для ряда команд данными, подобными тем, которые представлены в табл. 15.1, то можете вычислить относительное среднеквадратическое отклонение для каждой из этих команд, а затем взять среднее этих значений и применить его к полученной вами оценке скорости. Это даст вам вполне достоверную оценку скорости вашей новой команды в виде диапазона. Несмотря на то что такой подход приводит к вполне достоверным результатам, я хотел бы повторить, что было бы лучше воспользоваться “историческими” данными для вашей команды или выполнить один-два спринта и лишь после этого брать на себя какие-то обязательства. Я настаиваю на использовании такого альтернативного подхода лишь в случаях, когда невозможно ни то, ни другое.

Численный состав команды меняется довольно часто

Проблема иного рода возникает, когда численный состав команды меняется довольно часто или когда ожидается, что он будет меняться часто. Как и в предыдущем случае, мой первый ответ предельно прост: не нужно менять численный состав команды. Стабильность состава идет на пользу команде. Разумеется, на длительном промежутке времени состав команды будет меняться. Нужно лишь не усугублять проблему, переводя людей из одной команды в другую без крайней необходимости. Между тем многие организации злоупотребляют такими перемещениями.

Мой второй ответ снова заключается в том, что необходимо все время собирать данные. Это позволит вам быть готовым к любым изменениям в составе команды и предвидеть возможное влияние этих изменений. Попросите кого-либо в своей организации отслеживать процентные изменения скорости в течение нескольких первых спринтов после любого изменения численного состава команды. Эти изменения нужно отслеживать в течение пары спринтов после изменения численного состава команды, поскольку во время первого спрингта скорость почти всегда снижается, даже если число членов команды увеличилось. Это является результатом возросшего объема общения, необходимостью ввести в курс дела новых членов команды (этим обычно занимаются опытные члены команды, которые могли бы посвятить это время выполнению конкретных работ, запланированных на соответствующий спрингт) и т.п. Мой собственный опыт свидетельствует о том, что долгосрочное влияние изменения численного состава команды оказывается вплоть до третьего спрингта после этого изменения.

См. также Сбором общеорганизационного показателя такого типа вполне мог бы заниматься отдел управления проектами (Project Management Office — PMO). PMO обсуждается в главе 20, “Кадры, техническое обеспечение и отдел управления проектами”.

Я рекомендую вычислять изменение скорости не по отношению к последнему спрингту перед рассматриваемым изменением, а по сравнению со средним значением за пять предшествующих этому изменению спринтов. Можно, конечно, попытаться рассматривать и большее количество спринтов, предшествующих изменению численного состава команды, но зачастую это просто не представляется возможным. Помните, что вы пытаетесь решить проблему в среде, в которой численный состав команды меняется довольно часто. Поэтому, если мы пытаемся рассматривать восемь спринтов, предшествующих изменению численного состава команды, то может оказаться, что в течение этих восьми спринтов численный состав команды также претерпевал изменения.

В результате мы получаем данные наподобие приведенных в табл. 15.3. В этой таблице показано, что численный состав команды в первой строке увеличился с шести членов до семи, в результате чего скорость команды снизилась на 20% в течение первого спрингта и на 4% в течение второго спрингта, однако в течение третьего спрингта скорость команды повысилась на 12%. Численный состав команды в последней строке увеличился с семи членов до восьми, но в этом измененном составе команда успела выполнить лишь один спрингт. Оставшиеся столбцы последней строки будут заполнены в конце следующих двух спринтов.

Таблица 15.3. Данные, отражающие последствия изменения численности команды

Первоначальная численность команды, человек	Новая численность команды, человек	Спринт +1, %	Спринт +2, %	Спринт +3, %
6	7	-20	-4	+12
6	7	0	-6	+15
7	5	-12	-8	-8
8	6	-20	-20	-16
7	8	-15		

На обновление электронной таблицы такого рода может уходить в каждом спринте не более нескольких минут, даже если при этом приходится отслеживать десятки команд. Я старался максимально ее упростить, например я не отслеживаю, кто именно из специалистов — программист, тестер или кто-либо другой — был включен в состав команды или выведен из ее состава. Таким образом, вполне вероятно, что вы уже располагаете исходными данными, необходимыми для построения электронной таблицы, подобной табл. 15.3. Если каждая из команд отслеживает свою скорость и знает, каким был ее состав во время каждого спринта, то вы сможете воссоздать значительный объем таких “исторических” данных.

См. также Вы можете загрузить электронную таблицу для отслеживания данных этого типа с веб-сайта www.SucceedingWithAgile.com.

Данные такого рода можно использовать для получения ответов на широкий круг вопросов, например таких.

- Какой окажется скорость этой команды, если в ее состав включить еще двух специалистов?
- К какому сроку нам удалось бы завершить проект, если бы мы добавили в каждую команду по одному человеку?
- Если требуется завершить совокупность проектов к концу года, то сколько людей нам нужно добавить к нынешнему составу исполнителей этих проектов?
- Какими могли бы оказаться последствия отказа от новых сотрудников?
- Какими могли бы оказаться последствия сокращения состава исполнителей проекта на 15%?

Рассмотрим простой пример использования только данных, представленных в табл. 15.3. Допустим, вас попросили оценить, насколько больший объем работы может быть выполнен в течение следующих семи спринтов, если численный состав команды увеличить с шести до семи человек. Усреднив первые две строки в табл. 15.3,² вы приходите к выводу, что скорость снизится примерно на 10% в первом спринте, снизится примерно на 5% во втором спринте, но после этого (когда новый член ко-

² Для простоты в табл. 15.3 содержится только две строки для команд, численный состав которых увеличивается с шести до семи человек. Если бы речь шла о реальном проекте, то для принятия решений я предпочел бы располагать большим объемом данных, чем приведенный в книге.

манды будет полностью “ассимилирован”) повысится примерно на 13%. Эти значения представлены в табл. 15.4.

Таблица 15.4. Вычисление последствий изменения численности команды с шести до семи человек

Спринт	Изменение скорости, %
1	-10
2	-5
3–7	+13

Усредняя эти изменения скорости, выясняем, что ожидаемое изменение скорости на протяжении семи спринтов составит немногим более 7%. Теперь вы можете ответить своему начальству, что наращивание численного состава команды с шести до семи человек (т.е. речь идет о 17-процентном наращивании численного состава команды и, возможно, бюджета) позволит вам в течение запланированных семи спринтов нарастить объем функциональных возможностей примерно на 7%. Владелец продукта может воспользоваться этой информацией для оценки целесообразности повышения издержек.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Приступайте к сбору данных, которые могут понадобиться в вашей ситуации. Как минимум, создайте электронную таблицу, в которой будет регистрироваться скорость каждой команды в каждом спринте. Рассмотрите возможность включения данных об изменениях численного состава команды или других факторов, если такие изменения имеют место в вашей организации.
- ❑ После того как вы соберете достаточный объем данных об “исторической” скорости команд, приступайте к построению диаграммы, подобной той, которая изображена на рис. 15.4. Такой диаграммой может воспользоваться владелец продукта каждого проекта для принятия решений, касающихся выбора между изменением масштаба проекта и изменением календарного плана проекта.

Резюме

Умение составлять эффективные планы — весьма важное качество для любой Scrum-команды. В этой главе мы рассмотрели способы, с помощью которых команда, освоившая основы планирования спринтов и выпусков, может улучшить свои результаты путем

- постепенного уточнения планов;
- работы в постоянном и приемлемом для себя темпе;
- изменения масштаба проекта как предпочтительного метода исправления ситуации в случаях, когда невозможно выполнить весь объем запланированных работ в сроки, предусмотренные календарным планом;
- уяснения разницы между оценками и обязательствами.

Дополнительная литература

Cohn, Mike. 2005. *Agile estimating and planning*. Addison-Wesley Professional.

Это исчерпывающий источник информации об оценивании и планировании проектов, выполняемых с помощью гибкой методологии разработки. В этой книге рассказывается о достоинствах и недостатках баллов историй и идеальных дней — двух самых распространенных показателей, которые могут использоваться Scrum-командами для получения оценок. Автор знакомит читателей с популярным методом оценивания под названием *Planning Poker*. Подробно рассказывается также о приоритизации и планировании в разных ситуациях.

Moløkken-Østvold, Kjetil, and Magne Jørgensen, 2005. A comparison of software project overruns: Flexible versus sequential development methods. *IEEE Transactions on Software Engineering*, September, 754–766.

Эта статья была написана двумя уважаемыми исследователями из Simula Research Lab. В ней представлен углубленный обзор проектов, связанных с разработкой программного обеспечения. Вывод авторов этой статьи сводится к тому, что в проектах, выполняемых с помощью гибкой методологии разработки, используется меньший объем сверхурочных работ, чем в проектах, выполняемых с помощью традиционного метода последовательной разработки. Авторы указывают на ряд возможных причин такого положения дел, к числу которых относят более точные спецификации требований (журнал запросов на выполнение работ) и более эффективное взаимодействие исполнителей с заказчиком.

Глава 16

Качество

В молодости, когда я был начинающим программистом, я уволился из солидной и стабильной компании и перешел на работу в недавно сформированную компанию, где в то время трудилось лишь восемь человек. В моей прежней организации был специальный отдел тестирования и обеспечения качества, тогда как в новой компании я был лишь вторым программистом, а должность тестера в ней вообще не была предусмотрена. Поначалу такая ситуация меня шокировала: я сам отвечал за качество своего продукта! Я не мог рассчитывать на тестеров, которые проверяли бы мою работу или подстраховывали меня в моих робких попытках протестировать исходный код. А несколько позже наступило еще большее прозрение: в отсутствие тестера я мог выглядеть полным дураком в глазах наших заказчиков (хотя никто из них не был лично знаком со мной). Более того, я мог выглядеть полным дураком в глазах моего начальника, а это грозило мне куда большими неприятностями.

Я запаниковал. На мое счастье, та же беда постигла другого нашего программиста, который приступил к работе на две недели позже меня. Мы постарались взять себя в руки и не поддаваться панике. Вместе с ним мы создали замечательный комплект инструментов и методов для тестирования нашего приложения, связанного с компьютерной телефонией. Вспоминая спустя двадцать лет системы, разработанные нами в то время, я продолжаю настаивать на том, что они относятся к числу наиболее скрупулезно протестированных приложений, в разработке которых мне когда-либо приходилось участвовать. А все объясняется тем, что наша компания, испытывая острую нехватку денежных ресурсов, не могла позволить себе принять на работу тестера и вынуждала нас, программистов, отвечать за качество создаваемых нами продуктов.

К тому времени, когда мы все же смогли принять на работу тестеров, наша растущая команда разработчиков уже пришла к пониманию того, что за качество должна отвечать именно команда в целом. С тех пор я старался донести это понимание до каждой организации-разработчика, с которой мне приходилось работать. В этой главе я рассказываю о важности интеграции тестирования в процесс разработки, вместо того чтобы оставлять тестирование “на потом”. Кроме того, я знакомлю читателей с

пирамидой автоматизации тестирования и показываю, как меры, направленные на автоматизацию тестирования в большинстве компаний, не приносят желаемого результата, потому что эти компании упускают из виду добрую треть этой пирамиды. Наконец будет рассмотрена важность разработки на основе приемочного тестирования, а также важность выплаты “технического долга”.

Включайте тестирование в процесс разработки

Новые автомобили я покупаю нечасто, обычно каждые 10–12 лет. Поэтому, купив в 2003 году новый автомобиль, я был потрясен тем, как далеко шагнула техника со временем приобретения мною предыдущего автомобиля (это была “Хонда” выпуска 1993 года). Одним из достижений, которые произвели на меня наибольшее впечатление, был датчик, который автоматически обнаруживает пониженное давление в автомобильных шинах. Зачастую это нелегко определить на глаз, а проверка давления вручную — малоприятная работа, поэтому я делаю это неохотно и сравнительно редко. Постоянный контроль давления в автомобильных шинах является, на мой взгляд, выдающимся достижением.

За тот же период, в течение которого производители автомобилей придумали эффективный способ постоянного контроля давления в автомобильных шинах, разработчики программного обеспечения пришли к выводу, что постоянное тестирование их продукции также является великолепной идеей. Раньше мы полагали, что тестирование — это одноразовое действие, которое нужно выполнить по завершении написания кода. Цель тестирования заключалась в том, чтобы проверить код на отсутствие ошибок, которые могли быть внесены в него на предыдущих этапах разработки. Тестирование было чем-то вроде того, как, уезжая в отпуск, вы проверяете, выключена ли газовая плита, закрыты ли окна и заперта ли на замок наружная дверь. Разумеется, после того как вы обнаружите все ошибки, допущенные на предыдущих этапах процесса разработки (а такие ошибки практически неизбежны), тестирование можно рассматривать уже не как окончательную проверку, а как способ повышения качества продукта.

Некоторым командам понадобилось не так уж много времени, чтобы понять, что тестировать качество по окончании разработки не только неэффективно, но и недостаточно. Такие команды в большинстве случаев переходили к итеративному способу разработки. При этом они расчленяли продолжительную фазу тестирования, приходившуюся на окончание проекта, на множество небольших фаз тестирования, каждая из которых наступала после фазы анализа, проектирования и кодирования. Это был шаг вперед, но этого было недостаточно.

Перейдя к использованию Scrum, мы пошли в этом смысле еще дальше.

Для Scrum-команд тестирование является одним из центральных видов деятельности и частью процесса разработки, а не чем-то таким, что совершается уже после того, как разработчики завершат свою работу. Вместо того чтобы пытаться тестировать качество после того, как продукт уже создан, мы встраиваем качество в процесс разработки и в продукт по мере его разработки. Американский профессор и консультант У. Эдвардс Деминг (W. Edwards Deming) прославился своей работой в Японии, в течение которой он неустанно подчеркивал влияние качества на издержки и производительность. Он настаивал на том, что качество невозможно “добавить в продукт

впоследствии”. Он писал, что необходимо “устранить зависимость качества от массового контроля. В первую очередь, нужно усовершенствовать сам процесс и встроить качество в создаваемый продукт” (2000, 23).

Почему тестирование в конце не дает нужного результата

Есть много причин, обуславливающих неэффективность традиционного подхода, который состоит в перенесении тестирования на конец разработки.

- **Трудно повысить качество уже существующего продукта.** Мне всегда казалось, что легко только снизить качество продукта, а вот повысить его нелегко, и для этого требуются значительные затраты времени. Вспомните какой-нибудь случай из своего прошлого, когда вы работали над приложением, которое уже было поставлено заказчику. Допустим, вас попросили добавить новый набор функциональных возможностей и заодно улучшить качество уже существующего приложения. Несмотря на все ваши старания, вполне может оказаться, что пройдут месяцы или даже год, прежде чем качество удастся повысить настолько, что это будет замечено пользователями. Тем не менее именно этого мы пытаемся достичь, когда тестируем качество продукта в конце его разработки.
- **Ошибки остаются незамеченными.** Лишь после того, как продукт будет всесторонне протестирован, можно с уверенностью говорить о его работоспособности. До тех пор нельзя исключить, что вы будете вновь и вновь повторять одну и ту же ошибку, даже не осознавая этого. Приведу пример. Джейффи возглавлял разработку веб-сайта, который должен был обеспечивать гораздо больший трафик, чем планировалось вначале. У Джейффи были кое-какие соображения насчет того, что следует сделать, чтобы улучшить пропускную способность каждой страницы сайта, и ему удалось реализовать соответствующие изменения. Для этого ему потребовалось сначала написать в одном месте новый Java-код, а затем включить в каждую страницу одну строку, которая позволяла воспользоваться этим новым кодом, повышающим пропускную способность страниц сайта. Это была очень кропотливая работа, потребовавшая значительных затрат времени. На внесение изменений Джейффи понадобился почти весь двухнедельный спринт. После этого он протестировал систему и пришел к выводу, что выигрыш в пропускной способности оказался ничтожным. Ошибка Джейффи заключалась в том, что он не протестировал теоретический выигрыш в пропускной способности на нескольких первых модифицированных им страницах. Постоянное тестирование на протяжении всего периода разработки позволяет избежать неприятных неожиданностей в конце разработки.
- **Трудно оценить текущее состояние проекта.** Допустим, я прошу вас оценить для меня две вещи: во-первых, некую совокупность новых функциональных возможностей и, во-вторых, сколько времени займет тестирование и исправление ошибок в продукте, который разрабатывался на протяжении шести месяцев и в настоящее время готов к своему первому раунду тестирования. Большинство людей согласится с тем, что оценить новую работу гораздо легче, а точность такой оценки наверняка окажется выше. Периодическое (а еще лучше — постоянное) тестирование продукта — это его “зондирование”, которое позволяет понять, какой объем работы мы выполнили в действительности.

- **Невозможно получить информацию обратной связи.** Очевидное преимущество использования Scrum заключается в том, что команда может получать информацию обратной связи о результатах своей работы по крайней мере к концу каждого спрингта. Соответствующий продукт может выкладываться на серверы с ограниченным доступом, а избранные заказчики могут получать право загружать его на свои компьютеры. Если качество продукта удается обеспечить на достаточно высоком уровне лишь ближе к концу цикла выпуска и, соответственно, обеспечить доступ пользователей к продукту можно также лишь к этому времени, то команда утрачивает замечательные возможности для более раннего получения ценной информации обратной связи.
- **Повышается вероятность сокращения тестирования.** Из-за давления, которое испытывают на себе разработчики при приближении срока сдачи готового продукта заказчику, вероятность сокращения объема работы по тестированию, запланированной ближе к моменту завершения проекта, неизбежно повышается.

ВОЗРАЖЕНИЕ

“Для постоянного тестирования потребуется слишком много времени. Нужно быть реалистами и признать, что лучше всего выполнять тестирование после каждого пятого или шестого спрингта”.

Когда кажется, что лучше тестировать реже, это обычно говорит о том, что тестирование отнимает слишком много времени. Как правило, это бывает в случае приложений, по отношению к которым в прошлом применялось тестирование вручную и которые в настоящее время начинают разрабатываться по методологии Scrum. Если стоимость тестирования настолько высока, что его невозможно выполнять в каждом спрингте, нужно приложить все силы к радикальному снижению этой стоимости. В указанном выше случае этого можно достичь, создав автоматизированные тесты, которыми следует заменить тестирование вручную. Нехватка автоматизированных тестов является формой “технического долга”. В одном из последующих разделов этой главы рассказывается о том, как выплатить этот долг.

“Было бы разумнее, если бы тестеры работали с отставанием на один спрингт от программистов”.

Если тестеры будут работать с отставанием на один спрингт от программистов, то к кому они обратятся, когда у них возникнут вопросы? Будет ли это разумным с точки зрения программистов, занятых выполнением своей работы в данном спрингте? Смогут ли тестеры эффективно участвовать в спрингте, если остальные члены команды заняты обсуждением способов реализации очередной совокупности функциональных возможностей, в то время как тестеры занимаются тестированием уже реализованных функциональных возможностей? Подробнее о совместной работе всех членов команды во время спрингта читайте в главе 14, “Спрингты”.

Как выглядит встраивание качества в процесс разработки на практике

Команда, которая включила тестирование в свою повседневную работу, будет выглядеть и вести себя совсем не так, как команда, которая пытается тестировать

качество продукта в конце его разработки. Ниже перечислены некоторые из заслуживающих внимания особенности команды, которая встраивает качество в процесс разработки своего продукта.

- **Самым очевидным будет использование эффективных методов программирования.** Команда, поставившая перед собой задачу встраивать качество в процесс разработки своего продукта, будет делать все от нее зависящее для написания максимально качественного кода. В частности, она будет использовать парное программирование и тщательный контроль кода по крайней мере в отношении наиболее сложных частей разрабатываемой системы. Самое пристальное внимание команда уделят если не разработке программного обеспечения на основе составления тестов, то хотя бы автоматизированному тестированию исходного кода. Рефакторинг осуществляется постоянно и по мере возникновения необходимости в нем, а не крупными, заметными рывками. Код будет непрерывно интегрироваться, а просчеты в сборке будут устраниться почти так же оперативно, как и любая серьезная ошибка, обнаруженная заказчиком. Вы обратите внимание и на то, что у кода будет не индивидуальный, а коллективный владелец, т.е. команда. Это необходимо для того, чтобы каждый, кто увидел возможность улучшить качество продукта тем или иным способом, мог сделать это, не ссылаясь на то, что об улучшении качества должен заботиться кто-то другой.

См. также Разработка программного обеспечения на основе составления тестов, парное программирование, рефакторинг и непрерывная интеграция обсуждались в главе 9, “Технические приемы”.

- **Передача работы из рук в руки между программистами и тестерами будет так мало (если они вообще будут), что ими можно пренебречь.** В главе 11, “Организация коллективного труда”, рассказывалось о том, как постоянное выполнение всего понемногу (проектирования, кодирования, тестирования и т.п.) помогает совместной работе всех членов команды. Работая таким способом, программист и тестер обсуждают, какая функциональная возможность (или частичная возможность) будет добавлена в продукт следующей. Затем тестер создает автоматизированные тесты, а программист — программирует. Когда та и другая работа выполнены, их результаты объединяются. Хотя это по-прежнему можно интерпретировать как передачу работы из рук в руки между программистом и тестером (и такая интерпретация будет, вообще говоря, правильной), в данном случае соответствующий цикл окажется настолько коротким, что на эти передачи можно не обращать внимания.
- **Объем тестирования в первый день и в последний день спринта один и тот же.** Команда, которая встраивает качество в процесс разработки своего продукта, избегает работы миниатюрными “водопадами”, описанными в главе 14, “Спринты”. В рамках одного спринта вы не обнаружите отчетливо выраженных, обособленных фаз анализа, проектирования, кодирования или тестирования. Тестеры (а также программисты и прочие специалисты) трудятся в первый день спринта с таким же напряжением, как и в последний день. Типы работ в первый и в последний день спринта могут быть разными. Например, в первый

день спринта тестеры могут специфицировать контрольные примеры и готовить тестовые данные, а в последний день спринта они могут выполнять автоматизированные тесты, но с первого и до последнего дня спринта им будет чем заняться.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ В течение очередного спринта фиксируйте каждый день количество выявленных ошибок. Отслеживайте все эти ошибки — те, которые входят в систему дефектов; те, которые превращаются в новые элементы вашего журнала запросов на выполнение работ; те, которые добавляются в журнал спринта; и даже ситуации, когда тестер просто сообщает программисту об ошибке и эта ошибка тотчас же регистрируется. Если тестирование встроено в процесс разработки, количество ошибок, обнаруженных за день, должно быть достаточно стабильным на протяжении всего спринта.
- ❑ Следующую ретроспективу посвятите обсуждению способов повышения качества.

Автоматизируйте на разных уровнях

Еще до появления гибких методологий разработки, подобных Scrum, мы знали, что тесты нужно автоматизировать. Знали, но не делали этого. Написание автоматизированных тестов считалось весьма дорогостоящим делом. После программирования той или иной функциональной возможности для написания таких тестов нередко требовалось месяцы (а в некоторых случаях — и годы). Одной из причин, по которым командам казалось трудным писать такие тесты быстрее, было то, что они автоматизировались не на том уровне, на каком следует. Эффективная стратегия автоматизации тестов предполагает автоматизацию тестов на трех разных уровнях, как показано на рис. 16.1. На этом рисунке представлена так называемая *пирамида автоматизации тестов*.

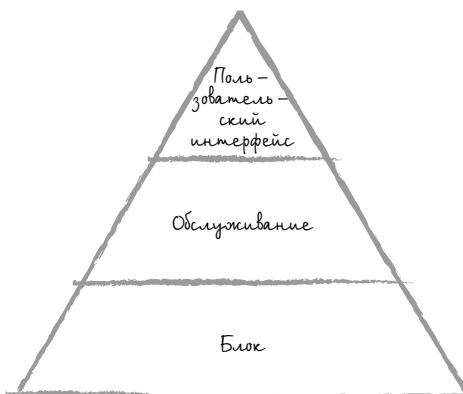


Рис. 16.1. Пирамида автоматизации тестов

В основании пирамиды автоматизации тестов находится блочное тестирование (unit testing). Оно должно быть фундаментом надежной стратегии автоматизации тестов и как таковое представляет самый крупный строительный элемент пирамиды.

Автоматизированные тесты исходного кода — чрезвычайно эффективный инструмент, поскольку они обеспечивают программиста конкретными данными, например “в исходный код вкрадась ошибка; она находится в строке 47”. Программисты понимают, что в действительности ошибка может находиться в строке 51 или 42, но если ее местоположение определит какой-либо автоматизированный тест исходного кода, то это будет гораздо лучше, чем услышать от тестера примерно следующее: “В написанной вами программе поиска записей в базе данных содержится ошибка”, поскольку это будет означать, что ошибка может содержаться в любой из примерно 1000 строк кода. К тому же, поскольку блочные тесты обычно пишутся на том же языке, что и соответствующая система, программисты практически не испытывают проблем при их написании.

Давайте отвлечемся от средней части пирамиды автоматизации тестов и перейдем сразу к ее вершине — уровню пользовательского интерфейса. Автоматизированное тестирование пользовательского интерфейса помещено на вершину пирамиды автоматизации тестов, поскольку мы хотим максимально сократить для себя объем соответствующей работы. Такое желание объясняется тем, что тесты пользовательского интерфейса зачастую имеют следующие отрицательные характеристики.

- **Хрупкость.** Даже незначительное изменение пользовательского интерфейса способно привести в негодность многие тесты. Если в ходе реализации проекта это повторяется многократно, команды просто прекращают вносить поправки в тесты каждый раз, когда в пользовательский интерфейс вносятся очередные изменения.
- **Сложность написания.** Метод быстрой записи тестов пользовательского интерфейса по принципу “записать и воспроизвести” может сработать, однако тесты, записанные подобным образом, как правило, оказываются наиболее хрупкими. Написание хорошего теста пользовательского интерфейса, в который не придется каждый раз вносить все новые и новые поправки, требует значительных затрат времени.
- **Затраты времени.** Выполнение тестов пользовательского интерфейса зачастую связано с значительными затратами времени. Мне встречалось немало команд, которые располагали впечатляющими наборами автоматизированных тестов пользовательского интерфейса, работавшими так долго, что их невозможно было выполнять даже каждый вечер, не то что несколько раз в день.

Допустим, мы хотим протестировать очень простой калькулятор, который дает пользователю возможность ввести два целых числа, щелкнуть на кнопке умножения или деления, а затем увидеть на экране результат этой операции. Чтобы протестировать такой калькулятор посредством пользовательского интерфейса, нужно написать ряд тестов для приведения в действие пользовательского интерфейса, ввести соответствующие значения в предназначенные для этого поля, нажать кнопки, а затем сравнить ожидаемое и фактическое значения. Тестирование такого рода, несомненно, даст нужный результат, но приведет к упоминавшимся выше проблемам хрупкости и дороговизны.

Кроме того, тестирование приложения таким способом в некотором смысле избыточно: задумайтесь над тем, сколько раз пользовательский интерфейс будет тестироваться с помощью набора тестов, подобных этому. Каждый контрольный пример

инициирует код, который связывает кнопку умножения или деления с встроенным в данное приложение кодом, выполняющим соответствующую математическую операцию. Кроме того, каждый такой контрольный пример тестирует код, который отображает результаты, и т.д. Такое тестирование посредством пользовательского интерфейса обходится достаточно дорого и должно быть минимизировано. Несмотря на наличие множества контрольных примеров, которые необходимо выполнить, не все они должны выполняться посредством пользовательского интерфейса.

Именно на этот случай в пирамиде автоматизации тестов предусмотрен *уровень обслуживания* (сервисный уровень).

Несмотря на то что я обозначил средний уровень пирамиды автоматизации тестов как уровень обслуживания, я не ограничиваю вас использованием лишь сервисно-ориентированной архитектуры. Все приложения состоят из разных сервисов. В том смысле, в каком я использую понятие уровня обслуживания, обслуживание — это то, что приложение делает в ответ на некоторый ввод данных (или совокупность таких вводов). В нашем примере с калькулятором предполагаются два вида обслуживания: умножение и деление.

См. также В главе 13, “Журнал запросов на выполнение работ”, тестирование сервисного уровня описывалось также как метод специфицирования поведения системы посредством примеров.

Тестирование уровня обслуживания — это тестирование обслуживания (сервисов) того или иного приложения отдельно от его пользовательского интерфейса. Поэтому вместо десятка контрольных примеров на умножение посредством пользовательского интерфейса калькулятора мы выполняем тесты на сервисном уровне. Чтобы увидеть, как это делается, допустим, что мы создаем электронную таблицу, подобную табл. 16.1, каждая строка которой представляет один контрольный пример. В первых двух столбцах представлены числа, которые нужно перемножить, в третьем столбце представлен ожидаемый результат, а в четвертом — примечания, которые не будут использоваться в ходе тестов, но сделают их более удобочитаемыми и понятными.

Таблица 16.1. Таблица с подмножеством сервисных тестов умножения

Множитель	Множимое	Результат	Примечания
5	1	5	Умножить на 1
5	2	10	
2	5	10	По сравнению с предыдущим тестом множитель и множимое поменялись местами
5	5	25	Умножить число само на себя
1	1	1	
5	0	0	Умножить на 0

Еще нужна простая программа, которая может считывать строки этой электронной таблицы, передавать содержащиеся в столбцах данные соответствующему сервису вашего приложения и проверять правильность полученных результатов. Несмотря

на нарочитую простоту этого примера, в котором результатом является простое вычисление, таким результатом может быть практически все что угодно: данные, обновляемые в базе данных; сообщение электронной почты, отправленное конкретному получателю; деньги, переведенные с одного банковского счета на другой; и т.д.

ПРИМЕЧАНИЕ

Несмотря на то что написание инструмента, предназначенного для чтения электронной таблицы и передачи данных соответствующим сервисам в вашем приложении не должно представлять особой проблемы для программистов вашей команды, существуют готовые инструменты, которые превосходно справляются с этой задачей. Наиболее популярным среди них является FitNesse, представленный на сайте www.fitnesse.org.

Другие роли тестов пользовательского интерфейса

Но не нужно ли нам выполнить тестирование пользовательского интерфейса? Несомненно, нужно, но в гораздо меньшем объеме, чем любой другой тип тестирования. В нашем примере с калькулятором нет необходимости выполнять все тесты умножения через пользовательский интерфейс; вместо этого большинство тестов (например, граничные испытания) мы выполняем на сервисном уровне, обращаясь непосредственно к методам (сервисам) умножения и деления для подтверждения того, что соответствующая математическая операция выполняется правильно. На уровне пользовательского интерфейса остается провести тестирование, призванное подтвердить, что указанные сервисы надлежащим образом связаны с соответствующими кнопками и что результаты выполнения математической операции правильно отображаются в поле результата. Для этого необходима значительно меньшая совокупность тестов, которые должны выполняться на уровне пользовательского интерфейса.

Ошибка многих организаций в их многолетней деятельности по автоматизации тестов в том и заключалась, что они игнорировали весь средний уровень сервисного тестирования. Несмотря на всю свою привлекательность, автоматизированное тестирование исходного кода способно охватить лишь какую-то часть тестовых потребностей каждого конкретного приложения. При отсутствии тестирования сервисного уровня, призванного заполнить прореху между блочным тестированием и тестированием пользовательского интерфейса, все остальное тестирование сводится к его выполнению посредством пользовательского интерфейса. В результате мы вынуждены иметь дело с хрупкими тестами, написание и выполнение которых к тому же обходится слишком дорого.

Роль тестирования вручную

Автоматизировать все тесты для всех сред не представляется возможным. К тому же автоматизация некоторых тестов обходится непозволительно дорого. Многие тесты, которые мы не можем автоматизировать (или предпочитаем не автоматизировать), связаны с аппаратными средствами или с интеграцией с внешними системами. Однажды я консультировал компанию, выпускающую фотокопировальную технику. В этой компании использовался ряд тестов, которые перед выполнением требовали вмешательства специалиста. Например, перед выполнением теста нужно было

убедиться, что в лоток для бумаги помещено пять листов бумаги. Разумеется, автоматизировать такой контроль было бы гораздо сложнее, чем поручить выполнение этой операции человеку.

Вообще говоря, тестирование вручную следует рассматривать главным образом как способ проведения исследовательского, или экспериментального, тестирования. Этот тип тестирования предполагает выполнение быстрого цикла, включающего такие этапы, как планирование теста, проектирование теста и выполнение теста. Экспериментальное тестирование должно осуществляться короткими циклами, обеспечивающими получение информации обратной связи. Это достигается посредством выполнения указанных выше этапов способом, аналогичным короткому циклу “тестирование–кодирование–рефакторинг”, который используется при разработке программного обеспечения на основе составления тестов.

В ходе экспериментального тестирования можно не только выявлять ошибки, но и находить пропущенные контрольные примеры, которые можно затем добавить на подходящем уровне пирамиды автоматизации тестов. Экспериментальное тестирование позволяет также выявить идеи, которые ускользнули из пользовательской истории в том ее виде, как она понималась вначале. Оно может помочь команде выявить идеи, которые понапалу казались плодотворными, но представляются достаточно сомнительными сейчас, когда соответствующая функциональная возможность уже разработана. Подобные ситуации обычно приводят к добавлению новых элементов в журнал запросов на выполнение работ.

Автоматизируйте во время спрингта

Автоматизация при выполнении Scrum-проектов не является чем-то необязательным. Чтобы команда эффективно выполняла свои спринты (и, следовательно, быстро обеспечивала ценность своему заказчику), она должна широко практиковать автоматизацию тестов. Автоматизированные тесты позволяют, не прибегая к значительным затратам, убедиться в том, что все, что работало правильно раньше, работает правильно и сейчас. Кроме того, постоянно расширяющаяся совокупность автоматизированных тестов позволяет лучше уяснить состояние разрабатываемого продукта (и процесса разработки). Если автоматизированный набор тестов не выполнялся успешно на протяжении двух недель, это должно серьезно настороживать. С другой стороны, если автоматизированный набор тестов расширяется буквально каждый день и выполнялся без единой ошибки каждый вечер на протяжении данного спрингта, то можно утверждать, что команда пребывает в хорошей форме.

Scrum-команды подходят к вопросу автоматизации тестирования не так, как команды, применяющие процесс последовательной разработки. Наличие высокоавтоматизированного набора тестов для Scrum-команд считается обязательным, тогда как для традиционных команд это скорее роскошь. Одной из причин отказа традиционных команд усматривать в автоматизации тестирования какую-то существенную пользу является то, что они не желают заниматься автоматизацией тестирования на как можно более ранних стадиях выполнения проекта. Тесты зачастую автоматизируют спустя месяцы после написания первоначального варианта кода. Команды, лишь приступающие к использованию Scrum, зачастую допускают одну и ту же ошибку: в ходе одного спрингта пишут код, а в ходе одного из последующих спрингтов автоматизируют тесты этого кода. Если тесты автоматизируются намного позже написания

соответствующего кода, ценность такой автоматизации существенно снижается. Код изменяется чаще всего на стадии его активной разработки, поэтому автоматизированные тесты приносят наибольшую пользу на этой же стадии.

См. также Если вам кажется невозможным кодировать, затем тестировать, а затем автоматизировать, и все это в рамках одного спрингта, вернитесь к главе 11, “Организация коллективного труда”.

На рис. 16.2 проиллюстрирована ценность ранней автоматизации тестирования. Стоимость автоматизации напоминает хорошо известную S-образную кривую: стоимость не повышается в течение пары спрингтов, после чего она существенно повышается, а затем кривая вновь переходит в пологий участок. Каждый, кто хоть однажды пытался применить автоматизированные тесты к какому-либо существующему приложению, знает, что иногда это сделать труднее, чем при добавлении тестов, когда их дизайн мог оказаться влияние на проектирование разрабатываемого продукта. При добавлении тестов на более поздних стадиях разработки зачастую приходится чрезмерно полагаться на верхний уровень пирамиды автоматизации тестирования; добавление автоматизированных блочных тестов и тестов сервисного уровня оказывается слишком трудным делом до тех пор, пока не будет выполнен значительный рефакторинг соответствующего приложения.

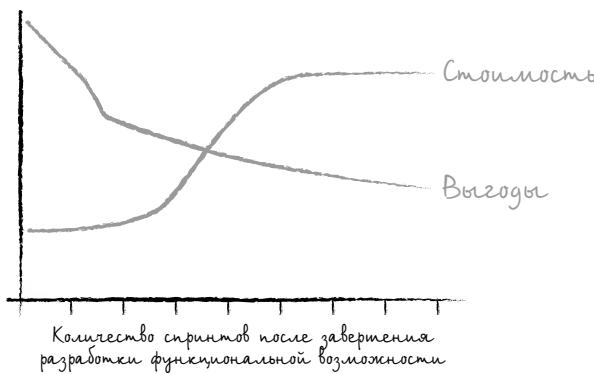


Рис. 16.2. Издержки и выгоды отсроченной автоматизации тестов

Несмотря на то что начальная пологость кривой стоимости может подтолкнуть вас к отсрочиванию автоматизации на один-два спрингта, делать это не следует. Достаточно стремительное снижение выгод автоматизации должно заставить вас заняться автоматизацией как можно раньше. Со временем выгоды автоматизации снижаются, поскольку вероятность и частота изменений, влияющих на эту область приложения, также снижаются. В конечном счете продукт может оказаться настолько стабильным, а оставшийся срок его службы столь коротким, что стоимость автоматизации перевесит выгоды от нее. Именно такой аргумент выдвигают многие традиционные команды, а также команды, полностью отказавшиеся от автоматизации.

Как бы то ни было, из рис. 16.2 должно быть ясно, что максимальная выгода может быть достигнута путем автоматизации тестов в том же спрингте, когда новая

функциональная возможность добавляется в систему. Это гарантирует наибольшую ценность, которая достигается с наименьшими затратами.

Выгоды от автоматизации тестов

Чтобы увидеть, насколько существенными могут быть выгоды, получаемые в результате тестирования по всем уровням пирамиды автоматизации тестирования, рассмотрим случай Salesforce.com. Компания Salesforce.com предлагает программное обеспечение в виде службы для управления работой с клиентами (Customer Relationship Management — CRM). Спустя девять месяцев после перехода к использованию Scrum Salesforce.com удалось добиться следующих сокращений.

- На 65% (до 15 человек) сокращена численность персонала, занятого в девяти вариантах применения данного приложения.
- Сокращено количество времени, затрачиваемого на финальные проверки работоспособности: два-три часа ручного тестирования превратились в десять минут автоматизированного тестирования.
- Сокращено количество времени, затрачиваемого на тестирование работоспособности после выпуска: три-четыре часа ручного тестирования превратились в 45 минут, которые требуются для прогона более чем 200 автоматизированных тестов.
- Почти на 80% (примерно до пяти человек) сокращено количество сотрудников, занятых в разработке исправлений.
- Сэкономлено свыше 300 человеко-часов на каждой основной версии и на сотни человеко-часов больше на всех версиях исправлений.

Такие результаты — вполне обычное явление для любой организации, которая всерьез занимается автоматизированным тестированием. Эти результаты можно использовать как исходные цели вашей собственной деятельности, направленной на автоматизацию тестов.

ВОЗРАЖЕНИЕ

“Пока программисты занимаются реализацией той или иной функциональной возможности, ситуация меняется слишком быстро, чтобы тестеры могли автоматизировать тесты”.

Чтобы ответить на это возражение, рассмотрим три типа автоматизации тестов, которые были представлены в пирамиде автоматизации тестирования и должны встречаться на практике. Очевидно, что блочные тесты должны создаваться во время спринта, возможно, с применением методики разработки на основе тестов. Опыт показывает, что программист лишь в редких случаях возвращаетесь впоследствии к работоспособному коду и добавляет блочные тесты. Обычно именно наша прошлая история с автоматизированным тестированием в стиле “записать и воспроизвести” заставляет нас думать, что к автоматизации можно приступить лишь после завершения реализации соответствующей функциональной возможности. Понятно, что тестер не может завершить выполнение автоматизированных тестов сервисного уровня и пользовательского интерфейса до тех пор, пока программисты не завершат кодирование. Но это не означает, что их разработку нельзя начать одновременно с кодированием. Как именно это делается, мы рассмотрим в следующем разделе.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- В ходе вашей следующей ретроспективы обсудите три уровня пирамиды автоматизации тестирования. Где именно тестирование происходит сейчас? Какие типы ваших текущих тестов можно было бы выполнить лучше при использовании иного типа автоматизации? Попытайтесь указать два-три способа, с помощью которых можно было бы приступить к автоматизации тестирования уже в следующем спринте.

Выполните разработку на основе приемочных тестов

Scrum-команды научились выравнивать поток работы в ходе спринта, выполняя разработку на основе приемочных тестов (Acceptance Test-driven Development — ATDD). В случае использования ATDD работа выполняется в ответ на приемочные тесты. Приемочные тесты играют роль фиксации решений, принятых в отношении реализации той или иной функциональной возможности. Как таковые они пишутся на протяжении спринта по ходу принятия таких решений.

ATDD можно представлять себе как аналог разработки программного обеспечения на основе составления тестов (Test-driven Development — TDD), речь о которой шла в главе 9, “Технические приемы”. Лasse Коскела (Lasse Koskela), автор книги *Test Driven: Practical TDD and Acceptance TDD for Java Developers*, отображает взаимосвязь между ATDD и TDD в таком виде, в каком она представлена на рис. 16.3. Я подкорректировал первоначальный вариант диаграммы, предложенный Коскелой, чтобы показать в этом цикле лишь роль условий удовлетворенности.

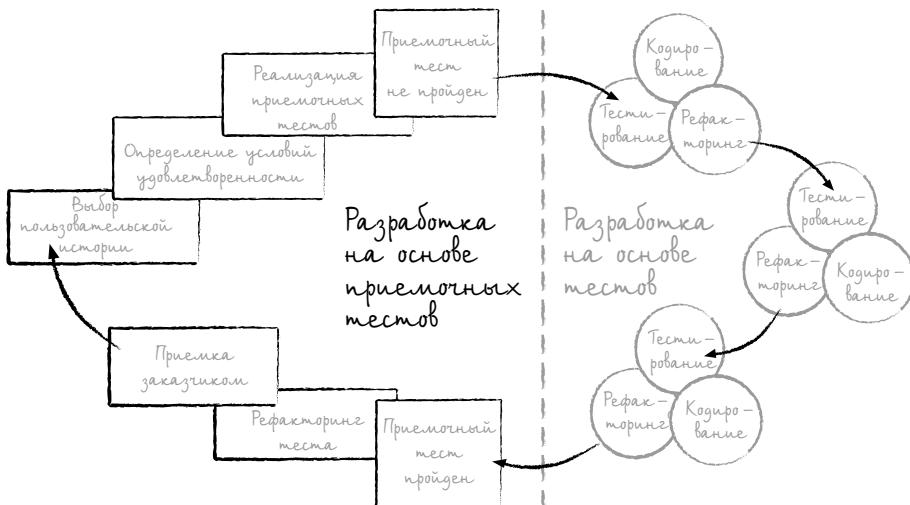


Рис. 16.3. Взаимосвязь между разработкой на основе приемочных тестов и разработкой на основе тестов (составлено на основе диаграммы, предложенной Коскелой, 2007)

См. также Условия удовлетворенности описаны в главе 13, "Журнал запросов на выполнение работ".

Как было сказано в главе 13, "Журнал запросов на выполнение работ", условия удовлетворенности призваны отражать высокоуровневые ожидания владельца продукта в отношении той или иной пользовательской истории. Как таковые они обычно располагаются на один уровень выше конкретных вариантов приемочных тестов, что делает их идеальным средством для стимулирования самого процесса ATDD.

Использование условий удовлетворенности в цикле, показанном на рис. 16.3, освобождает владельца продукта от необходимости постоянно участвовать в цикле ATDD. Владелец продукта может сформулировать условия удовлетворенности для какой-либо пользовательской истории либо в начале спринта, либо когда команда будет готова приступить к работе над этой историей. В той мере, в какой определяемые впоследствии приемочные тесты будут соответствовать ожиданиям владельца продукта, эти приемочные тесты могут составляться тестерами, аналитиками или, возможно, другими членами команды без вмешательства со стороны владельца продукта.

В идеальном случае владелец продукта должен появляться на совещаниях, посвященных планированию спринта, с последними сформулированными условиями удовлетворенности для каждой высокоприоритетной пользовательской истории, представленной в журнале запросов на выполнение работ. Владелец продукта не будет испытывать серьезных проблем с ответами на вопросы, которые обычно задаются на совещаниях, посвященных планированию спринта. Так, выполнение ATDD и заблаговременное формулирование условий удовлетворенности помогают сократить время на проведение совещаний, посвященных планированию спринта.

К сожалению, суровая действительность иногда нарушает наши планы, и владельцы продукта далеко не всегда приходят на совещания, посвященные планированию спринта, с заблаговременно сформулированными условиями удовлетворенности. Например, возникновение кризисной ситуации ближе к концу одного спринта может помешать владельцу продукта заблаговременно подготовиться к следующему спринту. Или в ходе совещания, посвященного планированию спринта, команда может потребовать при реализации той или иной пользовательской истории продвинуться несколько дальше по журналу запросов на выполнение работ, тогда как владелец продукта к этому моменту еще не сформулировал соответствующие условия удовлетворенности (поскольку их следует формулировать как можно ближе к началу реализации).

Подобные ситуации не препятствуют выполнению ATDD. Если условия удовлетворенности еще не сформулированы, у владельца продукта и команды есть два варианта. Во-первых, условия удовлетворенности для тех элементов журнала запросов на выполнение работ, для которых такие условия еще не сформулированы, можно выработать в ходе совещания, посвященного планированию спринта. Во-вторых, условия удовлетворенности можно сформулировать в качестве одного из первых действий в новом спринте. Приемлем любой из этих двух подходов, но первый предпочтительнее в большинстве случаев, когда позволяет время.

Наиболее подходящий уровень детализации

На первый взгляд может показаться, что формулирование условий удовлетворенности непосредственно перед началом каждого совещания, посвященного планированию спрингта, требует достаточно большого объема работы. При этом, однако, не следует забывать, что условия удовлетворенности представляют элементы верхнего уровня, которые должны соблюдаться для пользовательской истории, чтобы ее можно было считать выполненной к концу данного спрингта. Цель заключается в том, чтобы обсудить приемочные тесты верхнего уровня, ориентируясь на которые, разработчики могут судить об ожиданиях владельца продукта, а не в том, чтобы определить каждый мельчайший контрольный пример, который может со временем понадобиться.

Например, одна из команд, с которой мне пришлось однажды работать, занималась созданием веб-сайта, предназначенного для использования трейдерами, специализирующимися на купле/продаже акций. Соответствующий продукт предусматривал множество разных способов отображения данных, характеризующих акции и их цены, для трейдеров. Одна из диаграмм, которая называлась *treemap* (представление в виде дерева), отображала компании в виде маленьких прямоугольников, помещенных в более крупный прямоугольник. Размер каждого из таких маленьких прямоугольников отражал совокупную рыночную стоимость акций соответствующей компании. Если рыночная стоимость какой-либо компании превышала в два раза рыночную стоимость какой-то другой компании, то величина представляющего ее прямоугольника также была в два раза больше (соответствующий пример показан на рис. 16.4). Пользователь выбирает, хочет ли он видеть весь фондовый рынок, какой-либо его сегмент (например, компании, занимающиеся разработкой программного обеспечения) или другие сравнительные совокупности.

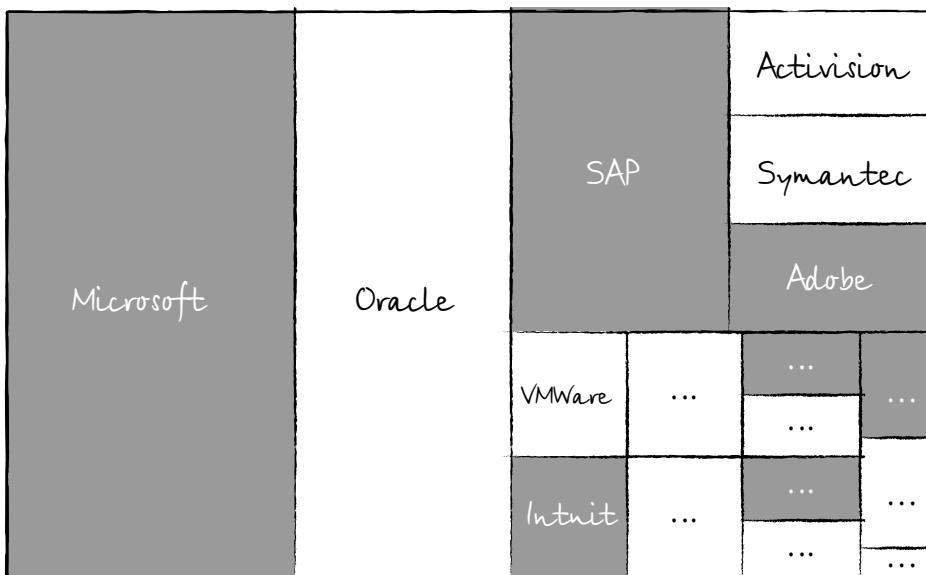


Рис. 16.4. Диаграмма, на которой размер прямоугольника представляет рыночную стоимость компании

На первый взгляд, размещение маленьких прямоугольников внутри более крупных не представляет особой проблемы. Но в действительности все гораздо сложнее, чем может показаться. Представьте одну из тех игр, в которых вам дают пару десятков блоков разных форм и размеров и предлагают уложить их в некоторую фиксированную форму таким образом, чтобы между блоками не было зазоров. Создание этих прямоугольников несколько похоже на такую игру, хотя речь в данном случае может идти о тысячах отдельных фрагментов, которые нужно уложить в большой прямоугольник. В ходе совещания, посвященного планированию спринта, владелец ее продукта указал этой команде следующие условия удовлетворенности.

- Прямоугольники по форме должны быть как можно ближе к квадрату: средняя величина отношения длинной стороны к короткой в идеальном случае должна равняться 1.1.
- На запланированном оборудовании данная система должна уметь генерировать каждую секунду пять таких диаграмм заданной сложности.
- Поддержка до 5 тысяч элементов на одной диаграмме.
- Поддержка до 500 групп на одной диаграмме.

Нетрудно заметить, насколько высокоуровневыми являются эти условия удовлетворенности. Это явилось одной из причин, по которым условия удовлетворенности были выбраны для описания “тестов”, применяющихся для пользовательской истории или элементов журнала запросов на выполнение работ. Условия удовлетворенности зачастую находятся на один уровень выше действительного приемочного теста. Например, чтобы создать действительный приемочный тест, соответствующий условию удовлетворенности “поддержка до 5 тысяч элементов на одной диаграмме”, нам нужно было бы располагать большей информацией об этих элементах: составление диаграммы из 5 тысяч элементов одинаковой величины является достаточно тривиальной задачей; решение такой задачи, когда речь идет о 5 тысячах элементов разной величины, — задача нетривиальная.

Информация об этих 5 тысячах элементов будет добавлена во время спринта. Тестер возьмет высокоуровневые ожидания владельца продукта (выраженные в виде условий удовлетворенности) и преобразует их в конкретные тесты. Кроме того, тестер добавит все конкретные тесты, подразумевавшиеся, но не указанные владельцем продукта. Например, для нас вполне очевидно, что владелец продукта хочет, чтобы диаграмма отображалась корректно, даже если она содержит лишь одну точку данных. Владелец продукта не сказал об этом команде потому, что это было и без того очевидно. Но это по-прежнему оставалось тем, что требовало кодирования и тестирования в ходе данного спринта.

Выполнение разработки на основе приемочных тестов заставляет команду постоянно сосредоточиваться на целях, поставленных владельцем продукта. Начиная спринт с условий удовлетворенности, сформулированных для каждой пользовательской истории, над которой предстоит работать команде в данном спринте (или формулируя их как можно раньше в ходе выполнения этого спринта), команда меньше рискует сбиться с пути (“О, а я-то думал, что вам нужно то-то и то-то!”) и избегает соблазна бесконечного совершенствования уже созданных функциональных возможностей (“Мне казалось, было бы здорово, если бы это работало вот так”). Еще одним преимуществом разработки на основе приемочных тестов является то, что она

стимулирует дополнительное общение между тестерами и другими разработчиками на той стадии, когда тестеры еще не до конца определились со своими обязанностями.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Выберите в журнале запросов на выполнение работ какой-либо элемент, над которым команда работает в текущем спринте. Предложите каждому из членов команды по отдельности сформулировать для него условия удовлетворенности. Затем, начиная с условий удовлетворенности, сформулированных владельцем продукта, проанализируйте условия удовлетворенности, сформулированные каждым из членов команды. Не исключено, что владелец продукта не указал какие-то из важных условий удовлетворенности. (В конце концов, мы не рассчитываем на то, что вы потратите на это упражнение очень много времени.) Но еще вероятнее то, что среди этих условий удовлетворенности окажутся условия, противоречащие друг другу. Вероятно также, что при выполнении данного упражнения вы столкнетесь с довольно неожиданными представлениями членов вашей команды о том, какие элементы журнала запросов на выполнение работ следует считать важными.

Выплатите технический долг

Технический долг, концепция, автором которой является Уорд Каннингем (Ward Cunningham), означал поначалу повышенную стоимость работы над приложением с “незрелым”, или “не вполне правильным”, кодом (1992). Сейчас этот термин используется в основном для обозначения стоимости работы над неудачно спроектированной, плохо кодированной, содержащей незавершенную работу и несовершенной во многих других отношениях системой. Каннингем предупреждает о последствиях накопления технического долга.

Опасность возникает, когда этот долг не выплачивается. Каждая минута, затраченная на “не вполне правильный код”, засчитывается как процент, выплачиваемый на такой долг. Целые проектные организации, испытывающие долговое бремя “сырой” разработки, могут в конечном счете оказаться в тупике (1992).

Технический долг зачастую является результатом поспешной разработки. Это не всегда плохо. Как пишет Каннингем, “сдача заказчику первого варианта системы — это почти всегда влезание в долг. Небольшой долг ускоряет разработку, если он своевременно выплачивается путем переписывания системы” (1992). Главным в этом случае является то, что долг должен выплачиваться быстро. Это удается не всегда. В результате у многих команд накапливается огромный технический долг. Поскольку Scrum-команды рассматривают свой продукт в долгосрочной перспективе, выплата технического долга становится для них важным соображением.

Некоторые виды технического долга совершенно очевидны. Появление непредусмотренных данных в базе данных приводит к аварийному завершению работы соответствующего приложения и, несомненно, является техническим долгом. Хрупкий код, который разрушается любым программистом, прикасающимся к нему, также, несомненно, является техническим долгом. Но как быть в случае команды, которая не выполнила обновление в соответствии с новой версией Java, появившейся в прошлом

месяце? Это также является техническим долгом. Вероятно, не является проблемой то, что команда еще не выполнила обновление, и я вовсе не хочу сказать, что каждая команда должна выполнять обновление сразу же после появления каждого нового инструмента. Однако долгом является использование даже несколько устаревшего языка, библиотеки или инструмента. Помните следующее соображение Каннингема: “Небольшой долг ускоряет разработку, если он своевременно выплачивается”.

Выплата технического долга в три этапа

Чтобы соответствовать принципам гибкой методологии разработки, команде нет необходимости одномоментно выплачивать весь накопившийся у нее технический долг. Разумеется, это было бы неплохо, но такая возможность у команд имеется далеко не всегда. Однако технический долг должен быть выплачен в объеме, достаточном для того, чтобы команда не рухнула под его тяжестью. В качестве примера того, как команда может выплатить накопившийся у нее технический долг, рассмотрим способ выплаты одной из самых распространенных форм технического долга: критической нехватки автоматизированных тестов.

Когда команда, целиком и полностью полагавшаяся на тестирование вручную, приступает к внедрению Scrum, она быстро осознает, насколько нелегко выполнять короткие спринты, если в каждом таком спринте приходится выполнять значительные объемы тестирования вручную. Команда приходит также к пониманию того, что если не принимать радикальных мер, накопление технического долга продолжится. Команды, оказавшиеся в подобной ситуации, могут воспользоваться трехэтапным процессом (показанным на рис. 16.5), чтобы избавиться по крайней мере от наихудших из этих проблем.



Рис. 16.5. Выплата стоимости тестирования вручную в три этапа

1. Остановить “кровотечение”.
2. Удержать ситуацию на достигнутом уровне.
3. Наверстать упущенное.

Первым приоритетом команды, накопившей технический долг в форме чрезмерного упования на тестирование вручную, является “остановка кровотечения”, т.е. прекращение дальнейшего ухудшения ситуации. Наилучшим вариантом является поиск способов автоматизации чего-то из того, что в данный момент тестируется вручную. Можно воспользоваться другой метафорой: команде следует найти фрукт, висящий не очень высоко, т.е. тесты, которые будет легко автоматизировать, но которые позволят

сэкономить значительный объем ручного труда. Брайан Марик (Brian Marick), ведущий специалист по тестированию и один из авторов Agile Manifesto, пришел к выводу, что “подлинным низковисящим фруктом зачастую оказывается не автоматизация выполнения каких-то тестов, а автоматизация других задач тестирования, таких как наполнение баз данных или автоматический переход на страницу, где вы начинается тестирование вручную. При этом вы сокращаете не количество ручных тестов, а суммарное время их выполнения”.

Во время выполнения указанных этапов члены команды приобретают квалификацию в области автоматизированного тестирования, которое может оказаться для некоторых сотрудников совершенной новинкой. Необходимо настроить тестовые серверы и тестовые среды, а также выбрать инструменты. Все это требует немалых затрат времени. Но если это не будет сделано, каждая новая функциональная возможность, добавленная в систему, будет лишь наращивать объем времени тестирования вручную, способствуя, таким образом, наращиванию технического долга. То, на что потребуется 20 часов в этом спринте, может потребовать 21 час в следующем спринте. Со временем проект может рухнуть под бременем технического долга, образовавшегося в результате ручного тестирования.

После того как удастся остановить кровотечение, ситуация перестанет ухудшаться от одного спринта к другому. Ручные тесты по-прежнему будут добавляться в каждом спринте, но в каждом спринте команда будет находить достаточное количество низковисящих фруктов, чтобы компенсировать время, необходимое для выполнения новых ручных тестов. С этого момента настает время перехода ко второму этапу — научиться удерживать ситуацию на достигнутом уровне. В течение этой фазы команда сосредоточивается на обучении тому, как писать и автоматизировать тесты для тех новых функциональных возможностей, которые добавляются в течение соответствующего спринта. На этом этапе дополнительный долг не накапливается и, соответственно, ситуация не ухудшается. Правда, она и не улучшается. Добавление автоматизированных тестов в том же спринте, в течение которого разрабатываются соответствующие функциональные возможности, — новый навык, который придется приобрести команде. Освоить его будет не так трудно, как первоначальные навыки, осваиваемые в течение первой фазы, но он потребует новой дисциплины.

Со временем команда перейдет в последнюю фазу, в течение которой она будет наверстывать дополнительный невыплаченный технический долг тестирования. Эта фаза изображена на рис. 16.5 снижающейся линией. Обычно я говорю командам, что меня не интересует крутизна этого снижения, поскольку уже само по себе это снижение является положительным фактом. Разумеется, я предпочел бы, чтобы долг сокращался как можно быстрее. Но при этом я подчеркиваю, что меня волнуют главным образом первые два этапа.

Качество зависит от команды

Упор на качество может иметь весьма важные последствия. Спустя девять месяцев после внедрения Scrum и выполнения рекомендованных выше этапов Стив Гриени (Steve Greene) из Salesforce.com говорит о том, что ими уже достигнута экономия в размере “более чем 300 человеко-часов на каждую основную версию и еще сотни человеко-часов по сроку службы всей совокупности версий исправлений. Все это

сэкономленное время мы тратим сейчас на такие приносящие непосредственную пользу вещи, как разработка новых функциональных возможностей, автоматизация, проектирование и т.п.” (2007).

Какими бы впечатляющими ни были эти результаты, они не являются чем-то необычным и могут быть достигнуты любой эффективной Scrum-командой. От низкокачественного продукта страдают все клиенты. Если тестирование не интегрировано в процесс разработки или не выполняется на должных уровнях, страдает вся команда. Освоение новых способов тестирования, обучение их применению в жестких временных рамках, налагаемых методологией Scrum, и выплата технического долга являются обязанностью всей команды. Это не те проблемы, решение которых можно полностью возложить на тестеров. Эффективная Scrum-команда будет постоянно отслеживать состояние своих методов тестирования и постоянно изыскивать способы их совершенствования.

Дополнительная литература

Adzic, Gojko. 2009. *Bridging the communication gap: Specification by example and agile acceptance testing*. Neuri Limited.

Автор этой превосходной книги описывает способы совершенствования коммуникаций между членами команды и всеми остальными лицами, заинтересованными в успешном выполнении проекта, путем использования спецификаций на основе примеров и формулирования условий удовлетворенности.

Crispin, Lisa, and Janet Gregory. 2009. *Agile Testing: A practical guide for testers and agile teams*. Addison-Wesley Professional.

Эта книга предназначена для каждого, кто хочет понять, как интегрировать тестирование в проект, выполняемый посредством гибкой методологии разработки. Тестирование разделяется на четыре квадранта (нечто, подобное уровням пирамиды автоматизации тестирования) и описываются типы тестирования, выполняемые в каждом из них. Эта книга дает тестерам, пользующимся гибкой методологией разработки, представления и навыки, необходимые для успешного исполнения их новых ролей.

Mugridge, Rick, and Ward Cunningham. 2005. *Fit for developing software: Framework for integrated tests*. Prentice Hall.

Эта книга начинается с изложения основ, а затем рассматривается конкретный случай. Первые 180 страниц книги предназначены для всех участников проекта (программистов, тестеров, экономистов и т.д.) и демонстрируют преимущества использования Fit при выполнении проектов. Следующие 150 (или что-то около того) страниц предназначены для тех, у кого имеется опыт программирования, и содержат описание способов расширения Fit путем написания и использования специализированных дополнений.

Koskela, Lasse. 2007. *Test driven: TDD and acceptance TDD for Java developers*. Manning

В части IV этой книги содержится превосходное описание разработки на основе приемочных тестов. Здесь излагаются доводы в пользу выполнения ATDD и описываются способы выполнения ATDD с помощью FIT (Framework for Integrated Tests — концептуальный каркас для интегрированных тестов). Кроме того, показано, как приступить к реализации своего проекта.

ЧАСТЬ IV

Организация

Во имя того, чтобы выжить в будущем, каждая организация должна быть готова отказаться от всего, что она делает в настоящем.

— Питер Друкер

Глава 17

Изменение масштаба Scrum

Моя жена, Лора, готовит ужин практически каждый день. Иногда, когда у нее появляется вдохновение, она готовит что-нибудь особенно вкусненькое. Когда же у нее нет настроения, она готовит еду попроще. Но в любом случае ее блюда вкусны, полезны и не являются результатом приложения титанических усилий. Исключение составляет лишь то, что она готовит к Рождеству. Приготовление рождественского ужина — это всегда стресс, ведь ожидается прибытие нескольких гостей: родителей жены, моих родителей, возможно, тети и дяди, а также брата и одной-двух сестер. Как правило, порций всегда оказывается больше, чем гостей. Масштаб рождественского ужина несопоставим с масштабом любого другого ужина в году. Все, что приготовлено в существенно большем масштабе, чем тот, к которому мы привыкли (в том числе проект, связанный с разработкой программного обеспечения), вызывает дополнительные трудности.

Когда приходится выполнять крупный проект, связанный с разработкой программного обеспечения, задача разработчиков усложняется не только тем, что на его реализацию уходит больше времени. Крупные проекты зачастую более важны для организации, им уделяется больше внимания, они в большей степени чувствительны ко времени, их реализация чаще вызывает личные конфликты и на них уходит больше времени. Нередко в проекте участвует несколько команд, удаленных географически друг от друга.

См. также Распределенная разработка влечет за собой столь уникальные проблемы, что мы решили уделить ей особую главу — следующую за этой.

Первый способ победы над крупными проектами заключается в том, чтобы атаковать их не одной крупной командой, а несколькими небольшими. В главе 10, “Структура команды”, читатели познакомились с идеей команды разработчиков, которую можно накормить двумя пиццами, т.е. команды из пяти-девяти человек. Реализуя

крупный проект, мы можем использовать несколько таких “команд на две пиццы”, вместо одной крупной команды.

В этой главе речь пойдет о способах эффективного применения методологии Scrum к реализации крупного, “многокомандного” проекта. В частности, мы рассмотрим изменение масштаба роли, исполняемой владельцем продукта, использование большого журнала запросов на выполнение работ, управление взаимозависимостями команд, участвующих в реализации крупного проекта, координирование работ, выполняемых этими командами, изменение масштаба совещания, посвященного планированию спринтов, а также роль сообществ практиков в реализации крупных проектов.

Изменение масштаба роли владельца продукта

Роль владельца продукта при выполнении Scrum-проекта может быть одной из самых важных. При выполнении любых проектов владельцу продукта приходится разрываться между конкурирующими между собой потребностями, обращенными внутрь команды, и потребностями, обращенными вовне команды. К числу задач, обращенных внутрь команды, относятся участие в совещаниях посвященных планированию спринтов, обзорах спринтов, ретроспективах спринтов и ежедневных совещаниях разработчиков; управление журналом запросов на выполнение работ; ответы на вопросы членов команды; да и, в конце концов, просто доступность для команды в течение спринта. К числу задач, обращенных вовне, относятся обсуждение с пользователями их потребностей, организация опросов пользователей и интерпретация результатов этих опросов, посещение организаций-заказчиков, посещение отраслевых выставок, управление ожиданиями лиц, заинтересованных в выполнении проекта, приоритизация журнала запросов на выполнение работ, определение цены продуктов, выработка средне- и долгосрочной стратегий для соответствующего продукта, отслеживание тенденций в отрасли и на рынке, выполнение конкурентного анализа и многое другое. В случае проекта, выполняемого одной командой разработчиков, это зачастую оказывается достаточно внушительным, но вполне обозримым объемом работы. Однако в случае крупного проекта, выполняемого несколькими командами разработчиков, роль владельца продукта оказывается непосильной для одного человека, поэтому необходимо изыскать способы ее масштабирования.

Когда масштаб проекта становится таким, что выполнять проект приходится нескольким командам, в идеальном случае у каждой команды должен быть свой владелец продукта. Если реализация такого варианта невозможна, постарайтесь, чтобы каждый владелец продукта отвечал не больше чем за две команды. Это обычно максимум, на который способен один владелец продукта, если мы хотим, чтобы его работа была достаточно эффективной.

На определенной стадии роста масштаба проекта имеет смысл сформировать иерархию владельцев продукта, сотрудничающих между собой. Такая иерархическая структура показана на рис. 17.1. В ней с каждой командой работает свой владелец продукта. Имеется также два владельца линеек продуктов, каждый из которых работает со своим кластером команд. Кроме того, предусмотрен главный владелец продукта. Естественно, в зависимости от масштаба конкретного проекта какие-то уровни могут добавляться или устраиваться.

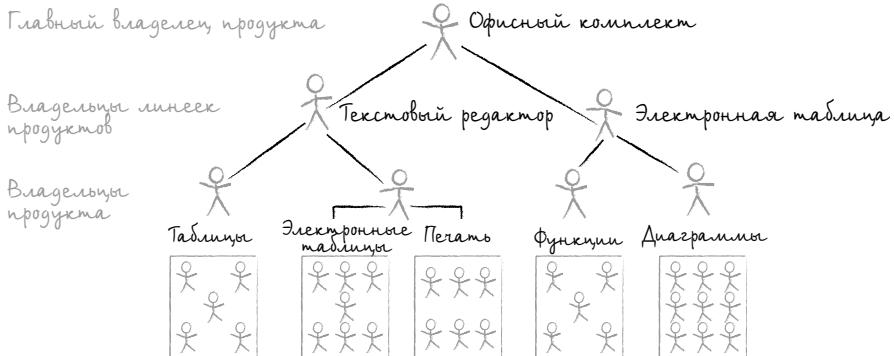


Рис. 17.1. Роль владельца продукта может расширяться и включать владельцев линеек продуктов и главного владельца продукта

Совместная ответственность и разделение функциональности

Главный владелец продукта должен знать, как будет выглядеть продукт (или комплект продуктов) в целом. Он доносит это понимание до всей команды в ходе общих собраний команды, путем рассылки писем по электронной почте, совещаний команд и любыми другими доступными ему способами. Но главный владелец продукта почти наверняка слишком занят, чтобы брать на себя ответственность за повседневную практическую деятельность команд, т.е. действовать так, как обычно действуют владельцы продукта команд в составе пяти-девяти человек, разрабатывающих соответствующий продукт. На данном уровне внешние требования этой роли слишком велики. Правильно понимающий свои задачи главный владелец продукта наверняка будет участвовать в делах команд: время от времени посещать ежедневные совещания разработчиков, время от времени заглядывать в помещения, где работают команды, высказывать свое мнение по тем или иным вопросам и предлагать свою помощь. Однако у него нет возможности вникать в тонкости работы, которой занимается каждая из команд, и разбираться в подробностях, касающихся сегментов их продуктов. Всем этим должны заниматься владельцы соответствующих линеек продуктов и владельцы конкретных продуктов.

Допустим, мы решили разработать офисный комплекс, направленный на повышение производительности офисных работников. Этот офисный комплекс должен включать текстовый редактор, электронную таблицу, программное обеспечение для создания презентаций и персональную базу данных. Конкурировать с Microsoft Office, Google Apps и другими продуктами известных производителей кажется малоперспективным делом, но владелец продукта не теряет оптимизма. Поскольку главный владелец продукта сосредоточится на решении стратегических вопросов, конкурентном позиционировании и тому подобном, отдельными продуктами этого офисного комплекса — текстовым редактором, электронной таблицей, программой для создания презентаций и базой данных — будут заниматься владельцы линеек продуктов. Каждый владелец линейки продуктов, в свою очередь, подбирает владельцев продуктов, которые будут отвечать за определенные наборы функциональных возможностей, реализуемых в рамках соответствующего продукта. Например, владелец линейки продуктов, отвечающий за текстовый редактор, может работать с владельцем продукта,

отвечающим за таблицы, с владельцем продукта, отвечающим за таблицы стилей и печать, с владельцем продукта, отвечающим за блок орфографического контроля, и т.д.

Хотя, как указывалось выше, главный владелец продукта — слишком занятой человек, чтобы исполнять роль владельца продукта для какой-то одной команды, вполне возможно, что он будет выступать в роли владельца линейки продуктов для какой-то части продукта. Возвращаясь к нашему примеру с офисным комплектом, отметим, что главный владелец продукта может принять решение исполнять роль владельца линейки продуктов, отвечающего за текстовый редактор. Он может принять такое решение, например, потому, что раньше ему уже приходилось выступать в такой роли. Аналогично кто-то из владельцев линейки продуктов может пожелать заниматься более практическими вопросами, выступая в роли владельца одного из продуктов. Например, владелец линейки продуктов, отвечающий за электронную таблицу, может выступать в роли владельца продукта команды, которая добавляет диаграммы в продукт “электронная таблица”.

Несмотря на возможность четкого распределения между командами задач по реализации функциональных возможностей, важно, чтобы все владельцы продукта чувствовали коллективную ответственность за продукт в целом. Они должны также внушить это чувство коллективной ответственности командам, с которыми работают.

ПРИМЕЧАНИЕ

Лично я предпочитаю употреблять словосочетания “главный владелец продукта” и “владелец линейки продуктов”, но это не более чем мое личное предпочтение. Вы можете, если пожелаете, использовать другие обозначения. В частности, мне встречались следующие термины: “владелец программы”, “супервладелец продукта”, “владелец сферы ответственности” и “владелец функциональной возможности”.

Стремясь к соблюдению единства терминологии в существующей литературе, посвященной Scrum, я предпочитаю использовать термин “владелец продукта” в отношении человека, который работает непосредственно с одной или двумя командами, приоритизируя их работу и выполняя все остальные обязанности, ассоциирующиеся с ролью владельца продукта. В этих многоуровневых иерархических структурах владельцем продукта зачастую является тот, в чьей визитной карточке написано “Business Analyst” (“Бизнес-аналитик”).

Ведение большого журнала запросов на выполнение работ

Большинство команд, работающих над крупными проектами, предпочитают использовать один из коммерческих инструментов гибкой методологии разработки, которые обеспечивают поддержку для использования большого журнала запросов на выполнение работ. Так что я не буду распространяться насчет собственных предпочтений, касающихся использования большого журнала запросов на выполнение работ, поскольку способ использования той или иной организацией ее журнала запросов на выполнение работ во многом зависит от того, каким инструментам отдает предпочтение соответствующая организация. Однако мне представляется целесообразным изложить две рекомендации, эффективность которых не зависит от конкретного инструмента.

- Если речь идет только об одном продукте, должен быть только один журнал запросов на выполнение работ.
- Размер журнала запросов на выполнение работ не должен превышать разумных пределов.

Эти рекомендации обсуждаются в последующих разделах.

Один продукт — один журнал

Есть причины, по которым журнал запросов на выполнение работ не называется “журналом работ проекта” или “журналом работ команды” или не обозначается каким-либо подобным, но столь же неадекватным термином. Он называется журналом запросов на выполнение работ потому, что для каждого продукта должен быть один такой журнал. Если команда работает на основе нескольких журналов запросов на выполнение работ, то этим журналам должны быть присвоены соответствующие приоритеты друг относительно друга. Недостаточно, однако, назначить каждому из журналов запросов на выполнение работ соответствующий приоритет и указать команде, чтобы она взяла по пять верхних элементов каждого из этих журналов. Верхний элемент в одном журнале запросов на выполнение работ может иметь более низкий приоритет, чем самый нижний элемент в каком-то другом журнале запросов на выполнение работ.

Рассмотрим в качестве примера компанию Ultimate Software из г. Уэстона, шт. Флорида, успешно внедрившую у себя Scrum. Ultimate Software разрабатывает программное обеспечение, работающее в качестве служб (Software as a Service — SaaS) и предназначеннное для управления кадрами. Оно включает в себя функциональные возможности, предназначенные для управления кадрами и платежными ведомостями. Несмотря на то что эти функциональные возможности, очевидно, взаимосвязаны между собой (кадры, которыми вы управляете, должны получать заработную плату), исходное программное обеспечение построено по модульному принципу. В Ultimate Software есть команды, которые специализируются на части продукта, которая соответствует управлению кадрами, и есть команды, которые специализируются на части продукта, отвечающей за управление платежными ведомостями. Однако даже с учетом того, что разные команды специализируются на разных подсистемах разрабатываемой системы, Ultimate Software ведет единый журнал запросов на выполнение работ для продукта в целом.

Наличие единого журнала запросов на выполнение работ позволяет главному владельцу продукта в Ultimate Software видеть взаимосвязь между высокоприоритетными функциональными возможностями управления кадрами и высокоприоритетными функциональными возможностями управления платежными ведомостями. Допустим, что все элементы, расположенные в верхней части журнала запросов на выполнение работ, соответствуют функциональным возможностям управления кадрами. Для главного владельца продукта это является указанием на необходимость либо перенаправить команды, специализирующиеся на платежных ведомостях, на разработку функциональных возможностей управления кадрами (где поначалу они будут менее производительны, поскольку недостаточно знакомы с данной областью или кодом), либо приказать им продолжить разработку низкоприоритетных функциональных возможностей, связанных с управлением платежными ведомостями.

Однако при использовании единого журнала запросов на выполнение работ несколькими командами и несколькими владельцами продукта не исключено появление логистических проблем. Хотя мы можем согласиться с тем, что все функциональные возможности должны быть приоритизированы друг относительно друга, осуществить это при выполнении проекта, предполагающего использование многочисленных владельцев продукта, нескольких владельцев линейки продуктов и одного главного владельца продукта, может оказаться чрезвычайно трудно. Вместо предоставления каждому владельцу продукта возможности вести собственный журнал запросов на выполнение работ лучше ограничиться одним журналом запросов на выполнение работ, но предоставить возможность пользоваться им каждому владельцу продукта. Такой подход представлен на рис. 17.2.

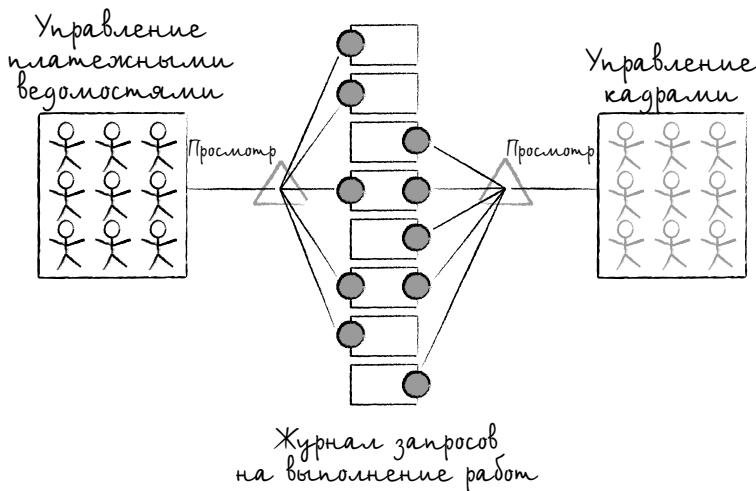


Рис. 17.2. Для каждого продукта должен быть предусмотрен один журнал запросов на выполнение работ, но просматриваться этот журнал может по-разному

На рис. 17.2 представлены две команды, использующие единый журнал запросов на выполнение работ. Команда, которая разрабатывает функциональные возможности системы, связанные с управлением кадрами (на рис. 17.2 справа), глядя в журнал запросов на выполнение работ, обращает внимание на элементы, за разработку которых она несет (или может нести) ответственность. Команда, которая разрабатывает функциональные возможности системы, связанные с управлением платежными ведомостями (на рис. 17.2 слева), также в первую очередь обращает внимание на те элементы журнала запросов на выполнение работ, за разработку которых она несет (или может нести) ответственность. Обратите внимание, что некоторые из элементов журнала запросов на выполнение работ могут представлять интерес для обеих команд. Это может указывать как на то, что соответствующая функциональная возможность может быть полностью реализована любой из команд, так и на то, что в разработке данной функциональной возможности может потребоваться участие обеих команд. Допустим, что команды, участвующие в разработке продукта “Офисный комплект” и специализирующиеся на текстовом редакторе и электронной таблице,

рассматривают журнал запросов на выполнение работ каждая со своей точки зрения. Скорее всего, они обе будут рассматривать элемент журнала запросов на выполнение работ, расширяющий возможности совместно используемого орфографического контроля.

Поддерживайте разумный размер журнала

Необходимо сбалансировать желание иметь единый журнал запросов на выполнение работ с желанием, чтобы журнал запросов на выполнение работ не разросся до такой степени, что стал неуправляемым. Очевидно, что эти желания противоречат друг другу. Мой собственный опыт свидетельствует о том, что проектируемая система быстро “идет вразнос”, если кому-то из участников проекта приходится держать в голове более 100–150 элементов журнала запросов на выполнение работ. Я могу указать две причины того, почему, как мне кажется, это можно рассматривать как разумный верхний предел. Во-первых, наблюдая за работой сотен Scrum-команд (или непосредственно участвуя в их работе), я неоднократно слышал нарекания на слишком большие размеры журнала запросов на выполнение работ. Каждый раз, когда я слышал подобные нарекания, это означало, что журнал запросов на выполнение работ содержит не менее ста элементов. Мой второй довод в поддержку того, что размер журнала запросов на выполнение работ не должен превышать 100–150 элементов, связан со случаем, который произошел со мной в 2000 году.

Занимая пост вице-президента по разработке новой продукции в одной из организаций, я был приятно удивлен, когда оказалось, что мои команды своевременно завершили выполнение очень крупного проекта. В честь этого неординарного события мы решили организовать вечеринку и поздравить команды с впечатляющим успехом. Вечеринка состоялась в банкетном зале гостиницы. В ней участвовало 160 членов команд, а также лиц, так или иначе причастных к этому успеху. Я прошествовал через весь зал в сопровождении своей жены, на ходу знакомя ее со своими товарищами по работе и участниками проекта. Однако во время этой церемонии со мной произошел конфуз: к нам подошла пара, с которой мне предстояло познакомить жену, а я не мог вспомнить, кто из этих двоих является участником проекта. Это было очень некрасиво: кто-то из этих двоих был моим подчиненным или подчиненным кого-то из моих подчиненных, но как я ни силился, так и не смог вспомнить, кто именно был этим подчиненным — “он” или “она”. Мой мозг изменил мне по причине, обусловленной так называемым “числом Данбара”. Впрочем, о существовании этого числа я узнал уже после этой вечеринки.

Британский антрополог Робин Данбар (Robin Dunbar) высказал предположение, что мозг среднестатистического человека устроен таким образом, что позволяет ему поддерживать нормальные социальные отношения не более чем со 150 другими индивидуумами. Иными словами, вы можете помнить имя каждого из них, помнить, кто он такой и кем он приходится остальным вашим знакомым (например, “Это Иоахим, он работает в группе тестирования”). Свыше этого предела у вас в голове начинается путаница. Вот вам и объяснение, почему я не смог запомнить каждого из участников той вечеринки. Но какое отношение это имеет к журналу запросов на выполнение работ? Возможно, мое объяснение носит менее научный характер. Однако если человеческий мозг устроен таким образом, что позволяет человеку запомнить не более 150 других индивидуумов и кем они приходятся друг другу, то, экстраполируя эту идею

на журнал запросов на выполнение работ, я прихожу к выводу, что большинство из нас способно помнить не более 100–150 элементов журнала запросов на выполнение работ и взаимосвязи между ними. В сочетании с многочисленными свидетельствами того, что команды, в журналах запросов на выполнение работ которых содержится 100 и более элементов, жалуются на непомерный объем своих журналов, а команды, в журналах запросов на выполнение работ которых содержится меньше элементов, подобных жалоб не высказывают, 100–150 элементов кажутся мне вполне разумным верхним пределом.

Хотя такое количество элементов журнала запросов на выполнение работ может показаться незначительным, следует принять во внимание, что в нашем распоряжении есть два способа контроля количества элементов журнала запросов на выполнение работ.

- **Пользуйтесь эпическими историями и темами.** Путем включения в журнал запросов на выполнение работ нескольких крупных историй (эпических историй) и путем тематического группирования мелких историй журнал запросов на выполнение работ даже для огромного проекта можно организовать таким образом, чтобы он соответствовал моим рекомендациям, т.е. чтобы в нем содержалось не более 150 элементов.
-

См. также С эпическими историями и постепенным уточнением журнала запросов на выполнение работ читатели познакомились в главе 13, “Журнал запросов на выполнение работ”.

- **Обеспечьте разные точки зрения на журнал запросов на выполнение работ.** Обратите внимание: я не утверждаю, что в журнале запросов на выполнение работ не может быть больше 150 элементов. Я лишь сказал, что одному человеку не следует держать в своей голове более 150 элементов журнала запросов на выполнение работ. Этого можно достичь, обеспечив разные точки зрения на журнал запросов на выполнение работ, как показано на рис. 17.2. Вообразите еще раз, что вы разрабатываете программу-конкурент таким офисным комплексам, как Microsoft Office и Google Apps. Главный владелец такого продукта не справится со своей работой, если ему одновременно придется помнить свыше 500 элементов журнала запросов на выполнение работ. Ему было бы полезнее видеть тематическую разбивку отдельных элементов журнала запросов на выполнение работ, тогда как командам нужно предоставить возможность видеть отдельные пользовательские истории.

Эта ситуация отражена на рис. 17.3, на котором показана часть журнала запросов на выполнение работ. Главный владелец продукта может видеть темы, которые представляют собой группы родственных пользовательских историй. Однако отдельные команды и владельцы их продуктов могут видеть отдельные пользовательские истории, представленные на таком уровне детализации, который необходим для их превращения в новые функциональные возможности продукта.

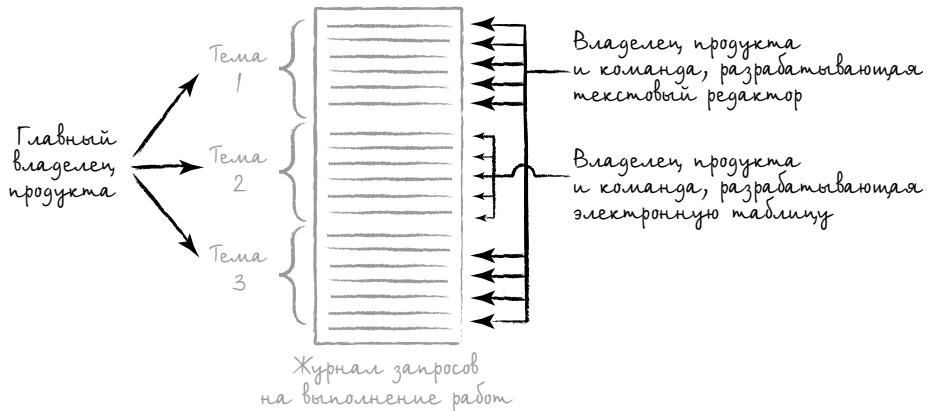


Рис. 17.3. Разные точки зрения на один и тот же журнал запросов на выполнение работ

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Если в вашем проекте используется несколько журналов запросов на выполнение работ, предложите владельцу продукта помочь в деле их объединения в единый журнал запросов на выполнение работ.
- ❑ Если в вашем журнале запросов на выполнение работ содержится свыше 100–150 элементов, сгруппируйте их по тематическому признаку. Напишите пользовательскую историю или другую описательную метку для данной темы, чтобы эту совокупность пользовательских историй можно было рассматривать как один элемент.

Упреждающий режим управления зависимостями

Реализация проекта Тома продвигалась в основном неплохо: команды уже успели привыкнуть к итеративной и инкрементной природе Scrum и приступили к полноценному автоматизированному тестированию, разработке программного обеспечения на основе составления тестов и даже к парному программированию. Том начал осваивать новую для себя роль Scrum-мастера, что было для него не так уж легко, учитывая тот факт, что в прошлом он был типичным руководителем проектов, исповедующим командный стиль управления. После каждого очередного спринта две команды Тома должны были демонстрировать новые функциональные возможности своего приложения (платежи в режиме он-лайн и перевод денег). Ход реализации проекта вполне удовлетворял заказчиков. Время от времени командам Тома удавалось выполнить лишь небольшую часть того, что они запланировали. В ходе ретроспектив команды анализировали глубинные причины таких задержек. Обращаясь ко мне по электронной почте в связи с этой ситуацией, один из членов команды, Кэмпбелл, подытожил выводы, к которым они пришли.

Практически каждый раз, когда мы срывали выполнение плана того или иного спринта, это объяснялось взаимозависимостями между двумя нашими командами. Обычно мы не задумывались над этим во время планирования спринтов, а если и задумывались, то неправильно оценивали соответствующий объем работ.

См. также Советы по структуризации команд приводились в главе 10, “Структура команды”. О непрерывной интеграции речь шла в главе 9, “Технические приемы”.

При выполнении любого многокомандного проекта (как в случае с Томом и Кэмпбеллом) вероятность возникновения взаимозависимости команд оказывается достаточно высокой. Правильный выбор структуры команд помогает снизить степень такой взаимозависимости, но не позволяет полностью избавиться от нее. Аналогично непрерывная интеграция помогает выявить проблемы, порождаемые некоторыми видами зависимостей. К счастью, существуют дополнительные методы, которыми могут воспользоваться Scrum-команды для решения проблем, вызванных этими зависимостями. Такими методами являются выполнение планирования с накатом на несколько последующих спринтов (*rolling lookahead planning*), проведение организационных совещаний, предваряющих очередной релиз-цикл (*release kick-off meeting*), использование несколькими командами одних и тех же специалистов и даже использование специализированной команды интеграции.

Планирование с накатом на несколько последующих спринтов

Зачастую бывает так, что команда, едва завершив свое совещание, посвященное планированию спринта, обнаруживает, что небольшую часть работы должна выполнить какая-то другая команда. Проблема, однако, в том, что такой команды в данный момент нет. Планирование с накатом на несколько последующих спринтов существенно снижает частоту возникновения такой проблемы, заставляя команды потратить в каждом спринте несколько минут на обдумывание того, чем они будут заниматься в ходе следующей пары спринтов. Как правило, самым подходящим временем для этого является окончание совещания, посвященного планированию спринта, когда команда и владелец продукта собрались вместе и занимаются планированием.

В зависимости от того, о какой команде идет речь, а также в зависимости от продолжительности спринта и ряда других факторов, планирование одного спринта может длиться от одного часа до целого рабочего дня. Планирование дополнительной пары спринтов во время того же совещания может показаться и невозможным, и чрезвычайно утомительным делом. К счастью, нам нужно лишь рассмотреть два последующих спринта, воспользовавшись понятием “средней исторической скорости” и не вникая в конкретные задачи или количество времени, которое понадобится для их решения. Это означает, что на планирование двух последующих спринтов потребуется примерно десять минут (мы исходим из того, что команде известна ее средняя скорость, а владелец продукта уже приоритизировал журнал запросов на выполнение работ).

Итак, команда, выполняющая планирование с накатом на несколько последующих спринтов, завершает совещание, посвященное планированию спринта, подробно спланировав очередной спринт (выбрав определенную совокупность элементов

журнала запросов на выполнение работ и определив время, которое потребуется для выполнения каждой задачи) и выбрав предварительно некоторую совокупность элементов журнала запросов на выполнение работ, которые должны быть реализованы в течение последующих двух спринтов. Соответствующий пример приведен в табл. 17.1. Этот пример иллюстрирует, что может представлять собой планирование с накатом на несколько последующих спринтов после планирования третьего спрингта. Спринтом позже команда определит задачи для элементов журнала запросов на выполнение работ четвертого спрингта и заглянет еще дальше в будущее, в пятый и шестой спринты, которые попадут в их поле зрения. Неважно, что элементы журнала запросов на выполнение работ, выбранные для последующих спринтов, претерпят изменения, когда начнется выполнение работы по этим спрингтам. Планирование с накатом на несколько последующих спринтов следует рассматривать как возможность принять во внимание, над чем команда будет работать впоследствии, а не как план, имеющий строго ограниченные рамки. Ничто не мешает владельцу продукта впоследствии изменить свое мнение на основе текущей информации.

Таблица 17.1. Планы с накатом на несколько последующих спринтов включают подробности для текущего спрингта, для последующих же двух спрингтов предусмотрены лишь элементы верхнего уровня

Спринт 3

Как посетитель сайта я могу читать текущие новости на начальной странице, чтобы знать, что сейчас происходит в мире Scrum и гибкой методологии разработки.

Кодировать средний уровень.	12 часов
-----------------------------	----------

Кодировать новый пользовательский интерфейс.	4 часа
--	--------

Проектировать и автоматизировать тесты.	12 часов
---	----------

Проектировать новый пользовательский интерфейс и проверить его работу с помощью нескольких пользователей.	8 часов
---	---------

Как редактор сайта я могу добавлять на сайт свежие новостные элементы, чтобы пользователи были в курсе последних событий.

Идентифицировать и внести в базу данных изменения.	12 часов
--	----------

Написать код Ruby on Rails.	4 часа
-----------------------------	--------

Проектировать и автоматизировать тесты.	8 часов
---	---------

Как посетитель сайта я могу читать новости, которых уже нет на начальной странице. Это нужно мне для того, чтобы находить старые новости, которые мне хотелось бы перечитать еще раз, или прочитать их впервые, если я в силу тех или иных причин пропустил их, когда они были впервые опубликованы на начальной странице.

Добавить код для реализации этих возможностей.	6 часов
--	---------

Проектировать и автоматизировать тесты.	8 часов
---	---------

Спринт 4

Как редактор сайта я могу устанавливать начальную и конечную даты для всех новостных элементов, чтобы на странице отображались только актуальные новости.

Как посетитель сайта я могу отправлять (посредством определенной формы) сообщения веб-мастеру данного сайта по электронной почте.

Как посетитель сайта я могу отправлять (посредством определенной формы) сообщения редактору данного сайта по электронной почте.

Спринт 5

Как посетитель сайта я хочу раз в неделю читать новую статью, публикуемую на первой странице.

Как посетитель сайта я могу выполнять полнотекстовый поисковый просмотр тела статьи, названия и фамилии автора.

Я рекомендую выполнять накат на два спринта вперед, поскольку такой подход предоставляет командам адекватное время для своевременного реагирования на большинство нововыявленных зависимостей. Чтобы узнать, почему так происходит, рассмотрим, что произошло бы, если бы команда планировала лишь на один спрингт вперед. Если вам нужно что-то от другой команды к началу следующего спринта, вам остается одно — попросить эту команду выполнить ваше пожелание в ходе предстоящего спринта. Если эта команда планировала предстоящий спрингт в то же время, что и вы, она, скорее всего, уже полностью спланировала этот свой спрингт к тому времени, когда вы обратились к ней с этой просьбой. Выполнение наката на два спринта вперед позволяет вам обратиться к этой команде с большим запасом времени. У нее будет достаточно времени для планирования и выполнения необходимой вам работы в следующем спринте, что даст ей возможность завершить эту работу к началу следующего спринта, т.е. именно тогда, когда вам понадобится результат этой работы. Некоторые команды (как правило, те, которые занимаются разработкой аппаратного обеспечения или проектированием встраиваемого оборудования), пользуясь планированием с накатом, могут выполнять накат и на большее число спринтов.

ВОЗРАЖЕНИЕ

“Нам едва хватает времени на планирование одного спринта. Мы ни в коем случае не хотим планировать сразу три спринта. Если мы станем планировать на два спринта вперед, то совещания, посвященные планированию спринтов, будут дублировать друг друга, поскольку один и тот же спрингт нам придется планировать дважды или даже трижды”.

Нужно помнить о том, что планирование с накатом на несколько последующих спринтов не предполагает разбиения пользовательских историй на задачи и оценивания трудоемкости (выраженной в количестве часов) каждой из этих задач, как это требуется при планировании текущего спринта. Команда просто использует свою среднюю скорость, чтобы прикинуть, какие элементы журнала запросов на выполнение работ могут быть реализованы в ходе последующих двух спринтов. Это почти всегда можно легко осуществить за каких-нибудь десять минут, если взять за основу приоритизированный журнал запросов на выполнение работ.

В ходе планирования с накатом на несколько последующих спринтов команда не принимает на себя обязательство реализовать в ходе последующих двух спринтов какую-то конкретную совокупность элементов. Она, скорее, оценивает, над чем ей придется работать впоследствии. Это необходимо команде для того, чтобы она могла определить любые зависимости или подготовительную работу, которая должна быть выполнена в предстоящем спринте.

Проведение организационных совещаний, предваряющих очередной релиз-цикл

Еще один способ упреждающего управления зависимостями — собрать всех исполнителей проекта для проведения организационного совещания, предваряющего очередной релиз-цикл. Идеальным моментом для проведения такого совещания является начало нового проекта или релиз-цикла. Такое организационное совещание может снизить один из серьезнейших рисков, возникающих при выполнении любого крупного проекта. Этот риск заключается в том, что разные команды или исполнители могут начать действовать в неправильных или разных направлениях.

До проведения этого организационного совещания каждая команда (которую можно накормить двумя пиццами!) вместе со своим владельцем продукта составляет примерный план того, что будет реализовано в течение обозримого времени (как правило, около трех месяцев). В ходе организационного совещания, предваряющего очередной релиз-цикл, команды делятся своими планами со всеми остальными участниками проекта. Как правило, владельцы продукта каждой команды поочередно делятся с остальными участниками проекта планами предстоящей работы своих команд.

Каждые три-четыре месяца Salesforce.com производит очередной новый выпуск своей платформы SaaS и считает весьма полезным проведение таких организационных совещаний, предваряющих очередной релиз-цикл. Эрик Бэбинет (Eric Babinet) и Раджани Раманатан (Rajani Ramanathan) утверждают, что “довольно нелегко выявить зависимости или договориться с другими командами о взаимных обязательствах, если эти команды еще не знают, чем именно они будут заниматься в ходе соответствующего релиза. Организационное совещание, предваряющее очередной релиз-цикл, исполняет для команд важную роль точки синхронизации и обеспечивает более продуктивное обсуждение взаимозависимостей команд” (2008, 403).

Команды в Salesforce.com внедрили у себя важное новшество, помимо обычного организационного совещания, предваряющего очередной релиз-цикл. Позже на той же неделе они проводят так называемое “Открытое пространство релиза” (Release Open Space). Такое название было выбрано по аналогии с подходом, который получил название “Технология открытого пространства” (Open Space Technology)¹ и приобрел в последние годы немалую популярность на конференциях. Каждой команде предлагается откомандировать для участия в этом открытом пространстве по меньшей мере одного своего члена. Это неформальное совещание начинается с того, что его участники формулируют актуальные темы, касающиеся соответствующего релиза, и записывают их на больших листах бумаги, развешанных на стенах. После того как темы будут сформулированы, формируются группы для обсуждения тем, представляющих интерес для членов этих групп. В Salesforce.com на такое обсуждение выделяется 45 минут. Обсуждение сопровождается 30-минутным подведением итогов. Этот цикл повторяется до тех пор, пока участники совещания проявляют живой интерес к темам, записанным на плакатах.

¹ Открытое пространство — это самоорганизующийся подход к проведению совещаний, конференций и т.п. Оно стало основным способом проведения конференций, посвященных гибкой методологии разработки. Подробнее об открытом пространстве можно прочитать на сайте http://en.wikipedia.org/wiki/Open_Space_Technology.

Совместное использование специалистов командами

Еще один возможный подход к упреждающему управлению зависимостями заключается в использовании несколькими командами одних и тех же специалистов. Этот подход достаточно эффективен в случаях, когда зависимости трудно определить заранее или когда проблему этих зависимостей нужно решить как можно быстрее. Подобная стратегия оказывается не очень эффективной в случаях, когда зависимости могут возникать между любым числом команд и в любом направлении. Однако она вполне приемлема, когда зависимости могут возникать между функциональными и компонентными командами.

См. также Разумеется, участие одних и тех же специалистов в нескольких командах не лишено недостатков. Некоторые из этих недостатков были рассмотрены в главе 10, “Структура команды”.

При таком подходе некоторые из специалистов оказываются членами сразу двух команд, работая, если можно так выразиться, по обе стороны известной или вероятной зависимости. Такой вариант представлен на рис. 17.4. На этом рисунке показаны три функциональные команды, в двух из которых частично занято по одному специалисту из компонентной команды. Кроме того, один специалист является одновременно членом двух функциональных команд.

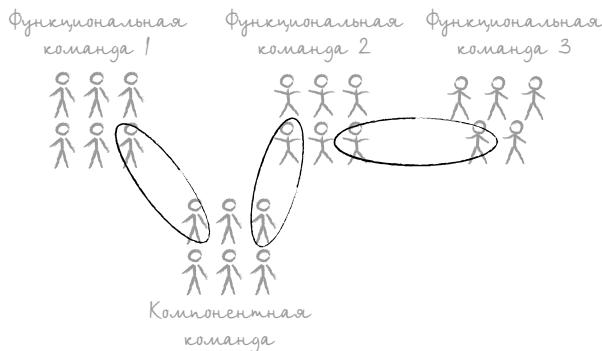


Рис. 17.4. Использование несколькими командами одних и тех же специалистов — хороший способ обеспечения связи между командами

Использование специализированной команды интеграции

Несмотря на то что использование несколькими командами одних и тех же специалистов иногда является шагом в правильном направлении, этого шага может оказаться недостаточно. Иногда необходимо создать интеграционную команду. Это чаще всего случается при выполнении проектов, в которых задействуется не менее десяти команд. Интеграционная команда работает в “зазорах”, которые могут возникать между командами разработчиков. Большинство таких зазоров возникает в местах взаимодействия команд, если можно так выразиться, на их стыках (Sosa, Eppinger, and Rorles, 2007). Соответствующие проблемы можно разделить на две обширные категории.

- **Невыявленные стыки.** Невыявленный стык — это реально существующий, но еще не обнаруженный стык.
- **Необслуживаемые стыки.** Необслуживаемый стык — это, реально существующий стык, о котором известно по меньшей мере одной команде, однако эта команда ничего не делает для решения связанных с ним проблем.

Интеграционные команды сосредоточиваются непосредственно на необслуживаемых стыках, отслеживая в то же время ситуацию с невыявленными стыками. После того как интеграционная команда выявит необслуживаемый или невыявленный стык, она прежде всего должна побудить одну из команд разработчиков взять на себя ответственность за него. Если же это невозможно или нецелесообразно по практическим соображениям, то интеграционная команда сама принимает на себя ответственность за этот стык.

Как правило, первое, что делают каждое утро члены интеграционной команды, — это проверка результата официальной ежевечерней сборки, чтобы убедиться в том, что система успешно собрана и что все тесты выполнены. Если результаты такой проверки оказались отрицательными, члены интеграционной команды делают все необходимое для прохождения всех тестов. Обычно это включает выявление соответствующей проблемы, поиск ее источника, определение причастной команды (или команд) и последующую работу с этими командами с целью разрешения выявленной проблемы.

При выполнении крупного проекта интеграционная команда может быть сформирована из членов, работающих в ней полный рабочий день. Вообще говоря, при выполнении особо крупного проекта может быть сформировано несколько интеграционных команд, члены каждой из которых работают в них полный рабочий день. Многие другие проекты — те, в которых задействовано от нескольких до десятка команд — вполне обходятся виртуальной интеграционной командой, формальные члены которой в основном посвящают свое время работе в своих командах разработчиков. Члены такой виртуальной интеграционной команды собираются каждое утро, чтобы обсудить состояние сборки, выполненной накануне вечером, и договариваются между собой, кто какие проблемы будет решать. Остаток рабочего дня члены виртуальной интеграционной команды посвящают работе в своих командах разработчиков.

Обычно в начале выполнения нового проекта специалисты назначаются в интеграционную команду на несколько спринтов. Перед этой командой ставится задача настройки всех необходимых серверов и конфигурирования общепроектного программного обеспечения, такого как вики-доски, серверы постоянной интеграции и т.п. После того как все эти системы будут готовы, члены интеграционной команды возвращаются к работе в своих командах разработчиков, а сама интеграционная команда становится структурой, вступающей в действие лишь в случае необходимости.

Рассмотрим в качестве примера крупную компанию из Сан-Франциско, специализирующуюся на биоинформатике. В этой компании есть примерно десяток функциональных команд, две компонентные команды и одна интеграционная команда. Помимо отслеживания ежевечерних сборок и решения возникающих проблем эта интеграционная команда занимается разработкой автоматизированных тестов, предназначенных для проверки точек интеграции. Это такие типы тестов, ответственность за которые невозможно с очевидностью возложить ни на функциональные, ни на компонентные команды. Тем не менее кто-то должен взять на себя ответственность за

эти тесты, без выполнения которых продукт невозможно передать заказчику. На всех обычных совещаниях, проводимых функциональными и компонентными командами, как правило, присутствует представитель интеграционной команды. Бывают дни, когда тот или иной представитель интеграционной команды участвует в трех ежедневных совещаниях разработчиков. Они участвуют в этих совещаниях — а также в совещаниях, посвященных планированию спринта, обзора спринтов и ретроспективных совещаниях — в надежде выявить необслуживаемые или невыявленные стыки.

Поскольку членство в интеграционной команде требует хороших аналитических навыков, в том числе умения сопоставлять комментарии, высказанные разными командами в разное время, интеграционную команду не следует рассматривать как отстойник для посредственных работников. Напротив, членами интеграционной команды должны быть опытные сотрудники с широким кругом навыков. Учитывая только что сказанное, первоначальная стажировка в интеграционной команде является для новых сотрудников превосходным способом составить представление о разрабатываемой системе в целом. Кроме того, подобная стажировка является для новых сотрудников очень структурированным способом познакомиться с каждым из участников проекта и установить важные связи, которые пригодятся им в дальнейшем. Нужно лишь позаботиться о том, чтобы ваша интеграционная команда не состояла в основном из новых сотрудников.

ВОЗРАЖЕНИЕ

“Если бы проект действительно реализовывался посредством гибкой методологии разработки, интеграционная команда не понадобилась бы. Если бы в результате каждого спринта команда действительно создавала продукт, потенциально пригодный для передачи заказчику, интеграционная команда также не понадобилась бы. Использование интеграционной команды является признаком того, что команда разработчиков неэффективно использует гибкую методологию разработки”.

Обычно, когда я слышу подобные высказывания, они принадлежат людям, которым не приходилось участвовать в по-настоящему крупных проектах. Иначе говоря, эти высказывания являются лишь предположениями, и не более того. Линда Райзинг (Linda Rising), независимый консультант, которой приходилось участвовать в ряде очень крупных проектов, включая разработку программного обеспечения для самолета Boeing 777, по ее собственным словам, “никогда не участвовала в крупных проектах, в которых бы не было интеграционной команды”.

Мне кажется, нет ничего удивительного в том, что людям, которым никогда не приходилось участвовать в очень крупных проектах, кажется, что такие проекты можно выполнять как совокупность небольших проектов, каким-то образом “склеенных” между собой. Проблема, с которой сталкиваются очень крупные проекты, заключается в том, что такого “клена” может оказаться слишком много, особенно если такую ситуацию не предвидели. Со временем качество этого “клена” ухудшается.

Во-вторых, использование интеграционной команды не следует рассматривать как неадекватность других команд. Напротив, использование интеграционной команды является признаком того, что мы имеем дело с крупным или сложным проектом.

Рассмотрим альтернативу: каждая команда должна самостоятельно находить все необслуживаемые или невыявленные стыки между собой и каждой из остальных команд. Совокупное время и силы, затраченные на такой поиск всеми командами, оказались бы гораздо большими, чем время и силы, затраченные с той же целью интеграционной командой. Каждая из функциональных и каждая из компонентных команд должны отвечать самым высоким требованиям в том, что касается количества и типов интеграционных проблем, возникающих на стыках, которые предстоит выявить интеграционной команде, но само по себе использование интеграционной команды вовсе не является признаком того, что команда разработчиков неэффективно использует гибкую методологию разработки.

Тем не менее, хотя в использовании интеграционной команды нет ничего изначально порочного, применять ее следует лишь в случае крайней необходимости.

Координация работы команд

Поскольку при реализации крупных проектов, согласно методологии Scrum, использование нескольких небольших команд предпочтительнее использования одной большой команды, возникает проблема координации работы всех этих команд. Scrum-мастер Джоанна рассказала мне, как она осознала важность такой координации после выполнения первого в ее жизни многокомандного проекта. Вдохновившись успехом однокомандного пилотного Scrum-проекта, Джоанна приняла предложение стать Scrum-мастером проекта, в реализации которого должно было принять участие пять команд. Этим командам предстояло работать вместе над созданием новой версии продукта, разрабатывавшегося ее компанией и выполняющего диспетчеризацию карет скорой помощи. Она провела краткий курс обучения каждой из этих пяти команд и дала старт реализации проекта. В течение первых четырех спринтов дела, как и следовало ожидать, шли вполне успешно. Но затем начали проявляться зависимости между командами, со временем становившиеся все более критичными. В конце концов стало очевидным, что команды действуют изолированно друг от друга и каждая из них пытается как можно быстрее продвигаться к своей собственной цели, не уделяя при этом должного внимания точкам интеграции. Джоанна сообщила мне по электронной почте, что не позаботилась должным образом о налаживании межкомандного взаимодействия.

Каждому члену каждой из команд было хорошо известно, что должна делать его команда. Если мы забывали о чем-то во время планирования спринта, команда вспоминала об этом в ходе спринта и решала возникшую проблему. Но никто не занимался отслеживанием тысячи мелких проблем, накапливавшихся между командами. Это было похоже на то, как два игрока в бейсбол смотрят на падающий между ними мяч, но ни тот, ни другой не бросается ловить этот мяч, надеясь, что это сделает его визави.

Между командами не было ни малейшей вражды или конкуренции. Проблема заключалась лишь в том, что каждая из команд была до такой степени сосредоточена на достижении своих целей, что не обращала внимания на общую цель. В главе 11, “Организация коллективного труда”, я рассказывал о том, что работа Scrum-команд

базируется на общекомандном мышлении и совместной ответственности. При выполнении многокомандных проектов “команда в целом” — это не отдельно взятая команда (которую можно накормить двумя пиццами) плюс ее владелец продукта и Scrum-мастер, а все “двуухпиццевые” команды плюс их владельцы продукта и Scrum-мастера.

В этом разделе мы рассмотрим, что могла бы сделать Джоанна для улучшения координации действий ее команд. В частности, мы рассмотрим способ проведения *объединенных Scrum-совещаний* (scrum of scrums meetings) и необходимость синхронизации дат начала и завершения спринтов.

Объединенное Scrum-совещание

Весьма распространенным способом координирования деятельности нескольких команд является объединенное Scrum-совещание. Такие совещания позволяют группам команд обсуждать свою работу, сосредоточиваясь главным образом на сферах, где их деятельность пересекается, а также на интеграции.

Представьте себе идеально сбалансированный проект, над выполнением которого работают семь команд, каждая из которых состоит из семи человек. Каждая из них будет проводить (независимо от остальных) собственные ежедневные совещания разработчиков и делегировать одного своего представителя для участия в объединенных Scrum-совещаниях. Кто именно будет представлять на таком совещании ту или иную команду, определяет сама команда. Обычно таким представителем оказывается кто-то из технических специалистов — например, программист, тестер, администратор базы данных или дизайнер, — а не Scrum-мастер или владелец продукта. Право представлять команду на объединенных Scrum-совещаниях не является пожизненным: в ходе реализации одного и того же проекта команду могут представлять на объединенных Scrum-совещаниях разные ее члены. Выбирая своего представителя для участия в объединенных Scrum-совещаниях, команда исходит из того, насколько хорошо тот или иной кандидат понимает и комментирует проблемы, которые могут возникнуть в ходе реализации проекта.

Если количество команд, участвующих в реализации проекта, невелико, то каждая из команд может при желании делегировать для участия в объединенных Scrum-совещаниях двух своих представителей: кого-либо из технических специалистов (как указывалось выше) и своего Scrum-мастера. Лично я выбираю такой вариант, как правило, лишь в тех случаях, когда в реализации проекта участвуют не более четырех команд (таким образом, в объединенных Scrum-совещаниях участвуют не более восьми человек). Большинство таких объединенных Scrum-групп не назначают себе конкретного Scrum-мастера. В конце концов, участники объединенных Scrum-совещаний являются членами самоорганизующихся команд. Однако в некоторых группах находится человек, который добровольно принимает на себя функции Scrum-мастера. В этом случае сама группа решает, нужен ли ей такой Scrum-мастер.

Объединенные Scrum-совещания могут, в свою очередь, строиться по иерархическому принципу, если необходимость в этом диктуется масштабом проекта. Если крупный продукт разрабатывается несколькими командами команд, то по одному представителю каждого из объединенных Scrum-совещаний могут участвовать в так называемом объединенном совещании объединенных Scrum-совещаний (scrum of scrum of scrums meetings). Поскольку такое название звучит несколько

глупо, большинство организаций предпочитает пользоваться названием “объединенное Scrum-совещание” независимо от того, о каком иерархическом уровне идет речь в каждом конкретном случае. Пример такой иерархической структуры Scrum-совещаний показан на рис. 17.5. На этом рисунке представлено 11 отдельных команд. Они сгруппированы в три “команды команд”, каждая из которых проводит свое совещание. Но поскольку эти три команды команд занимаются разработкой единого для них продукта, предусмотрен еще один уровень совещаний, в которых принимают участие по одному представителю от каждого из объединенных Scrum-совещаний.

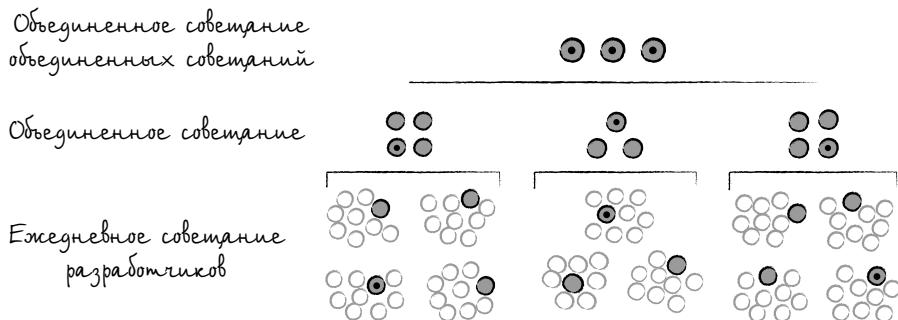


Рис. 17.5. Понятие объединенного Scrum-совещания может применяться рекурсивно на стольких уровнях, сколько необходимо для эффективного координирования работы кластера команд

Частота

Объединенные Scrum-совещания отличаются от ежедневных совещаний разработчиков в трех отношениях.

- Их не нужно проводить ежедневно.
- Их не нужно ограничивать 15-минутными рамками.
- Они посвящены решению проблем.

Я считаю, что проводить объединенные Scrum-совещания два-три раза в неделю вполне достаточно для большинства проектов. Вполне подходящим для проведения таких совещаний я считаю график “вторник, четверг” или “понедельник, среда, пятница”. Хотя для проведения объединенных Scrum-совещаний зачастую вполне достаточно 15 минут, я рекомендую отводить для них от 30 до 60 минут. Это объясняется тем, что, в отличие от ежедневных совещаний разработчиков, объединенные Scrum-совещания посвящаются решению проблем. Если на обсуждение этой группы выносится та или иная проблема, а в обсуждении участвуют люди, разбирающиеся в подобных проблемах, то 30–60 минут может оказаться вполне достаточно.

Задумайтесь над тем, сколько людей могут ожидать решения соответствующей проблемы. Вполне возможно, что ответа от объединенного Scrum-совещания ожидают около ста человек (гораздо больше людей могут ожидать ответа от объединенного совещания объединенных Scrum-совещаний). Проблемы, выносимые на обсуждение этой группы, должны решаться как можно быстрее. Это означает, что продолжительность таких совещаний не должна ограничиваться жесткими временными рамками, а решение вопросов нельзя переносить на какой-то другой день.

См. также Определенные типы проблем зачастую решаются сообществами практиков, которые представляют собой механизм учета масштаба проекта, описанный далее в этой главе.

Разумеется, иногда возникают проблемы, которые невозможно решить немедленно. Быть может, для этого нужны другие специалисты или дополнительная информация. Когда проблему невозможно решить немедленно, она фиксируется в *журнале проблем* соответствующей группы, представляющем собой перечень нерешенных проблем, которые данная группа либо планирует решить, либо намерена отслеживать их решение какой-либо другой группой. Для ведения такого журнала зачастую достаточно простого, низкотехнологичного механизма отслеживания. Большинство команд использует для этой цели большой лист бумаги, вывешенный в помещении, где работает команда, электронную таблицу или вики-доску.

Повестка дня

Объединенное Scrum-совещание практически ничем не похоже на ежедневное совещание разработчиков, несмотря на сходство в названиях. Ежедневное совещание разработчиков — это синхронизационное совещание: отдельные члены команды собираются, чтобы обсудить текущую работу и синхронизировать свои действия. Объединенное же Scrum-совещание — это совещание, посвященное решению проблем, и совершенно непохожее по своему стилю на скоротечные ежедневные совещания разработчиков, проходящие в темпе “давай-давай”. Повестка дня объединенного Scrum-совещания представлена в табл. 17.2. Как видно из этой таблицы, объединенное Scrum-совещание, подобно ежедневному совещанию разработчиков, начинается с того, что каждый из участников совещания отвечает на три следующих вопроса.

1. Что с момента последней нашей встречи сделала моя команда такого, что могло бы повлиять на другие команды?
2. Что до того, как мы встретимся вновь, сделает моя команда такого, что может повлиять на другие команды?
3. В решении каких проблем, возникших перед моей командой, мы могли бы получить помощь от других команд?

Таблица 17.2. Повестка дня объединенного Scrum-совещания включает три вопроса, сопровождаемых обсуждением элементов журнала проблем

Продолжительность	Пункт повестки дня
Ограничивается 15 минутами	Каждый из участников совещания отвечает на три следующих вопроса. Что с момента последней нашей встречи сделала моя команда такого, что могло бы повлиять на другие команды? Что до того, как мы встретимся вновь, сделает моя команда такого, что может повлиять на другие команды? В решении каких проблем, возникших перед моей командой, мы могли бы получить помощь от других команд? <i>Примечание.</i> В ходе этой части совещания не допускаются ссылки на конкретные лица
По мере необходимости	Решение проблем и обсуждение отдельных элементов журнала проблем

Темы, поднятые во время этого обсуждения, добавляются в журнал проблем данной группы. Предполагается, что эта часть совещания протекает достаточно быстро и не требует много времени. Ее, как и ежедневное совещание разработчиков, следует ограничить 15 минутами. Одним из способов достижения этой цели является применение следующего правила: в ходе этой части совещания не допускаются ссылки на конкретные лица. На это есть две причины. Во-первых, не ссылаясь на конкретные лица, можно поддерживать обсуждение на приемлемом уровне детализации. Участвуя в подобном совещании, я хочу услышать о каждой команде, а не о каждом члене каждой команды. Во-вторых, слишком многие склонны ставить знак равенства между важностью совещания и его продолжительностью. Следуя указанному правилу, можно максимально ускорить проведение этой части совещания.

После того как каждый ответит на эти три вопроса, участники совещания рассматривают любые проблемы, поднятые в ходе первоначального обсуждения или зафиксированные к тому времени в журнале проблем.

Группы, участвующие в объединенных Scrum-совещаниях, не занимаются формальным планированием спринтов и обзорами спринтов. Прежде всего, участники этих совещаний являются представителями соответствующих команд. Объединенное Scrum-совещание самого высокого уровня является наиболее мобильной и недолговечной группой, состав которой постоянно изменяется в ходе выполнения проекта. Планирование спринтов и деятельность, которая оказывает непосредственное влияние на реализацию проекта, осуществляются на уровне отдельных команд.

Синхронизация спринтов

Когда я выполнял свой первый Scrum-проект, мы приступали к его реализации силами лишь одной команды. Вскоре к нам подключились еще две команды, и между этими тремя командами образовались типичные зависимости. Довольно быстро я придумал способ, который, по моему мнению, позволял достаточно эффективно управлять этими зависимостями. Я решил смещать даты начала спринтов у разных команд на одну неделю друг относительно друга, как показано на рис. 17.6. Идея заключалась в том, что когда какая-то из команд подходит к началу своего спринта, она должна знать, какие истории уже решила реализовывать одна из оставшихся команд и какие истории эта команда, вероятнее всего, завершит.

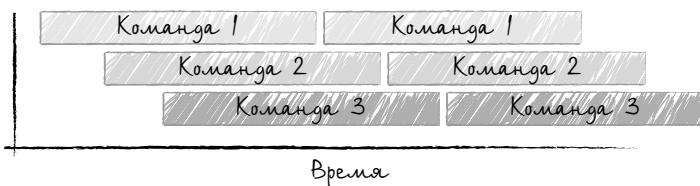


Рис. 17.6. Перекрывающиеся спринты команд порождают проблемы

Что ж, эта часть моего плана действительно доказала свою эффективность. Но в целом, как выяснилось, такое ступенчатое расположение дат начала спринтов у разных команд является крайне неудачной идеей. Самым крупным недостатком перекрывающихся спринтов является то, что никогда не наступает момент, когда все команды завершили свою работу (единственным исключением в этом смысле является завершение проекта). Одна или несколько команд всегда продолжают выполнять свои спринты. Одни команды планируют следующий спринт, другие выполнили

такое планирование неделю назад, а остальные приступят к планированию лишь на следующей неделе. Это не позволяет передать всю систему заказчику (для пробного использования и оценки) или операционной группе (для развертывания).

Все спринты необязательно завершаться в один и тот же день. При реализации крупного проекта вполне допустимо, если даты завершения спринтов у разных команд укладываются в двух- или трехдневный период. У такого подхода есть даже определенные преимущества. Если завершение спринтов у разных команд может укладываться в двух- или трехдневный период, то представителям разных команд не составит особого труда посетить все совещания, посвященные обзору и планированию спринтов, в которых должен участвовать сотрудник, состоящий в нескольких командах. Кроме того, преимуществом такого подхода во многих случаях является возможность лучше решить проблемы с посещением членов удаленных команд, которым для участия в таких совещаниях приходится ездить в другие города. Члену удаленной команды, участвующему в работе нескольких команд, будет легче оправдать время и деньги, затрачиваемые им на командировки, если он сможет полноценно участвовать в каждом из совещаний его команд.

Хотя главным преимуществом синхронизированных спринтов является то, что все команды начинают и заканчивают свои спринты примерно в одно и то же время (с разницей не более чем в два-три дня), это не означает, что все команды должны работать спринтами одинаковой продолжительности. Проект, в реализации которого участвует несколько команд, может предполагать разные продолжительности спринтов у разных команд при условии использования так называемых “вложенных спринтов”, как показано на рис. 17.7. Типичным случаем использования вложенных спринтов является вариант, когда разные команды, участвующие в проекте, не могут использовать одинаковую для всех продолжительность спринтов (например, когда одни команды предпочитают двухнедельные спринты, а другие — четырехнедельные).

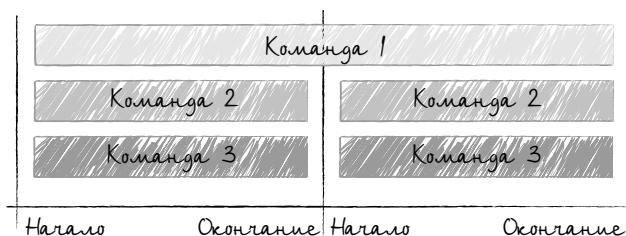


Рис. 17.7. Для обеспечения синхронизации спринтов всех команд их продолжительность необязательно должна быть одинаковой

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Синхронизируйте спринты команд, которые работают над одним и тем же проектом. Испытайте синхронизацию в течение двух спринтов, а затем проведите объединенную ретроспективу и обсудите, насколько полезной оказалась эта синхронизация. Попытайтесь найти решения проблем, которые возникли в ходе вашего эксперимента.
- ❑ Если вы еще не проводите продуктивные объединенные Scrum-совещания, попытайтесь делать это описанным здесь способом. Многие из команд, которые не знали, как проводить подобные совещания, таким образом достигли успеха.

Изменение масштаба совещания по планированию спринтов

Большинство стандартных совещаний, проводимых Scrum-командами, не претерпевают изменений при наращивании масштаба проектов и количества команд, участвующих в их реализации. Команды продолжают проводить ежедневные совещания разработчиков, обзоры спринтов и их ретроспективы точно так же, как они делают это, работая над проектом, выполняющимся силами одной команды. Разумеется, иногда команды решают проводить совместные обзоры, когда им представляется целесообразным совместно рассмотреть работу нескольких команд. Время от времени команды проводят совместные ретроспективы, чтобы уяснить ситуацию с выполнением проекта в целом и, возможно, сосредоточиться на рассмотрении межкомандных вопросов и проблем. Однако в наибольшей степени рост масштаба Scrum-проекта и увеличение количества команд, участвующих в его реализации, влияет на совещания, посвященные планированию спринтов.

См. также Конкретные рекомендации по проведению таких совещаний с распределенными командами приводятся в главе 18, “Рассредоточенные коллективы”.

Когда над одним и тем же проектом работает несколько команд, в ходе совещания, посвященного планированию спринтов, возникает ряд проблем, в том числе следующие.

- Некоторым специалистам нужно принимать участие в нескольких совещаниях, посвященных планированию спринтов. Если все спринты начинаются в один и тот же день, людям требуется быть сразу в нескольких местах.
- Если одна команда выявляет зависимость от какой-то другой команды, то не исключено, что она не сможет убедить эту другую команду взять на себя выполнение соответствующей задачи, если эта другая команда заканчивает свое планирование первой.
- Если несколько команд выбирают для себя элементы из одного и того же журнала запросов на выполнение работ, то эти элементы должны быть распределены между командами еще до того, как начнется планирование спринтов.

К счастью, существует пара подходов к крупномасштабному планированию спринтов, которые могут смягчить или даже полностью устраниТЬ эти (а также другие, подобные им) проблемы.

Смещение на один день

Как указывалось в предыдущем разделе, посвященном синхронизации спринтов, преимущества синхронизации могут быть сохранены даже в случае спринтов, рассинхронизированных друг по отношению к другу на один-два дня. Этим можно воспользоваться, и вместо того чтобы заставлять все команды заниматься планированием спринтов строго в один и тот же день, можно предложить одной трети команд заниматься планированием спринтов, например, во вторник, еще одной трети команд — в среду и еще одной трети — в четверг. В таком случае удастся в основном избавиться от проблем, связанных с совместным использованием журнала запросов

на выполнение работ. Если моя команда собирается планировать свой спринт завтра, то нам будет известно, какие элементы журнала запросов на выполнение работ ваша команда выбрала для себя сегодня.

Смещение на один день помогает также решить проблему одновременного присутствия в двух местах соответствующего владельца продукта, архитектора, проектировщика пользовательского интерфейса или другого специалиста, задействованного в нескольких командах. Однако вместо этой проблемы возникает другая, которую, по меткому выражению одного владельца продукта, можно было бы назвать “проблемой трехдневной головной боли”. “Совместно используемый” член команды способен помочь другим командам, но это может быть слишком утомительным для него: кому понравится присутствовать на совещаниях, посвященных планированию спринтов, на протяжении двух-трех полных рабочих дней подряд?

Несмотря на этот недостаток, смещение на один день остается приемлемым вариантом (как правило, для проектов, в которых участвуют до девяти команд, причем каждый день планированием занимаются три команды). В конце концов, даже при использовании смещения на один день, планирование обычно длится очень много дней, поэтому необходимы другие подходы.

Большая комната

При использовании подхода под названием “большая комната” все команды (или как можно большее их число) собираются в одной большой комнате. Главный владелец продукта начинает совещание с соображений, которыми ему хотелось бы поделиться с этой группой команд. Речь, возможно, пойдет о результатах недавних встреч с заказчиками, потенциальными клиентами или пользователями; возможно, главный владелец продукта предложит общее описание того, над чем предстоит работать всем командам на протяжении одного-двух следующих спринтов. После такого вступления каждая из команд (включая ее владельца продукта и Scrum-мастера) собирается в том месте комнаты, где она сможет поработать в течение двух-трех часов. Одна из команд может собраться в облюбованном ею углу комнаты, другая — у стены, третья — вокруг пары столов, сдвинутых в центре комнаты, и т.п. Собравшись таким образом, каждая из команд приступает к планированию предстоящего спринта, делая это точно так же, как если бы она была единственной командой, задействованной в данном проекте.

Комната наполняется шумом и гамом, но в такой атмосфере витают энергия и, смеем надеяться, энтузиазм. По мере того как команды планируют свои спринты, начинают проявляться зависимости между командами. Когда это происходит, один из членов команды поднимается, направляется к другой команде (которая, как мы помним, находится в той же комнате) и спрашивает: “Может ли ваша группа сделать для нас то-то и то-то в этом спринте? Нам это понадобится, чтобы завершить один из элементов журнала запросов на выполнение работ, как того требует наш владелец продукта”. Высказавший это пожелание может либо подождать ответа, либо вернуться к своей команде и подождать ответа вместе с ней. Ответ от другой команды может поступить через несколько минут.

Подход “большая комната” превосходно зарекомендовал себя в случаях, когда речь идет о критичных совместно используемых ресурсах. Владелец продукта, работающий

с двумя командами, может подходить попеременно то к одной, то к другой команде; главный архитектор программного обеспечения компании — слишком занятой человек, чтобы принадлежать какой-то одной команде, но нужный всем командам — также может переходить от одной команды к другой, которой он требуется в данный момент. В большинстве проектов та или иная команда обычно сигнализирует о своем желании пообщаться с владельцем продукта, архитектором или каким-либо другим “совместно используемым ресурсом”, громко обратившись к нему. Обычно это срабатывает — за исключением случаев, когда “совместно используемый ресурс”, завершив свое текущее обсуждение, просто забывает о том, какая из команд только что обратилась к нему за помощью.

Метод, который мне показался удачным, заключается в том, чтобы подкреплять словесные обращения к “совместно используемому ресурсу” использованием морских сигнальных флагов, как показано на рис. 17.8. Когда членам команды требуется, например, архитектор, они поднимают соответствующий сигнальный флаг размером 1 1 дюйм. Когда архитектор завершает свои текущие дела, он осматривает комнату в поисках “своего” флага. Если членам команды требуется владелец продукта, они вывешивают другой сигнальный флаг. Команде, которая вывесьла флаг владельца продукта, может помочь либо ее собственный владелец продукта, либо, возможно, владелец линейки продуктов или даже главный владелец продукта, если это возможно. Такой метод доказал свою эффективность: он позволяет решить большинство проблем, касающихся выяснения того, какая из команд нуждается в помощи в данный момент и позволяет “совместно используемому ресурсу” увидеть, сколько команд нуждаются в его помощи в данный момент; наконец, он просто забавен.

Морской смысл	Наш смысл
	Хочу говорить
	Нужна помощь
	Беру курс на порт
	Человек за бортом
	Нам нужен архитектор
	Нам нужен владелец продукта
	Время заказывать обед
	У нас перерыв

Рис. 17.8. Для просьбы о помощи — как на море, так и на совещании — можно воспользоваться флагами

Чтобы подход “большая комната” принес как можно большую пользу, владельцы продукта должны заранее подготовиться к совещанию такого рода. Это обычно предполагает проведение ряда коротких совещаний между главным владельцем продукта и его подчиненными. На каждом уровне иерархии владельцев продукта должно сформироваться собственное представление о продукте, соответствующее этому уровню. Представление о продукте изменяется по мере того, как оно “просачивается” сквозь разные иерархические уровни, начиная с уровня главного владельца продукта.

Создание сообществ практиков

При выполнении многокомандного проекта возможно нарастание изолированности разработчиков, которые общаются главным образом с членами своих команд. Такая изолированность препятствует распространению продуктивных идей в организации. В результате, например, схожие функциональные возможности по-разному реализуются разными командами. Пытаясь каким-то образом решить эти проблемы, мы проводим объединенные Scrum-совещания, но они не в состоянии полностью решить все проблемы. Дополнительным решением, причем таким, которое оказывается жизненно важным для успеха любого крупного Scrum-проекта, является создание и культивирование сообществ практиков. Подобно сообществам в поддержку перехода к Scrum (с которыми читатели познакомились в главе 4, “Движение в направлении гибкости”) сообщество практиков представляет собой группу единомышленников или специалистов примерно одинаковой квалификации, главной причиной добровольного объединения которых является приверженность определенной технологии, подходу или представлениям. При выполнении крупного проекта такие сообщества практиков помогают преодолеть разобщенность команд и собрать специалистов из многих межфункциональных команд. Соответствующий пример представлен на рис. 17.9.

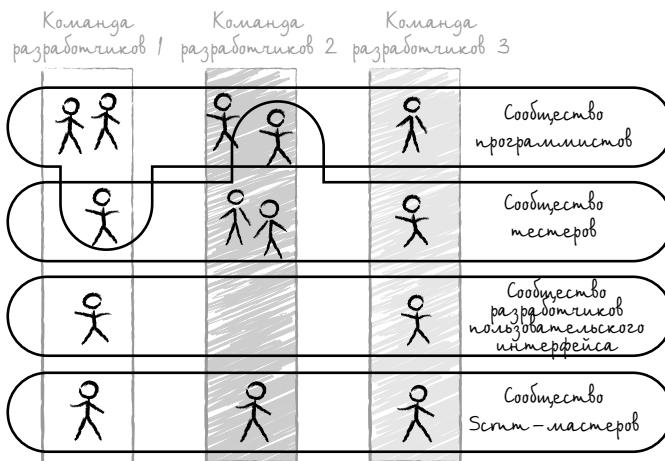


Рис. 17.9. Сообщества практиков пересекают границы команд и создают дополнительные каналы общения

На рис. 17.9 показаны сообщества практиков, сформированные по ролевому признаку, т.е. на основе ролей, используемых при реализации Scrum-проекта. Кроме того, при реализации достаточно крупного Scrum-проекта сообщества практиков могут формироваться и по технологическому признаку (например, Ruby-сообщество или .NET-сообщество), по интересам (например, имитация объектов, искусственный интеллект, автоматизация тестов) или на основе какой-либо взаимосвязанной совокупности интересов, общей для нескольких команд разработчиков.

Хорошим примером сообщества практиков является Virtual Architecture Team (Виртуальная архитектурная команда) в Salesforce.com, описанная Эриком Бэбинетом (Eric Babinet) и Раджани Раманатаном (Rajani Ramanathan).

Команда Virtual Architecture Team (VAT) является “виртуальной”, поскольку составлена из разработчиков, представляющих каждую из Scrum-команд. Являясь членами тех или иных Scrum-команд, члены VAT работают в своей виртуальной команде по совместительству. VAT занимается сопровождением и расширением нашей программной архитектуры, ведущей в отрасли. Они делают это, определяя архитектурную “дорожную карту”, анализируя архитектурно значимые изменения в коде и определяя стандарты с целью обеспечения совместимости и сопровождаемости кода (2008, 405).

Сообщества практиков могут охватывать несколько проектов. Например, сообщество, созданное на базе автоматизации тестов, может включать исполнителей, представляющих несколько совершенно не связанных между собой проектов. В отличие от сообществ в поддержку перехода к Scrum, типичное сообщество практиков не создается во имя достижения какой-либо отдельно взятой, конкретной цели. Напротив, у такого сообщества, как правило, есть несколько связанных между собой целей. Именно поэтому продолжительность деятельности сообщества практиков, вообще говоря, ничем не ограничивается. Сообщество практиков может быть расформировано после того, как оно достигнет всех поставленных перед ним целей, или после того, как его члены перестанут видеть смысл в дальнейшем его существовании.

Поскольку сообщества практиков включают представителей разных команд, они являются основным механизмом распространения продуктивных идей среди команд и обеспечения желаемых уровней совместимости и единства среди команд разработчиков. Сообщество программистов промежуточного уровня может, например, обсуждать и решать, когда лучше всего приступить к обновлению последней версии программного обеспечения сервера приложений для определенного семейства продуктов. Обсуждения между членами команды ортогонального тестирования должны обеспечить единство использования тестового инструмента и обмен передовым опытом.

Формальные или неформальные

Сообщество практиков может быть формальным или неформальным. В большинстве организаций используется сочетание формальных и неформальных сообществ. Организации, которые располагали сильным функциональным менеджментом до внедрения Scrum, как правило, полагаются на этих сильных функциональных менеджеров и в деле поддержки или даже формирования некоторых из сообществ практиков. Этьен Венгер (Etienne Wenger), который является автором термина “сообщество практиков”, и его коллеги различают пять типов сообществ практиков. Основой такой классификации является степень признания сообщества соответствующей организацией. Эти пять типов сообществ перечислены в табл. 17.3.

В табл. 17.3 не отражена некая иерархическая структура. Ни один из типов сообществ практиков не является заведомо лучше остальных — у каждого из них есть свои уникальные достоинства и недостатки. Точно так же в табл. 17.3 не указана продолжительность существования типичного сообщества практиков, хотя из этой таблицы легко видеть, как то или иное сообщество может постепенно трансформироваться из *непризнанного в институционализированное* по мере того, как его цель становится все более ясной для членов этого сообщества, а его полезность становится все более ясной для соответствующей организации. Несмотря на то что сообщества практиков иногда

проходят все перечисленные этапы трансформации из непризнанного в институционализированное, такая трансформация вовсе не является обязательной. Многие сообщества трансформируются в противоположном направлении, а некоторые быстро распадаются.

Таблица 17.3. Степень признания сообществ практиков простирается от сообществ, о существовании которых никому неизвестно, до сообществ, получивших официальный статус. (Заимствовано из Wenger, McDermott, and Snyder, 2002.)

Тип сообщества	Определение	Типичные проблемы
Непризнанное	Невидимое для соответствующей организации, а возможно, даже для своих членов	Очень нелегко уяснить ценность данного сообщества для организации или ее членов; возможно, не включает членов, которых должно было бы включать
Полулегальное	Видимое, но только для небольшой, избранной группы членов	Трудности с получением ресурсов или доверия; трудно оказывать влияние
Узаконенное	Официально санкционированное как полезная организация	Нереалистичные ожидания; быстрый рост и прием новых членов
Получающее поддержку	Получающее соответствующие ресурсы (время, деньги, оборудование, люди)	Ответственность за получение прибыли на инвестированные ресурсы; необходимость как можно быстрее доказать свою полезность для организации
Институционализированное	Имеющее официальный статус и предусматривающее определенные обязанности в соответствующей организации	Заорганизованность; неповоротливость; стремление продолжать свое существование даже при отсутствии реальной пользы для дела; постоянные члены отстраняются от участия в реальных проектах

Создание условий для формирования и процветания сообществ практиков

Самым эффективным типом сообществ практиков, которые создаются в рамках Scrum-организаций, являются, по-видимому, такие, которые возникают как бы сами по себе, а не по распоряжению администрации, хотя оба эти подхода могут использоваться для разных целей. Поскольку самоорганизация имеет решающее значение для успеха гибкой методологии разработки, выполняемой силами коллектива разработчиков, формирование самоорганизующихся сообществ практиков обусловливает возникновение мощного синергизма. В этом смысле именно организация и ее руководство должны заботиться о формировании обстановки, в которой могут образовываться, достигать своего расцвета, а затем, утратив свою ценность для организации, распадаться сообщества практиков.

Согласно Этьенну Венгеру и его коллегам, “поскольку сообщества практиков ограничены по своей природе, их создание является в большей степени вопросом способствования их эволюции, чем вопросом их создания «с нуля»” (Wenger, McDermott, and Snyder, 2002, 51). Этьенн Венгер и его коллеги сформулировали семь принципов формировании обстановки, способствующей такой эволюции.

- **Создавайте для эволюции.** Исходите из того, что каждое сообщество практиков со временем будет претерпевать изменения. Ценность каждого такого

сообщества будет возрастать и снижаться, состав его будет изменяться, и вполне вероятна трансформация целей сообщества. Такие изменения нужно только приветствовать, поскольку они свидетельствуют о реакции сообщества на изменения, касающиеся проекта, людей и организационных потребностей.

- **Инициируйте диалог между членами сообщества и другими сотрудниками.** Несмотря на то что сообщества практиков в основном “варятся в собственном соку”, они не могут быть полностью изолированы от организации в целом. Эффективное сообщество практиков небезразлично к тому, в чем нуждается соответствующая организация, за что она борется и что она может предоставить своим клиентами.
- **Допускайте разные уровни участия.** Не каждый, кого интересует деятельность сообщества практиков, может уделять ему одинаковое время и силы. Побуждайте людей участвовать в деятельности сообществ практиков на том уровне и с той частотой, какие им подходят.
- **Проводите открытые и закрытые мероприятия.** Эффективное сообщество практиков понимает, что самые плодотворные обсуждения происходят в тесном кругу единомышленников. Понимает оно и то, что иногда необходимы открытые мероприятия. В качестве примера можно привести сообщество, занимающееся совершенствованием опыта применения некоторой совокупности продуктов. Члены такого сообщества собираются в узком кругу раз в две недели для обсуждения тех или иных идей, но время от времени проводят открытые собрания, участие в которых может принять любой сотрудник компании, желающий знать, над какими проблемами работает данное сообщество.
- **Сосредоточивайтесь на полезности.** Чем большую ценность представляет для организации данное сообщество практиков, тем больше других сообществ такого типа появится в этой организации. К тому же, если сообщества представляют определенную ценность для организации, они получают большую поддержку и им предоставляется большая свобода действий.
- **Сочетайте привычные и неординарные мероприятия.** Зрелое сообщество практиков нередко приобретает ряд привычек, например еженедельные сеансы конференц-связи, ежемесячные совещания и ежегодные двухдневные собрания в штаб-квартире компании. Несмотря на полезность подобных мероприятий, целесообразно также время от времени проводить неординарные мероприятия, способные встряхнуть сообщество, вдохнуть в него свежие силы. Приглашение стороннего лектора, имеющего иную точку зрения, или проведение мероприятия в стиле “открытого пространства” — вот лишь два примера того, как можно встряхнуть сообщество практиков.
- **Задавайте для сообществ определенный ритм.** Вообще говоря, сообщества практиков не работают такими же регулярными спринтами, как команды Scrum-разработчиков. Если нет необходимости обеспечивать определенные результаты, ориентированные на конкретный проект (что обязаны делать команды Scrum-разработчиков), спринты для сообществ практиков необязательны. Однако все же лучше, если такое сообщество будет работать ритмично. Этого можно добиться, если предусмотреть для сообщества некую регулярную совокупность действий, которую оно должно выполнять с подходящей для себя периодичностью.

Если вы хотите, чтобы сообщества практиков создавались сами по себе, без административного вмешательства, позаботьтесь об обеспечении определенных мотиваций. Потенциальные члены сообщества должны знать, что создание сообщества практиков не только допустимо, но и приветствуется руководством организации. Мне, однако, пришлось столкнуться с несколькими ситуациями, когда руководство организации, предоставляя своим сотрудникам значительную свободу действий в деле создания сообществ практиков, не могло понять, почему такие сообщества не создаются. Когда я спрашивал об этом членов команд, они говорили мне, что опасались, что создание таких сообществ не понравится руководству организации. Постарайтесь довести до сведения своих команд, что в создании таких сообществ нет никакой крамолы; напротив, руководство организации будет приветствовать их появление.

Участие

Большинству формально признанных сообществ практиков выгодно наличие координатора сообщества. Координатор сообщества не является лидером данной группы, а исполняет в сообществе практиков две важные функции.

- Развивает ту практику, на основе которой сформировалось данное сообщество.
- Способствует развитию самого сообщества.

Координатор сообщества исполняет эти функции, планируя совещания и другие мероприятия, убеждая членов принимать в них участие, сводя между собою людей, имеющих общие интересы, лично участвуя в мероприятиях, проводимых сообществом, и т.п. В некоторых отношениях роль координатора сообщества подобна роли Scrum-мастера. Мой собственный опыт показывает, что исполнение роли координатора сообщества требует от 5 до 20 часов в месяц (в зависимости от направленности работы сообщества). Координатор сообщества может посвящать исполнению этой роли все (или почти все) свое рабочее время, если этому сообществу в организации поручено нести определенные формальные обязанности.

Что касается того, сколько часов член той или иной команды разработчиков должен уделять работе в сообществе, то здесь твердых правил не существует. Эта величина может колебаться в очень широком диапазоне, начиная с нескольких часов в год и заканчивая несколькими часами в неделю. Эрик Бэбинет и Раджани Раманатан указывают на относительно высокую степень задействованности членов команд разработчиков в работе уже упоминавшейся Virtual Architecture Team в компании Salesforce.com.

Дважды в неделю VAT собирается на два часа, чтобы обсудить техническую реализацию продуктов и функциональных возможностей, создаваемых Scrum-командами. Командам, реализующим наиболее сложные функциональные возможности в очередной версии продукта, предлагается представить свою работу VAT. VAT знакомит Scrum-команду со своей оценкой того, как предложенные этой командой технические решения влияют на другие области или как другие области влияют на технические решения, предложенные этой командой. Главное внимание VAT уделяет вопросам технической реализации, в частности соображениям производительности и наращивания масштаба. Команды, которым было предложено внести существенные изменения, должны еще раз представить свою работу VAT в том же релиз-цикле [примерно три месяца] и подробно рассказать о том, какие изменения они внесли в свой проект (2008, 405).

Сообщества практиков стоят времени, ресурсов и усилий, затрачиваемых на их создание и поддержание их деятельности. Польза, которую они приносят, способствуя взаимодействию и координации в масштабе всей организации или какого-либо крупного проекта, поистине бесценна. Если в вашей организации сообщества практиков еще не сформированы, попытайтесь создать такое сообщество на основе темы, интересующей вас или доставляющей вашей организации немалую головную боль. Когда это сообщество начнет приносить пользу вашей организации, вполне вероятно появление в вашей организации и других сообществ практиков.

Scrum и масштаб проекта

Можно лишь восхищаться интеллектуальной честностью авторов первых трудов, посвященных гибкой методологии разработки. Они не забывали подчеркнуть, что гибкие методологии разработки, подобные Scrum, предназначены для мелких проектов. Такой консерватизм объяснялся вовсе не тем, что гибкие методологии разработки (и, в частности, Scrum) непригодны для использования в крупных проектах, а тем, что сами они не использовали эти процессы в крупных проектах, а потому отказывались советовать своим читателям использовать гибкие методологии разработки (в том числе и Scrum) в крупных проектах. Однако в годы, минувшие после опубликования Agile Manifesto, мы узнали (в том числе из книг, которые вышли в свет как непосредственно перед опубликованием Agile Manifesto, так и после этого), что принципы и методы гибкой методологии разработки могут распространяться и на крупные проекты, хотя и со значительными накладными расходами. К счастью, если крупные организации используют описанные методы, касающиеся роли владельца продукта, применения совместно используемого журнала запросов на выполнение работ, учета зависимостей, координирования работ, выполняемых разными командами, и формирования сообществ практиков, то они могут с успехом реализовать Scrum-проект любого масштаба.

Дополнительная литература

Beavers, Paul A. 2007. Managing a large “agile” software engineering organization. В сборнике *Proceedings of the Agile 2007 Conference*, под ред. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 296–303. IEEE Computer Society.

В этом отчете описываются первые два года внедрения Scrum в компании BMC Software. Внедрение Scrum осуществлялось в ходе выполнения крупного проекта, в реализации которого участвовало 250 человек. Данный отчет, особенно ценен тем, что его автором является непосредственный участник проекта, представляет собой превосходное описание успехов и неудач, которые пришлось испытывать техническому руководителю этого проекта. Отчет завершается изложением девяти “правил успеха”, сформулированных автором на основе собственного опыта, полученного в ходе реализации данного проекта.

Larman, Craig, and Bas Vodde. 2009. *Scaling lean & agile development: Thinking and organizational tools for large-scale Scrum*. Addison-Wesley Professional.

Эта книга охватывает много тем, но в главе 11 Ларман и Водде сосредоточивают свое внимание на реализации крупных Scrum-проектов. Они представляют два концептуальных каркаса для изменения масштаба Scrum: один из них рассчитан на Scrum-проекты, выполняемые силами не более чем 10 команд, а другой — на Scrum-проекты, выполняемые силами большего числа команд.

Leffingwell, Dean. 2007. *Scaling software agility: Best practices for large enterprises*. Addison-Wesley Professional.

В этой книге описываются два типа масштабирования: изменение масштаба гибкой методологии разработки в рамках крупных организаций и изменение масштаба гибкой методологии разработки при выполнении крупных проектов. Несколько больше внимания уделяется первому типу масштабирования; впрочем, и тот, и другой типы рассматриваются достаточно подробно. Сердцевиной этой книги является часть II, в которой рассматриваются семь аспектов гибкой методологии разработки и способы изменения их масштаба: команды, двухуровневое планирование, итерации, мелкие релизы, параллельное тестирование, непрерывная интеграция и регулярное обдумывание.

Wenger, Etienne, Rechard McDermott, and William M. Snyder. 2002. *Cultivating communities of practice*. Harvard Business School Press.

Эта книга является авторитетным источником информации о сообществах практиков. Здесь можно найти сведения о том, как стимулировать образование таких сообществ, направлять их деятельность и измерять приносимую ими пользу. Указываются также некоторые недостатки, связанные с использованием сообществ практиков.

Глава 18

Рассредоточенные коллективы

Несколько лет назад нормой были команды, сосредоточенные в одном месте, тогда как географически рассредоточенные команды были редкостью. Сегодня все должно быть наоборот. Лично я сейчас удивляюсь, когда кто-то говорит мне, что все члены его команды работают в одном помещении. В наше время, когда преобладают команды, рассредоточенные по всему земному шару или по меньшей мере по нескольким часовым поясам, следует рассмотреть, насколько успешно могут использовать методологию Scrum географически рассредоточенные команды.

Типичное заблуждение заключается в том, что методология Scrum не очень-то подходит для географически рассредоточенной команды. Сторонники такой точки зрения утверждают, что предпочтение, отдаваемое этой методологией личному общению между членами команды, делают Scrum не лучшим вариантом для рассредоточенных команд. К счастью, такое утверждение ошибочно. Несмотря на справедливость утверждения о том, что команда, сосредоточенная в одном месте, всегда демонстрирует более высокие результаты, чем рассредоточенная команда (Ramasubbu and Balan, 2007), технология Scrum способна реально помочь географически рассредоточенным командам добиваться результатов, близких к тем, которые демонстрируют команды, сосредоточенные в одном месте. Допустим, вы приняли решение доверить реализацию части своего проекта разработчикам, работающим на каком-то другом континенте. Почему бы вам не воспользоваться перечисленными ниже преимуществами?

- Способность более точно оценивать ситуацию с исполнением проекта за счет возможности измерять реально достигнутый прогресс в конце каждого спринта.
- Возможность переопределять приоритеты после каждого спринта.
- Более частое общение исполнителей проекта.
- Акцент на качестве и автоматизации тестов.
- Более эффективная передача знаний, особенно между разработчиками, выполняющими парное программирование.

Очевидно, что выгоды, обусловленные применением Scrum, намного перевешивают любые трудности, которые могут обуславливаться опорой этой методологии на частое общение исполнителей проекта. Однако это вовсе не означает, что внедрение Scrum в рассредоточенной команде похоже на легкую прогулку. Как утверждают Майкл Вакс (Michael Vax) и Стефан Мишо (Stephen Michaud), “перенесение гибкой методологии разработки на рассредоточенную модель — занятие не для слабонервных” (2008, 314).

Эта глава посвящена действиям, которые может предпринять рассредоточенная команда для выведения своей производительности на уровень, максимально близкий к уровню, которого она могла бы достигнуть, если бы все ее члены работали в одном месте, сохраняя в то же время преимущества своей рассредоточенности, такие как экономия издержек, способность набирать сотрудников в разных городах и т.п. По ходу дела мы рассмотрим оптимальный подход к структурированию команды, члены которой рассредоточены на обширном географическом пространстве, рассмотрим, как создать спаянную рассредоточенную команду, оценим потребность времени от времени собирать всех членов рассредоточенной команды в одном месте, рассмотрим необходимые изменения в общении членов команды, а также подходящие способы проведения совещаний.

Решите, как рассредоточить несколько команд

Когда реализация проекта предполагает использование нескольких Scrum-команд, нужно принять важное решение об организации работы этих команд, расположенных на значительных расстояниях друг от друга. В реализации крупного проекта могут принимать участие несколько команд, *сосредоточенных в разных местах и взаимодействующих между собой*, или несколько *преднамеренно рассредоточенных команд*. Эти альтернативы представлены на рис. 18.1, который отображает ситуацию с использованием двух мест расположения команды.

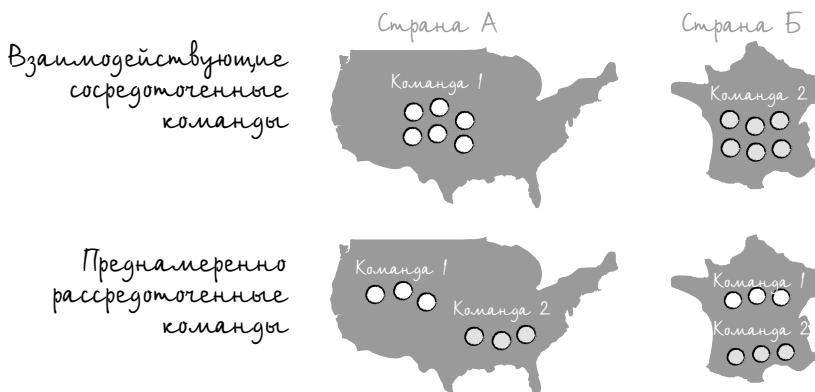


Рис. 18.1. Два разных подхода к рассредоточению команды между Соединенными Штатами Америки и Францией

Команды, сосредоточенные в разных местах и взаимодействующие между собой, создаются, когда в реализации проекта принимает участие количество людей,

сосредоточенных в двух или более городах, оказывающееся достаточным для того, чтобы в каждом из этих городов создать отдельную команду. Каждая из таких команд включает в себя всех специалистов, требующихся для того, чтобы провести любой элемент журнала запросов на выполнение работ через все стадии — от идеи до реализации. Эти команды называются командами, сосредоточенными в разных местах и взаимодействующими между собой, чтобы подчеркнуть то обстоятельство, что каждая из них работает вместе с другими удаленными (но сосредоточенными каждой в своем городе) командами над созданием единого для них продукта, т.е. не является совершенно независимой командой, которая входит в состав той же компании, что и другие независимые команды. В отличие от команд, сосредоточенных в разных местах и взаимодействующих между собой, преднамеренно рассредоточенные команды встречаются в случае, когда проект *мог бы* реализовываться командами, сосредоточенными в разных местах и взаимодействующими между собой, но компания принимает осознанное решение рассредоточить их.

У каждой из этих альтернатив есть, конечно же, свои преимущества и недостатки. Основным преимуществом реализации проекта с помощью команд, сосредоточенных в разных местах и взаимодействующих между собой, является то, что такой подход упрощает повседневную работу большинства членов команд (главным образом потому, что в этом случае отпадает необходимость в проведении сеансов конференц-связи для всей команды, охватывающих подчас весь земной шар). Поучаствовав в реализации проекта, который поначалу был структурирован с использованием команд, сосредоточенных в разных местах и взаимодействующих между собой, а затем реструктурирован с использованием преднамеренно рассредоточенных команд, Шарон Сайчелли (Sharon Cichelli), разработчик из города Остин, шт. Техас, пришла к выводу, что структура с использованием команд, сосредоточенных в разных местах и взаимодействующих между собой, все же лучше, чем преднамеренно рассредоточенные команды.

Разработчики, сосредоточенные в одном городе, работали над одной функциональной возможностью, аналогично обстояли дела в другом городе. Единственными, кто пострадал в этой ситуации (т.е. при использовании команд, сосредоточенных в разных местах и взаимодействующих между собой), были владельцы продукта и Scrum-мастер, которым приходилось мириться с разницей в десять с половиной часов при проведении совещаний, посвященных планированию спринтов (2008).

Хотя преимущества использования команд, сосредоточенных в разных местах и взаимодействующих между собой, совершенно очевидны, следует разобраться, в чем же заключаются преимущества использования преднамеренно рассредоточенных команд. Почему в ряде случаев целесообразнее создать две географически рассредоточенные команды вместо одной полноценно укомплектованной команды в каждом месте? Чтобы ответить на этот вопрос, попытаемся представить, какие коммуникационные проблемы могут возникнуть при рассредоточенном способе выполнения проекта. Можно указать, например, на следующие типичные проблемы.

- Разработчики, находящиеся на значительном географическом удалении от владельца продукта, не могут в достаточной мере уяснить соответствующий бизнес или область, в которой им приходится работать.
- Разработчики, находящиеся в разных городах, по-разному понимают то, что они разрабатывают.

- Разработчики, находящиеся в разных городах, принимают несовместимые между собой решения.
- Между людьми, находящимися на значительном удалении друг от друга, возникают антагонистические отношения типа “мы — это одно, а они — это совсем другое”.
- Разработчики, находящиеся в одном городе, не знают, чем в действительности занимаются разработчики, находящиеся в другом городе, и не понимают, почему они принимают именно такие, а не какие-то иные решения.

Каждая из таких проблем сама по себе способна поставить под угрозу реализацию всего проекта. К счастью, каждую из перечисленных проблем можно смягчить или сдвинуть ее возникновение менее вероятным, если соответствующий проект выполняется с использованием преднамеренно рассредоточенных команд. Один из изобретателей методологии Scrum Джeff Сазерленд (Jeff Sutherland) работал в компании Xebia, а его команды были рассредоточены между Нидерландами и Индией. В этот период они сполна воспользовались преимуществами преднамеренно рассредоточенных команд. Они пришли к следующему выводу: несмотря на то что преднамеренно рассредоточенная команда “должна, по логике вещей, создавать проблемы с коммуникацией и координированием, в действительности ежедневные совещания разработчиков помогают преодолевать культурные барьеры и сглаживать несоответствия в стилях работы, позволяя в то же время исполнителям проекта акцентировать свое внимание на заказчиках и лучше уяснить потребности клиентов” (Sutherland et al., 2008, 340).

Превосходным способом минимизировать проблемы с коммуникацией и координированием, порождаемые использованием преднамеренно рассредоточенных команд, является скрупулезный анализ состава каждой из команд. Джон Корнелл (John Cornell), директор по разработкам в компании Kofax (шт. Калифорния), работавший в свое время с членами команд во Вьетнаме, рекомендует организациям “заполнять команды людьми со связями. По мере возможности заполняйте команды людьми, которым уже приходилось работать вместе или людьми, располагающими обширными связями в своей организации. Те, у кого уже есть такие контакты, могут оказаться полезными для членов удаленных команд, которые могут не знать, к кому в организации следует обращаться по тем или иным вопросам”.

В своей книге *The Tipping Point* Малcolm Глэдуэлл (Malcolm Gladwell) называет таких людей “связными” (connectors). Связные “обеспечивают нам связь с окружающим миром... Это люди, обладающие даром наводить мосты” (2002, 38). Связные — это “люди, обладающие поистине удивительной способностью приобретать себе друзей и знакомых” (41). Связные, по утверждению Корнелла и Глэдуэлла, являются чрезвычайно полезными членами рассредоточенных команд.

Консультант и преподаватель методологии Scrum Кенни Рубин (Kenny Rubin) указывает, что корнем многих проблем, которые возникают между подгруппами, расположеными в разных местах, является недостаточная прозрачность.

Проблем, вызванных недостаточной прозрачностью, оказалось предостаточно в ходе реализации проекта, выполнение которого было рассредоточено между Нью-Йорком и Индией. Субподрядчиком выступала индийская команда. В Индии ситуация полностью вышла из-под контроля, в результате чего проект пришло

перестраивать заново. Я принимал участие в этой перестройке. Пытаясь понять, почему проект постигла неудача, я задавал те или иные вопросы. Типичный ответ звучал примерно так: “Не знаю, почему так произошло”. Или так: “Не знаю, почему им поначалу показалось, будто с этой работой могут справиться 15 человек. Сейчас им кажется, что будет достаточно 42 человек. Я не знаю, чем занимаются эти 42 человека”. Именно поэтому, когда мы перестраивали этот проект, было решено принудительно рассредоточить команды исполнителей”.

Что касается выбора между командами, сосредоточенными в разных местах и взаимодействующими между собой, и преднамеренно рассредоточенными командами, то каждый из этих двух подходов годится для определенных ситуаций. Когда такой выбор приходится делать мне, я каждый раз исхожу из того, насколько опасными, по моему мнению, являются проблемы, которые могут возникнуть в результате недостаточного уровня коммуникаций или прозрачности между командами, сосредоточенными в разных местах. Если мне кажется, что вероятность возникновения таких проблем высока и эти проблемы, возникнув, окажутся достаточно серьезными, я принимаю решение принудительно рассредоточить команды исполнителей. Например, в ситуации, когда команда рассредоточена по местам, которые объединились под крышей одной компании в результате слияния или поглощения, я всегда принимаю решение принудительно рассредоточить команды исполнителей. В подобных ситуациях всегда высока вероятность возникновения конфликта между сосредоточенными частями команды, и принудительное рассредоточение снижает риск возникновения таких полномасштабных конфликтов.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если в вашей организации используются команды, сосредоточенные в разных местах и взаимодействующие между собой, выясните, был ли такой способ рассредоточения команды выбран сознательно, выбран ли он просто “по умолчанию” или как самый легкий способ рассредоточения команды. В случае, если вы получили на свой вопрос последний ответ, подумайте над тем, не окажется ли более разумным вариантом принудительное рассредоточение команды. Можно даже испытать этот вариант в течение двух-трех спринтов.

Добейтесь спаянности команды

Мы хотим, чтобы команда проявляла единство и спаянность в своем стремлении достичь общей для всего проекта цели. Мы хотим, чтобы команда проявляла единство и спаянность в своей готовности преодолеть препятствия, с которыми приходится сталкиваться любой команде, перед которой поставлена непростая цель. Многие факторы препятствуют достижению спаянности рассредоточенной команды. Такими факторами являются, в частности, языковые и культурные различия, физическая разделенность, а также разница часовых поясов, в которых работает рассредоточенная команда. Вот почему так важно, чтобы члены рассредоточенной команды сознательно стремились к спаянности.

Учитывайте культурные различия

В своем стремлении к обеспечению спаянности команды мы должны исходить из признания того факта, что между членами рассредоточенной команды, работающими в разных географических пунктах, могут существовать значительные культурные различия. Одно из самых исчерпывающих и всесторонних исследований важных культурных различий было выполнено Геертом Хоффстеде (Geert Hofstede), который опросил сотрудников компаний IBM, работающих в более чем 50 странах. Хоффстеде выявил пять ключевых измерений, по которым наблюдаются культурные различия.

- **Индекс распределения власти (Power Distance Index — PDI).** Степень, в которой менее влиятельные члены некой культуры готовы мириться с неравномерным распределением власти.
- **Индивидуализм (Individualism — IND).** Степень, в которой исполнители предпочтуют действовать индивидуально, а не как части некоего коллектива.
- **Ориентация на достижения¹ (Achievement Orientation — ACH).** Степень, в которой соответствующая культура ориентирована на достижения, такие как доходы, видимые признаки успеха и имущество.
- **Стремление избегать неопределенности (Uncertainty Avoidance Index — UAI).** Степень, в которой соответствующая культура демонстрирует терпимость к неопределенности и неоднозначности.
- **Ориентация на долгосрочные результаты (Long-Term Orientation — LTO).** Степень, в которой соответствующая культура отдает предпочтение долгосрочным соображениям по сравнению с немедленными физическими и финансовыми выгодами.

Некоторые результаты² опросов, проведенных Геертом Хоффстеде, представлены в табл. 18.1. Несмотря на все мое недоверие к скоропалительным культурным обобщениям, я считаю целесообразным поделиться этими данными с членами команд, участвующими в том или ином рассредоточенном проекте. Эти данные можно огласить на организационном собрании, посвященном началу реализации нового проекта, или на каком-либо другом из ранних совещаний, когда члены команд знакомятся друг с другом.

Таблица 18.1. Культурные различия между репрезентативными странами.

Незаполненное поле указывает на отсутствие соответствующих данных

Страна	PDI	IND	ACH	UAI	LTO
Бразилия	69	38	49	76	65
Китай	80	20	66	30	118
Дания	18	74	16	23	

¹ В своем оригинальном исследовании 1967–1973 годов Хоффстеде обозначал это измерение как “Masculinity” (мужественность, энергичность). Даже спустя 40 лет трудно согласиться с тем, что такое обозначение было удачным. Вот почему я решил обозначить это измерение как “Achievement Orientation”.

² Конкретные величины взяты с сайта http://www.geert-hofstede.com/hofstede_dimensions.php, который включает гораздо больше стран.

Окончание табл. 18.1

Страна	PDI	IND	ACH	UAI	LTO
Финляндия	33	63	26	59	
Индия	77	48	56	40	61
Израиль	13	54	47	81	
Япония	54	46	95	92	80
Нидерланды	38	80	14	53	44
Норвегия	31	69	8	50	20
Польша	68	60	64	93	32
Россия	93	39	36	95	
Испания	57	51	42	86	
Швеция	31	71	5	29	33
Великобритания	35	89	66	35	25
Соединенные Штаты	40	91	62	46	29

Этими данными можно пользоваться, найдя строку, соответствующую вашей стране, и сравнив показатели для вашей страны с показателями для других стран, представители которых участвуют в вашем проекте. Если бы, например, я намеревался начать некий проект с членами команды в Китае, то сравнил бы строки Соединенных Штатов Америки и Китая. Я вижу, что показатель PDI у Китая намного выше, чем у Соединенных Штатов (80 против 40). Это говорит о том, что мои китайские коллеги будут гораздо менее, чем я, склонны подвергать сомнению авторитет руководителя. Если я исполняю роль реального или подразумеваемого руководителя (Scrum-мастер, старший разработчик и т.п.), то мне придется затратить дополнительные усилия, чтобы добиться такого положения дел, при котором члены команды в Китае вовлекали бы меня в открытое обсуждение.

Оценивая затем показатели индивидуализма (20 для Китая и 91 для Соединенных Штатов), я прихожу к выводу, что члены моей команды в Китае будут заинтересованы в большем единстве и сплоченности команды, чем я привык. Вряд ли они захотят, чтобы я как-то выделял отдельных исполнителей проекта, даже если это будет простая похвала.

Далее, я увидел бы, что показатели ориентации на достижения у наших стран примерно одинаковы. Большая разница в показателях ACH может указывать на различия в том, как именно члены команды определяли бы успех проекта. Рассмотрим, например, огромную разницу в ориентации на достижения между Соединенными Штатами (62) и Норвегией (8). Если бы некий проект разрабатывался в этих двух странах, то американцы могли бы считать его выполнение успешным, если бы полученный продукт пользовался финансовым успехом, даже если бы это было достигнуто за счет снижения морального духа и энергии команды, необходимых ей для выполнения следующего проекта. Однако норвежские участники того же проекта могли бы рассматривать такой результат как неудачный и предпочли бы пожертвовать финансовым успехом, если бы это оказалось единственным способом воспитать команду, которая была бы полностью готова к реализации следующей версии того же продукта или к выполнению нового проекта.

Возвращаясь к смешанной американо-китайской команде, я обратил бы внимание, что мои китайские партнеры по выполнению проекта испытывали бы, вероятно, меньший дискомфорт, имея дело с неопределенностью, чем мои американские партнеры. Знание этой особенности заставило бы меня помочь американской команде почувствовать себя увереннее при использовании Scrum, поскольку члены моей китайской команды будут спокойнее относиться к неопределенности в формулировках ролей, спецификациях продуктов, календарных планах и т.п.

Наконец, обратите внимание на огромную разницу в показателях ориентации на долгосрочные результаты (118 у Китая и 29 у Соединенных Штатов). Учитывая эту разницу, я прихожу к выводу, что регулярный видимый прогресс, выявляемый во время обзоров спринтов, окажется мощным положительным мотиватором для американской команды, но может не оказываться подобным образом на членах китайской команды, которые привносят в данный проект гораздо более широкий временной горизонт.

ПРИМЕЧАНИЕ

Любой анализ культурного контекста, подобный представленному здесь анализу Хофтеде, неминуемо ведет к обобщениям. Несмотря на то что такие обобщения могут оказаться верными по отношению к населению в целом, не следует забывать о том, что каждый человек уникален и именно таковым его и следует воспринимать. Обобщения могут давать нам некое первоначальное представление о той или иной культуре, но личный опыт — это наилучший способ уяснить особенности каждого из членов команды.

Учитывайте незначительные культурные различия

Исследование, выполненное Хофтеде, касалось значительных культурных различий. Помимо значительных культурных различий, проекты, предполагающие географически рассредоточенную разработку, сталкиваются с множеством довольно незначительных, но от этого не менее важных культурных различий. Например, вряд ли кто-то удивится, когда ему скажут, что в разных странах, религиях и культурах отмечаются разные праздники. Многие рассредоточенные команды решают эту проблему, создавая на веб-сайте или вики-доске своего проекта специальную страницу с перечнем совокупности всех праздников, отмечаемых членами соответствующей команды. Некоторые из команд, с которыми мне приходилось работать, тратят в ходе ежедневного совещания разработчиков накануне того или иного праздника несколько минут дополнительного времени на то, чтобы получить от местной команды разъяснения относительно цели и традиций данного праздника. Именно так я узнал о существовании индийского праздника Ганди Джайанти, канадского Дня королевы Виктории, норвежского Дня конституции и др. Само по себе знание того, что многие из моих индийских коллег воздерживаются от употребления мяса и алкоголя во время празднования Ганди Джайанти, или того, что мои канадские коллеги разъезжаются в День Виктории по своим коттеджам на берегах озер, конечно же, не делает меня выдающимся работником. Но и те, и другие относятся ко мне с большим уважением, когда видят, что мне вовсе небезразлично, что и почему они празднуют.

Несмотря на то что наличие разных праздников у представителей разных стран ни у кого не вызывает удивления, я был немало удивлен, когда узнал, что рабочая

неделя с понедельника по пятницу включительно не является общепринятым правилом. Прожив практически всю свою жизнь в Соединенных Штатах Америки, где рабочая неделя с понедельника по пятницу включительно является правилом почти для всех офисных профессий, я просто не мог представить себе существования какого-то другого варианта рабочей недели. “Разумеется, все в мире работают каждую неделю с понедельника по пятницу включительно,” — ошибочно полагал я. Так я продолжал считать до тех пор, пока мне не пришлось поработать с командами в Израиле и Египте. И теперь я знаю, что в Израиле и Египте, а также в других странах Ближнего Востока типичная рабочая неделя длится с воскресенья по четверг.

Еще одно культурное различие, с которым мне пришлось столкнуться, касается того, как люди проводят свои вечера. В Соединенных Штатах обычный рабочий день заканчивается в 17:00 или 18:00. Вернувшись домой с работы, люди ужинают (примерно с 18:00 до 19:00). Я полагал, что такой распорядок (возвращение с работы—ужин—отдых вместе с семьей—сон) практически универсален. Если мне нужно позвонить вечером кому-то из членов своей команды, то я предпочитаю делать это примерно в 21:00. К тому времени мои дети уже отправились спать и мне ничего не стоит потратить около 15 минут на звонки своим коллегам. Однако в Индии большинство семей садятся ужинать именно в 21:00 и нет ничего хуже, чем звонить им в это время. К сожалению, я узнал об этом слишком поздно. Я работал над парой проектов в Бангалоре и Хайдарабаде и полагал, что 21:00 — самое подходящее время для вечерних звонков моим индийским коллегам. Прошло несколько недель, прежде чем кто-то из них осмелился предложить мне выбрать для вечерних звонков более подходящее время, поскольку я тревожу своими звонками людей в самый разгар ужина. Одним словом, я старался выбрать самое подходящее время, а выбрал самое неподходящее.

Визиты знакомства и путешествующие посланники (и те, и другие обсуждаются ниже в этой главе) — превосходные способы ознакомиться с местными традициями и предпочтениями команды, сосредоточенной в данном месте, и довести эту информацию до сведения команды, сосредоточенной в другом месте.

Укрепляйте функциональную и командную субкультуры

Как бы сильны ни были наши национальные культуры, вполне может оказаться, что субкультура разработки программного обеспечения еще сильнее. Вот что пишет об этом явлении профессор Эрран Кармел (Erran Carmel), который многие годы занимался изучением деятельности рассредоточенных команд.

Специалисты по программному обеспечению, какой бы национальности они ни были, принадлежат к компьютерной субкультуре. Гуру в области программного обеспечения Ларри Константайн (Larry Constantine) утверждает, что компьютерная субкультура сильнее любой национальной культуры и что программист из Москвы больше похож на своих американских коллег, чем на других русских. Инженеры, подобно специалистам по программному обеспечению, высоко ценят личные достижения и относительно низко — общественные отношения. В стереотипе антисоциального программиста заложено определенное зерно истины (1998, 73–74).

Исследования подтверждают силу компьютерной субкультуры (Carmel, 1998, 74). Эффективные команды используют функциональную и командную субкультуры для повышения спаянности команд. Мы хотим, чтобы люди рассуждали так: “Я — член

команды, работающей над проектом Orion”, а не так: “Я — член индийской команды, работающей над проектом Orion”. Разница между этими двумя вариантами может показаться несущественной, но я считаю, что подразумеваемая ими небольшая перемена в образе мышления весьма важна.

Полезными могут оказаться многие методы привязки людей к их функциональной (“Я — программист/тестер/администратор базы данных”) или командной субкультуре, особенно когда речь идет о рассредоточенной команде. Например, корпоративная среда, которая стимулирует формирование сообществ практических пользователей Scrum, важна как для команд, сосредоточенных в одном месте, так и для рассредоточенных команд. Существует, однако, несколько методов, которые особенно важны для укрепления функциональной и командной субкультур в рассредоточенных командах.

Выработайте и доведите до каждого члена команды общее видение проекта

См. также С методами формирования представлений, общих для всех участников проекта, можно ознакомиться в разделе “Подзаряжайте систему энергией” главы 12, “Управление самоорганизующимся коллективом”.

В отсутствие представлений, единых для всей команды, будет практически невозможно развить сильную командную культуру. Это делает выработку таких общих представлений особенно важной задачей для рассредоточенной команды. Элейни Тьеррен (Elaine Thierren) из компании FirstAmerican CoreLogic (Санта-Фе, шт. Калифорния) входила в состав рассредоточенной команды, одна часть которой была дислоцирована в ее офисе, а другая часть — в Бангалоре. По завершении проекта Элейни Тьеррен проанализировала ход его выполнения и пришла к выводу, что исполнители проекта по-разному представляли себе продукт, которые они разрабатывали.

Отсутствие единых представлений о продукте (иначе говоря, отсутствие у команды единой “дорожной карты”) имело два важных последствия. Во-первых, продукт часто требовал переделок, когда заказчик запрашивал включение новых функциональных возможностей. Иными словами, у исполнителей проекта отсутствовало единое представление о направлении разработки продукта, которое позволяло бы проектировать приложение таким образом, чтобы в будущем в него можно было легко включать новые функциональные возможности. Во-вторых, у команды, работающей в Индии, возникло ощущение, что впереди у них уже не осталось серьезной работы, и, следовательно, вскоре многие из них могут оказаться не у дел (2008, 369).

По мнению Элейни Тьеррен, наличие у команды единых представлений важно с точки зрения обеспечения спаянности команды. Как правило, ответственность за наличие у команды единых представлений возлагается на владельца продукта. Марку Саммерсу (Mark Summers), наставнику по гибкой методологии разработки в EMC Consulting (Лондон), повезло работать с владельцем продукта, который понимал свою роль в формировании у всех исполнителей проекта единых представлений о разрабатываемом продукте.

В начале работы над каждой версией продукта наш владелец продукта выезжал в Индию, чтобы ознакомить местные команды с ее особенностями. Таким образом, очень важно, чтобы владелец продукта заранее доводил до всех исполнителей проекта единые представления о разрабатываемом продукте. Из своего предыдущего опыта мы знали, как плохо, когда зарубежные исполнители проекта не понимают, что именно от них требуется в тех или иных случаях. Заблаговременное их ознакомление с этими требованиями всегда оказывало существенную помощь в успешной реализации проекта (2008, 338).

Достигайте соглашений

Командная культура частично проистекает из соглашений, которые члены команды заключают друг с другом. Одни из таких соглашений носят явный характер, например “своевременно являться на ежедневные совещания разработчиков” или “не разрушать сборку”. Другие соглашения могут носить неявный характер, например “без крайней необходимости не рассылайте лишние копии электронной почты всем подряд”. Для рассредоточенных команд желательно, чтобы как можно большее число таких соглашений носило явный характер.

Например, рассредоточенные Scrum-команды зачастую вырабатывают соглашение относительно приемлемого времени ответа на сообщения электронной почты (например, на сообщения электронной почты нужно отвечать в течение одного рабочего дня, даже если ответ звучит как “Я работаю над этой проблемой. С результатом ознакомлю Вас завтра”). Несмотря на то что подобное соглашение может быть достигнуто и в команде, сосредоточенной в одном месте, с большей степенью вероятности оно может представлять собой неявно выраженное ожидание, чем явно выраженное правило поведения членов команды. Если я не тороплюсь отвечать на сообщение электронной почты и все мы работаем в одной комнате, то вы, наверное, откинетесь на спинку кресла и громко крикнете: “Эй, Майк, ты не ответил на мое вчерашнее письмо по поводу того, следует ли нам использовать Groovy”. При этом не только я один услышу, что должен поскорее ответить на ваш вопрос. Об этом услышат все присутствующие в комнате. Это будет служить им напоминанием о том, что *на сообщения, полученные по электронной почте, отвечать нужно максимально оперативно*.

Помимо соглашений относительно оперативной реакции на сообщения, полученные по электронной почте, многие рассредоточенные команды вырабатывают явные соглашения по поводу того, когда исполнители проекта должны находиться на своем рабочем месте, когда к ним можно обращаться в нерабочее время, как быстро должны начинаться совещания, какой тип связи (телефон, электронная почта, мгновенный обмен сообщениями и т.п.) лучше всего подходит для того или иного типа обсуждения, какой тип вопросов нужно обсуждать всей команде, а не узкому кругу специалистов, и т.п.

Один вопрос рассредоточенная команда должна согласовать обязательно: как именно она будет применять гибкую методологию разработки. Вовсе необязательно, чтобы каждая команда в каждом офисе применяла гибкую методологию разработки точно так же, как все остальные команды, однако некоторые базовые вещи должны быть одинаковыми для всех команд. Поскольку Scrum — это лишь концептуальный каркас управления проектами, как именно будет реализован этот концептуальный каркас на практике, зависит от каждой конкретной команды. В разных местах Scrum может

реализоваться по-разному. Некоторые из этих различий полезны: одни из них ведут к улучшениям, которые могут использоваться во всех местах дислокации отдельных частей рассредоточенной команды; другие представляют собой адаптации, необходимые для использования Scrum в конкретном месте. Остальные различия могут приводить к несовместимости и должны быть устраниены. Джейн Робартс (Jane Robarts), руководитель проекта в ThoughtWorks, столкнулась с подобной ситуацией, будучи участником проекта, рассредоточенного между Соединенными Штатами Америки и Индией.

Мы исходили из того, что команда, дислоцированная в Индии, применяет гибкую методологию разработки так же, как мы собирались применять ее в Соединенных Штатах. Однако вскоре после того, как мы приступили к первой итерации, стало ясно, что у каждой из групп исполнителей, участвующих в реализации нашего проекта, есть своя версия “гибкой методологии разработки” (2008, 331).

Хотя командам, участвующим в реализации одного и того же проекта, но сосредоточенным в разных местах, важно прийти к согласию относительно некой общей для всех них совокупности методов применения Scrum, это не означает, что рассредоточенные команды должны применять Scrum строго “по книге”. Новая Scrum-команда должна начинать свою работу именно так. Со временем ей придется добавить и адаптировать методы, специфические для соответствующего проекта, что будет способствовать повышению эффективности применения Scrum. По сути, сравнивая успешные и неудачные рассредоточенные Scrum-реализации в Yahoo!, наставники Брайан Драммонд (Brian Drummond) и Дж. Ф. Ансон (J. F. Unson) пришли к выводу, что применение Scrum в рассредоточенных командах строго “по книге” является источником серьезных проблем.

В отсутствие достаточного руководства команды склонялись к применению Scrum строго “по книге”. Результаты в таких случаях оказывались настолько плачевными, что люди были готовы вообще отказаться от использования Scrum и гибкой методологии разработки. Эти команды не пытались приспособить приемы гибкой методологии разработки к трудностям, обусловленным специфическими требованиями рассредоточенной разработки (2008, 320).

Как и в случае с другими типами рабочих соглашений, в данной ситуации решение заключается в том, чтобы Scrum-мастер, приняв участие в одном или более совещаниях с членами рассредоточенной команды, помог им выработать соглашения в отношении тех аспектов Scrum, которые они хотели бы сделать единообразными для всех мест дислокации их команды.

Жизненно важным для формирования спаянной команды является создание в команде атмосферы взаимного доверия. Особенно трудно создать атмосферу взаимного доверия среди членов рассредоточенной команды. Дебора Даурте (Deborah Duarte) и Нэнси Снайдер (Nancy Snyder) в своей книге *Mastering Virtual Teams* показывают, как иногда трудно доверять людям, которых мы и в глаза-то никогда не видели.

Когда у нас есть возможность лично общаться с людьми, ряд ключевых признаков помогает нам определить, кому можно, а кому нельзя доверять. У нас есть возможность оценивать невербальные аспекты при общении людей и наблюдать их взаимодействие с другими членами команды. Можно частично судить о надежности другого человека в процессе личного общения с ним, опираясь на чувственное его восприятие (2006, 85).

Невозможность полагаться на повторяющееся, частое личное общение заставляет рассредоточенные команды принимать другие меры для налаживания взаимного доверия. Путешествующие посланники, начало совещаний с непринужденных разговоров “о том, о сем”, периодические собрания команды в полном составе, рабочие соглашения и другие подобные меры — все это способствует налаживанию взаимного доверия между членами рассредоточенной команды. Неплохо также, если с самого начала выполнения проекта команду категорически обязывают создавать работоспособное программное обеспечение к концу каждого спринта (буквально с первых же спринтов). К сожалению, при реализации многих проектов уже на самых ранних стадиях выполнения проекта слишком много времени отводится дискуссиям и упражнениям, связанным с построением команды. Как свидетельствует исследование, проведенное профессором Линдой Граттон (Lynda Gratton) и ее коллегами (результаты данного исследования опубликованы в *MIT Sloan Management Review*), это очень распространенная и опасная ошибка.

Для эффективного управления рассредоточенными командами требуется применение ряда континтуитивных управленческих подходов. В частности, уже на ранних стадиях выполнения проекта лидеры команды должны сосредоточиться на задачах, а не на межличностных отношениях, а затем, когда наступит подходящий момент, переключиться на выстраивание отношений (Gratton, Voigt, and Erickson, 2007, 22).

Хотя в этом исследовании предполагается, что с самого начала выполнения проекта нужно сосредоточиться на задачах, из этого не следует, что владельцы продукта или Scrum-мастера должны распределить задачи между разработчиками. Напротив, я считаю, что эти лидеры должны подчеркивать, насколько команде важно добиваться ощутимого прогресса уже на ранних спринтах. Проблема с ранним акцентом на выстраивании отношений заключается в том, что такой акцент провоцирует образование подгрупп, весьма далеких от идеальных. Любая крупная группа неизбежно распадается на подгруппы. Если допустить слишком раннее образование таких подгрупп, то они сформируются на основе атрибутов поверхностного уровня — американцы, шведы, программисты C++, Java-программисты, женщины-конструкторы баз данных, программисты мужского пола и т.п. Граттон и ее коллеги пишут по этому поводу: “Попросту говоря, чем больше люди взаимодействуют друг с другом в самом начала работы команды, тем выше вероятность, что они будут выносить скоропалительные суждения и подчеркивать различия, существующие между ними” (26).

Выстраивание отношений нужно отложить до того момента, когда члены команды узнают друг о друге нечто более существенное, например степень профессионализма и уровень знаний каждого, подходы к работе и т.п. Этого можно добиться путем как можно более раннего акцента на прогрессе в работе, а не на выстраивании отношений в команде. Подгруппы, которые формируются в это время, будут основываться на взаимной потребности в совместной работе над созданием соответствующего продукта. Чтобы реализовать конкретную пользовательскую историю из журнала запросов на выполнение работ, вам и мне нужно работать вместе. В процессе работы мы определим степень профессионализма и уровень знаний каждого из нас. Например, я узнаю, что вы не просто Java-программист, а Java-программист, хорошо разбирающийся в автоматизированном тестировании исходного кода. Вы же узнаете, что я не просто администратор баз данных, но и неплохой специалист по оптимизации запросов SQL.

Команды с подгруппами, сформированными на основе совместимых профессий, позиций, подходов к работе и тому подобного, склонны к утрате взаимного доверия в меньшей степени, чем подгруппы, сформированные на основе поверхностных атрибутов (например, американец, швед, программист, тестер и т.д.). Вспомните о “проблемных” командах, членом которых приходилось бывать вам лично. Вполне вероятно, что конфликты в этих командах обусловливались их ярко выраженной природой “мы и они”, основанной на поверхностных атрибутах того или иного рода: “наш офис и их офис”, “программист и администратор баз данных”, “приверженцы Linux и приверженцы Windows”. Когда команды ощущают непосредственную потребность в достижении заметного прогресса, им просто не хватает времени для образования подгрупп такого рода.

После того как команда поработает вместе в течение нескольких спринтов, сместите акцент на выстраивание отношений в команде, задействовав в большей мере социальную деятельность и совместные периоды вынужденногоостоя во время спринтов. Команде требуется достаточный объем опыта совместной работы, прежде чем смогут принести пользу социальная деятельность и выстраивание отношений. Но когда это будет достигнуто, “налаживание в команде взаимного доверия и создание возможностей для общения на этом этапе помогают в выработке новых способностей и обеспечивает рост команды” (Gratton, Voigt, and Erickson, 2007, 29).

Хочу быть правильно понятым: я вовсе не утверждаю, что в начале выполнения проекта не должно быть предусмотрено время для общения между членами команды. Визиты знакомства и общие собрания команды в начале выполнения проекта для планирования первоначального выпуска могут оказаться весьма полезными. Важны три здесь являются три момента. Во-первых, проект должен начинаться как можно энергичнее, с акцентом на ранних демонстрациях достигнутого прогресса. Во-вторых, весь “бюджет” для общения не должен быть растратчен в течение двух первых спринтов. В-третьих, ранняя социальная деятельность должна быть максимально тесно связана с работой по выполнению проекта (например, проведение общего собрания команды, посвященного планированию выпуска).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Определите по табл. 18.1 местонахождение каждой из групп разработчиков, принимающих участие в реализации вашего проекта. В ходе ретроспективы очередного спрингта обсудите различия, выявленные вами с помощью табл. 18.1. Кажутся ли вам эти различия существенными? Какие проблемы эти различия могут породить в будущем?
- ❑ В ходе будущих совещаний, посвященных планированию спрингта, обсудите любые культурные или национальные торжества, которые придется на этот спрингт в местах, где сосредоточены группы разработчиков, принимающих участие в реализации вашего проекта. Недостаточно просто упомянуть о том или ином празднике или торжестве. Попросите людей, которые будут отмечать эти праздники, поподробнее рассказать другим членам команды об отмечаемом празднике (как они проводят этот день, мифы и легенды, связанные с отмечаемым событием, особые ритуалы, связанные с этим событием, и т.п.).
- ❑ В ходе ретроспективы очередного спрингта обсудите и задокументируйте рабочие соглашения своей команды. Каким должно быть поведение членов вашей команды?

Встречайтесь чаще

Все рассредоточенные команды, с которыми мне приходилось работать, в один голос заявляют о несомненной полезности периодического проведения общих собраний команды. Разные команды организовывают и проводят такие собрания по-разному: одни проводят в полном составе (и в одном и том же месте) несколько первых спринтов, а другие заранее планируют периодическое проведение общих собраний. В некоторых командах проводится ротация членов между местами дислокации разных групп разработчиков, принимающих участие в реализации проекта. Большинство команд использует то или иное сочетание указанных методов. В данном разделе мы рассмотрим каждый из таких методов по отдельности.

Визиты знакомства

Одним из самых популярных подходов к организации общения как можно большего числа членов рассредоточенной команды является метод, с легкой руки Мартина Фаулера (Martin Fowler) получивший название “визит знакомства”. Мартин Фаулер утверждает, что такие визиты “должны организовываться уже на ранних стадиях выполнения проекта, а их целью является формирование отношений” (2006). Визит знакомства, который представляет собой собрание всех участников проекта в самом начале его реализации, может оказаться одной из самых эффективных инвестиций в успешное завершение этого проекта. Это может быть особенно важным для проектов, исполнители которых незнакомы друг с другом. Это важно также для членов рассредоточенной команды, у которых практически отсутствует совместная история, которые говорят на разных языках и принадлежат разным культурам. Предоставляя возможность рассредоточенной команде побывать непродолжительное время вместе, мы позволяем ей сделать несколько важных первых шагов к тому, чтобы как можно лучше познакомиться друг с другом и выработать взаимное доверие. Продолжительность такого визита может составлять от нескольких дней до недели. Правда, многие команды находят весьма целесообразным полностью совместить визит знакомства с выполнением первого спрингта. Именно такой подход использовался в ходе реализации одного проекта, рассредоточенного между Гонконгом и Китаем. Вот как описывает этот подход Джейн Робартс.

Это общее собрание всех членов команды, проведенное в самом начале реализации проекта, дало нам возможность познакомиться друг с другом, наладить хорошие отношения и выработать единое понимание проекта. К тому времени, когда люди разъехались по местам своей постоянной дислокации, они уже успели познакомиться друг с другом, не стесняясь звонили друг другу по телефону и ощущали себя членами единой команды (2008, 328).

Эди Миллер (Ade Miller), сотрудник группы моделей и методов в компании Microsoft, рекомендует “использовать эти периоды работы в одном и том же месте для формирования не только совместного понимания соответствующей проблемной области, но и продуктивных рабочих отношений внутри команды” (2008, 18).

ВОЗРАЖЕНИЕ

“Обеспечивать доставку в одно и то же место членов рассредоточенной команды со всего мира, поселять их в гостиницы... Не слишком ли дорогое удовольствие? Мы используем рассредоточенную команду с целью экономии средств. Организовывая общее собрание такой команды, мы лишаем смысла само ее создание”.

Отказываясь собирать всех (или хотя бы некоторых) членов рассредоточенной команды в одном и том же месте, мы расписываемся в своем стремлении к ложной экономии: да, это сэкономит нам деньги сегодня, но будет стоить дополнительных затрат во все оставшееся время выполнения проекта. Никогда не планируйте экономию затрат аутсорсингового проекта, учитывая лишь суммарную стоимость рабочей силы (т.е. всех членов команды) в расчете на один час. Использование рассредоточенной команды сопряжено со многими видами скрытых издержек. Повышенный бюджет командировочных расходов должен быть частью любого рассредоточенного проекта. Эрран Кармел (Erran Carmel) в своей книге *Global Software Teams* (“Глобальные команды разработчиков программного обеспечения”) утверждает: “Перелеты с места на место — весьма дорогостоящий, но совершенно необходимый атрибут деятельности глобальных команд” (1998, 157). С этим мнением согласны Том Демарко (Tom DeMarco), Тим Листер (Tim Lister) и другие сотрудники консалтинговой организации Atlantic Systems Guild: “Чтобы рассчитывать на успех в рассредоточенной разработке, вам почти наверняка придется наращивать, а не сокращать свой бюджет командировочных расходов” (2008, 42).

Если позволяет бюджет, продолжительность визита знакомства может даже превышать продолжительность одного спринта. В самом начале проекта трудоемкостью в 20 человеко-лет компания Xebia решила собрать вместе голландских и индийских разработчиков на срок выполнения пяти двухнедельных спринтов. Джейфф Сазерленд, консультант по данному проекту, и соавторы из Xebia описывают преимущества использования такого подхода.

В ходе совместных итераций члены команды наладили личные взаимоотношения, которые продолжались до самого завершения проекта, а у членов индийской команды выработалось хорошее понимание клиентского контекста. Кроме того, у всех участников проекта выработались общие представления об используемых методах, стандартах, инструментарии и естественных ролях каждого члена команды (2008, 341).

Несмотря на то что проведение общего собрания всех членов команды в самом начале выполнения проекта представляется вполне логичным и полезным шагом, подчас осуществить его на практике не представляется возможным. В таком случае общее собрание всех членов команды следует провести тогда, когда такая возможность появится. Иными словами, такое собрание вовсе необязательно проводить в самом начале выполнения проекта. Подходящим временем для проведения такого собрания может быть любой момент, когда это представляется возможным и удобным. Эди Миллер из Microsoft указывает, что “личному, непосредственному общению нет полноценной замены; особенно это касается кульминационных моментов выполнения проекта” (2008, 10).

Один из самых эффективных способов использования визитов знакомства, с которыми мне приходилось встречаться, практиковался в компании Oticon, старейшем в мире изготовителе устройств для людей с ослабленным слухом. Головной офис разработчиков компании Oticon находится в городе Сморум, что неподалеку от Копенгагена (Дания). В июне 2007 года было принято решение принять на работу значительное число новых разработчиков в Польше. К тому времени у компании Oticon уже было собственное представительство в Польше, но оно не использовалось для разработки программного обеспечения. Польша была выбрана потому, что стало трудно найти подходящих разработчиков в Дании. Иначе говоря, экономия денег не была главной причиной найма польских разработчиков.

Руководство Oticon понимало, что объединение польских и датских разработчиков станет серьезной проблемой. Оле Андерсен (Ole Andersen), один из менеджеров Oticon, отвечал за прием на работу польских разработчиков и за их объединение с его командой в Дании. Он описывает, как ему удалось успешно их объединить.

Мы решили, что каждый из польских разработчиков должен приехать в Данию и в течение двух месяцев поработать в одной из наших Scrum-команд. Мы наняли неподалеку от нашего офиса уютную квартируку, чтобы каждый из наших гостей мог с комфортом провести эти два месяца. Кто-то из них воспользовался возможностью приехать в Данию вместе со своими семьями и пожить там с ними недельку-другую. В течение своего двухмесячного пребывания в Дании польские разработчики имели возможность получше познакомиться с компанией, причем в качестве гидов выступали члены Scrum-команды, в которой они временно работали. Их работа в Scrum-команде ничем не отличалась от работы их датских коллег. От датских разработчиков мы получили весьма позитивные отзывы о таком способе организации процесса ознакомления. Кое-кто из датских разработчиков даже успел подружиться со своими польскими коллегами. Польские разработчики были радушно приняты в Дании, а их впечатления от пребывания в этой стране оказались самыми благоприятными. В немалой степени этому способствовало то, что они проживали в отдельной квартире, а не в гостинице.

Визиты с целью поддержания контактов

После того как визит знакомства (в идеальном случае — в самом начале выполнения проекта, хотя это и необязательно) даст возможность установить некий начальный уровень отношений между членами рассредоточенной команды, последующие визиты (так называемые “контакт-визиты”) используются с целью поддержания этих отношений. Как и в случае визита знакомства, каждый контакт-визит должен быть посвящен решению определенной задачи (планирование, выработка решения той или иной проблемы и т.п.), но, как говорит Фаулер, “нужно помнить, что главная цель визита заключается не в том, чтобы выполнить ту или иную задачу, а в том, чтобы выстроить продуктивные рабочие отношения”. Он рекомендует наносить однодневные визиты не реже, чем один раз в два месяца (2006).

Некоторые команды считают, что ежеквартальное планирование выпусков является самым подходящим временем для проведения общего собрания команды. Рассмотрим случай продукта, каждый новый выпуск которого планируется примерно раз в три месяца на протяжении всего жизненного цикла этого продукта. Рассмотрим

крупный продукт, календарный план которого рассчитан на год или даже больше. В обоих случаях квартальный цикл может начинаться с выступления владельца продукта перед командой. В своем выступлении владелец продукта излагает свое видение перспектив данного продукта. Временное объединение команды под одной крышей облегчает решение этой задачи.

Кроме того, не следует забывать, что, как правило, в момент начала очередного релиз-цикла завершается предыдущий цикл (это относится к обоим рассмотренным нами случаям). Этот момент является идеальным для проведения общего собрания команды. Ограничившись расходами и заботами, связанными лишь с одной командировкой (каждого из членов команды), команду можно собрать в одном месте для рассмотрения перспектив соответствующего продукта и планирования нового выпуска и спринта, а также для последнего спринта предыдущего выпуска, включая его обзор и ретроспективу. Миллер из компании Microsoft рекомендует пойти еще дальше.

Объединение команды на последние пару итераций перед последним выпуском позволяет сделать более гладким процесс передачи заказчику последней совокупности функциональных возможностей. Оно помогает команде сосредоточиться на доведении продукта до такой степени готовности, когда его можно будет передать заказчику. Объединение команды под одной крышей означает, что в момент принятия ключевых решений под рукой окажутся все участники проекта (2008, 11).

Путешествующие посланники

Типичная практика управления путем обхода (Management By Walking Around — MBWA) заменяется в случае рассредоточенных проектов практикой управления путем облета (Management By Flying Around — MBFA). Разумеется, в случае рассредоточенных проектов требуется нечто большее, чем просто менеджеры, перелетающие из одного города в другой или из одной страны в другую. Во многих отношениях людей, перелетающих из одного города в другой, можно рассматривать как посланников. Мартин Фаулер определяет посланников как “полупостоянных людей, которые проводят по несколько месяцев в году в “другом” месте”. Хотя проводить по несколько месяцев в году в другом месте может быть идеальным вариантом, я пришел к выводу, что зачастую приходится идти на компромисс, отправляя посланников в командировки чаще, но с менее продолжительными визитами.

Я не очень-то знаю, как работают настоящие послы. Поэтому обычно я представляю себе этих “посланников проекта” в духе известной песни “The Real Ambassador” (“Настоящий посол”) знаменитого джазового певца Луи Армстронга (Louis Armstrong), написанной в 1961 году. В этой песне он поет о самом себе. Он рассказывает послам-политикам, что, играя роль джазового исполнителя, путешествующего по всему миру, он вместе с тем выступает в роли “настоящего посланника” своей страны. Правда, говорит он, “вся моя работа сводится лишь к исполнению блюзов и личным встречам с людьми”.³ Это заявление Армстронга идеально соответствует задачам, которые должны решать “посланники проекта”: писать код и лично встре-

³ Если вы хотите услышать эту песню в исполнении Луи Армстронга и узнать о его личной деятельности, а также о деятельности его коллеги Дэйва Брубека (Dave Brubeck) в качестве “настоящих посланников”, обратитесь по адресу <http://www.therealambassadors.com/2.htm>.

чаться с людьми. Мартин Фаулер подчеркивает важность неформального аспекта деятельности посланников.

Важной составляющей деятельности посланника является общение с людьми. При выполнении любого проекта требуется очень много неформального общения. Пусть не все это общение является таким уж важным, т.е. имеет непосредственное отношение к делу, частично оно все же имеет огромное значение. Проблема в том, что вы не всегда можете распознать, какая именно часть этого общения является важной, а какая — нет. Поэтому частично задача посланника заключается в изложении своим собеседникам множества мелочей и подробностей, которые не представляются достаточно важными для передачи посредством более формальных каналов коммуникации. Разумеется, я имею в виду те мелочи и подробности, которые помогают нам лучше понять своих сотрудников (“Фернандо сказал, что вчера его малыш сделал пять первых шагов в своей жизни”), а не распространение злонамеренных сплетен (2006).

Я пришел к выводу, что личные отношения, установленные посланниками, могут быть чрезвычайно цennыми даже спустя долгое время после того, как посланник вернется в свою родную страну. Бен Хоган (Ben Hogan) участвовал в проекте, распределенном между Бангкоком и Сиднеем. Он комментирует: “Мы пришли к выводу, что обмен посланниками между городами, где были сосредоточены группы наших разработчиков, является одним из наиболее эффективных методов налаживания межкомандного общения. Применение этого метода позволило нам наладить личные отношения и послужило механизмом выстраивания доверия и обмена знаниями. Посланникам удавалось делиться с остальными членами рассредоточенной команды полученными ими уроками, а также задавать дальнейшее направление разработки проекта” (2006, 322).

При выполнении одного из проектов, в котором я исполнял роль наставника, разработчики были рассредоточены между Денвером и Торонто. Команды в этих двух городах работали над одним проектом, что было обусловлено поглощением одной компании другой. Поначалу это привело к возникновению недружественных отношений между двумя командами. Фрэнк (программист, работавший в Денвере) вызвался совершить два двухнедельных визита в Торонто. Я был хорошо знаком с разработчиками из Торонто, поскольку ранее работал с ними в течение двух лет. Я хотел, чтобы поездки Фрэнка в Торонто принесли нам как можно большую пользу, поэтому я поговорил с ним об его увлечениях и интересах, не связанных с работой. Узнав, что он увлекается скалолазанием, я позвонил работавшему в Торонто Марселию, который также до самозабвения увлекался скалолазанием. Я попросил Марселя сделать мне одолжение, уделив немного времени Фрэнку. В частности, я предложил Марселя сходить вместе с Фрэнком на тренировочную базу для скалолазов, где Марсель проводил едва ли не все свое свободное время. Марсель охотно откликнулся на это предложение. В результате Марсель подружился с Фрэнком, к тому же у них обнаружились и другие общие интересы.

Дружеские отношения, установившиеся между Марселем и Фрэнком, с самого начала сослужили нашему проекту хорошую службу. Но самые большие дивиденды эта дружба принесла несколько месяцев спустя, когда между отделами на периферии нашего проекта, охватившего два города, начал разгораться конфликт. Сотрудники

отдела информационных технологий в Денвере решили присвоить название “Пандора” серверу, которым должна была пользоваться команда из Торонто. Те не на шутку обиделись из-за такого названия, решив, что оно было выбрано для того, чтобы их оскорбить, имея, очевидно, в виду древнегреческий миф о ящике Пандоры, содержащем все несчастья человечества. Я находился в Торонто, когда этот скандал начал разгораться, и попросил Марселя позвонить Фрэнку и попросить его осторожно разузнать, не было ли такое название выбрано именно с целью оскорбить торонтскую команду. Двумя часами позже Фрэнк доложил нам, что сотрудник, предложивший такое название, даже не подозревал, что оно означает. Разумеется, он немедленно отказался от этого варианта названия. Таким образом, доверительные отношения, установившиеся между Марселем и Фрэнком, помогли нам быстро уладить этот конфликт.

Джейн Робартс также пришла к выводу, что польза от визитов, наносимых посланниками, не ограничивается лишь достижением формальных целей, связанных с такими визитами.

На протяжении всего выпуска мы планировали визиты наших американских продукт-менеджеров и ведущих сотрудников команды в индийский офис. Идея таких визитов заключалась в том, чтобы облегчить передачу знаний предметной области бизнес-аналитикам, разработчикам и аналитикам качества в Индии. Замечательным побочным продуктом этих визитов было то, что посетители индийского офиса неизменно возвращались домой с более глубоким пониманием обстановки. После посещения индийского офиса они нередко начинали планировать телефонные звонки в более подходящее для их индийских коллег время, изыскивать способы ограничения количества телефонных звонков и, что самое важное, изменять сам тон своего общения с индийскими коллегами. Повышалась их осведомленность об усердии, с которым работала индийская команда. Они понимали, на какие личные жертвы приходится идти каждому исполнителю во имя успешного завершения проекта. Члены индийской команды, со своей стороны, стали лучше понимать американскую команду, лично познакомившись со многими ее членами (2008, 328).

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Купите билет на самолет. Очень важно время от времени организовывать личное общение. Если после вашего последнего визита в одно или несколько мест сосредоточения исполнителей проекта уже прошло какое-то время, запланируйте новый визит.
- ❑ Если выполнение вашего проекта началось совсем недавно, назначьте посланников и попросите их спланировать свои первоначальные визиты.

Измените способ своего общения

Посланники, о которых упоминает Робартс, указывают, что одним из самых серьезных последствий рассредоточения членов команды становится изменение способа их общения. Члены Scrum-команды, сосредоточенной в одном месте, все время общаются между собой. Но одно дело, когда вы встаете со своего кресла, делаете несколько шагов по направлению к своему коллеге и обращаетесь к нему со словами:

“Эй, Крис, что тебе известно об этом алгоритме шифрования?”, и совсем другое дело, если для того, чтобы задать тот же вопрос, вам нужно набрать телефонный номер и позвонить Крису, который находится от вас на расстоянии нескольких часовых поясов. И уж совсем другое дело, если для того, чтобы задать тот же вопрос, вам нужно отправить Крису сообщение по электронной почте и ждать ответа по меньшей мере до завтра.

Объем документации растет

С этим ничего не поделаешь: рассредоточенной команде придется заниматься ведением документации больше, чем команде, сосредоточенной в одном месте. Возможно, придется больше полагаться на письменные отчеты о состоянии выполнения проекта, которые будут использоваться в качестве вспомогательных материалов к обзорам спринтов теми сотрудниками, которые не смогут лично присутствовать на этих обзорах. Предполагаемые варианты проектных решений могут составляться в письменном виде и рассыпаться членам рассредоточенной команды (особенно это касается членов команды, у которых из-за разницы в часовых поясах рабочее время практически не “пересекается”). Разговоры, которые обычно ведутся в коридоре или холле, придется заменить общением по электронной почте. Таким образом, увеличение объема писаницы становится неизбежным.

К счастью, наращивание объемов письменного общения вовсе не обязательно означает невозможность использования гибкой методологии разработки проектов, но членам команды необходимо помнить о том, что письменное общение нередко приводит к всевозможным недоразумениям. Я лично участвовал в нескольких проектах, когда члены команды, сосредоточенные в разных городах, переставали доверять друг другу. Обычно это были команды, которые сформировались в результате слияния/поглощения двух компаний. Из-за взаимного недоверия, существующего в таких командах, а также учитывая то, насколько иногда легко неправильно интерпретировать сообщения электронной почты, эти команды предпочитают временно ограничить использование электронной почты и прибегать к использованию обычного телефона каждый раз, когда нужно обменяться какой-то важной информацией.

Не все последствия увеличения объемов письменного общения оказываются отрицательными. Например, Джейн Робартс рассказывает интересную историю одного своего коллеги, который пользовался письменными коммуникациями в качестве эффективного дополнения к устному общению.

Передавая кому-либо устное сообщение, он всегда дополнял его письменным вариантом (как правило, с помощью PowerPoint). Зачастую мы даже не пользовались PowerPoint во время сеансов конференц-связи, но позже при необходимости можно было обратиться к этому письменному варианту, прояснив таким образом то или иное сообщение для членов команды, которые не смогли четко расслышать информацию, переданную в ходе сеанса конференц-связи, не могли присутствовать на сеансе конференц-связи или хотели уточнить смысл услышанного (2008, 329).

Этот метод может оказаться особенно полезным, когда члены команды говорят на разных языках и недостаточно хорошо владеют языками своих коллег. В таких случаях письменные документы также могут использоваться для прояснения смысла того, что было услышано в ходе устного общения.

Добавление подробностей в журнал запросов на выполнение работ

В главе 13, “Журнал запросов на выполнение работ”, подчеркивалась важность перехода от письменного изложения требований к устному их обсуждению. Однако многие команды приходят к выводу, что в случае рассредоточенной разработки проекта они не могут отказаться от письменного изложения требований в той мере, в какой им хотелось бы. Мартин Фаулер сказал, что “географическая рассредоточенность команды заставляет ее привносить больше формальностей в процесс изложения требований” (2006). Резюмируя свой опыт использования журнала запросов на выполнение работ в случае рассредоточенного проекта, Элейн Терриен (Elaine Therrien) из компании FirstAmerican CoreLogic говорит, что “дополнение высокогородневых пользовательских историй более подробными спецификациями помогает лучше действовать зарубежные ресурсы. Наличие более подробных требований помогает команде лучше уяснить соответствующую функциональную возможность и потребности конечного пользователя в те моменты, когда у нее нет возможности обратиться за разъяснениями к владельцу продукта” (2008, 371).

Опыт Фаулера, а также опыт Элейн Терриен с ее командами в Калифорнии и Бангалоре подобен моему собственному опыту работы с сильно рассредоточенными командами. В условиях значительной разницы во времени (например, 12,5-часовой разница между Калифорнией и Бангалором), командам, работающим в столь удаленных друг от друга часовых поясах, придется столкнуться со значительными проблемами из-за полного несовпадения рабочего времени. Зачастую в подобных ситуациях владелец продукта и команда находятся в часовых поясах, удаленных друг от друга на значительное расстояние. В таких случаях я рекомендую пользоваться методом, который я сам называю “отправляй вместе с контрольным примером”. Идея этого метода заключается в следующем: когда владелец продукта отправляет команде какую-либо пользовательскую историю из журнала запросов на выполнение работ, эта пользовательская история должна дополняться высокогородневыми контрольными примерами, позволяющими проверить готовность этой пользовательской истории.

См. также Эти контрольные примеры представляют собой “условия удовлетворенности”, о которых рассказывалось в главе 13, “Журнал запросов на выполнение работ”.

Поощрение горизонтальных связей

В типичном проекте, использующем последовательный процесс разработки, большинство связей между группами исполнителей, расположенными в разных городах, осуществляется через лидера команды, в обязанности которого входит организация такого общения. При выполнении Scrum-проекта нужно избегать такого подхода к организации общения членов рассредоточенной команды, поощряя горизонтальные коммуникации, при которых любой член группы, дислоцированной в одном городе, может свободно общаться с любым членом группы, дислоцированной в другом городе. Такое общение не просто допускается, оно всемерно поощряется. Вот что говорит по этому поводу Эди Миллер из компании Microsoft: “Наставник должен помогать своей команде не забывать о ценности интенсивного общения, даже когда рассредоточенность команды затрудняет процесс такого общения” (2008, 13).

Важным преимуществом горизонтальных связей является то, что они помогают бороться с так называемым “эффектом умолчания” (Ramingwong and Sajeev, 2007). Эффект умолчания имеет место в случае, когда кто-либо из участников проекта не желает сообщать плохие новости своим коллегам. Не желая сообщать плохие новости своим коллегам, этот участник проекта подвергает риску проект в целом, поскольку невозможно решить проблему, не зная о ее существовании. Всем известно (и, вообще говоря, вполне объяснимо), что представители разных культур делятся плохими новостями по-разному и проявляют разную степень готовности делиться такими новостями со своими коллегами, друзьями и знакомыми. Из этого следует, что эффект умолчания в большей степени характерен именно для рассредоточенных команд, а его последствия оказываются потенциально более тяжелыми в случае рассредоточенных проектов, в выполнении которых принимают участие представители разных культур. Рамингвонг и Саджеев указывают на три причины, заставляющие членов команд умалчивать о плохих новостях.

- Боязнь наказания, в том числе и увольнения
- Ложно понимаемое чувство солидарности с остальными членами команды
- Отсутствие заранее определенного канала, по которому плохие новости могут быть доведены до сведения остальных членов команды

Проект, в котором предусмотрена и поощряется возможность горизонтальных коммуникаций, в меньшей степени подвержен влиянию эффекта умолчания. Очень не-просто сформировать такую культуру проекта, при которой каждый из исполнителей готов поделиться любой известной ему информацией со всеми остальными членами команды. К счастью, горизонтальные коммуникации снижают потребность в этом. Возможно, я не желаю сообщать плохие новости непосредственно владельцу продукта, но я готов, как бы между прочим, упомянуть о них вам, когда мы вместе будем работать над какой-то задачей. К тому же я знаю, что вы готовы сообщить эти новости владельцу продукта. Этот тип горизонтальных коммуникаций особенно важен при выполнении проектов, в которых участвуют представители разных культур или люди, особенности характера которых мешают им (например, из чувства страха) делиться плохими новостями с теми, кого они воспринимают как своих “начальников”.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Во время очередного совещания, посвященного планированию спринта, обсудите, достаточно ли подробностей предоставляемых вместе с каждым элементом журнала запросов на выполнение работ, планируемым для нового спринта. Если недостаточно, то либо включите дополнительные подробности в ходе этого совещания, либо рассмотрите возможность выбора каких-либо других элементов журнала запросов на выполнение работ и добавления подробностей во время этого спринта.
- ❑ Во время следующей ретроспективы спринта обсудите эффект умолчания. Обсудите возможные последствия умолчания кем-либо из членов команды о какой-либо плохой новости. Исследуйте методом мозгового штурма способы, посредством которых можно было бы помочь друг другу избежать подобных проблем.

Совещания

Когда мне было десять лет, я провел лето со своей бабушкой, которая проживала неподалеку от Нью-Орлеана, в южной части Соединенных Штатов Америки (довольно жарким и влажном месте). Поскольку я вырос в Южной Калифорнии, климат которой близок к идеальному (с моей точки зрения), я постоянно жаловался бабушке на невыносимую жару в Нью-Орлеане. Она каждый раз отвечала в том духе, что дело не столько в жаре, сколько во влажности. Примерно так же я могу утверждать, что проблемы рассредоточенной команды заключаются не столько в значительных расстояниях, разделяющих членов команды, сколько в разнице во времени.

Разница часовых поясов оказывает гораздо большее влияние на слаженность действий команды, чем географические расстояния, разделяющие между собой разные группы исполнителей. Однажды мне пришлось принимать участие в проекте, исполнители которого дислоцировались в Калифорнии, Лондоне и Южной Африке. На рис. 18.2 указаны расстояния между этими тремя местами как в километрах, так и в количестве часовых поясов. Физическое расстояние между Сан-Франциско и Лондоном (8600 км) не очень-то отличается от расстояния между Лондоном и Кейптауном (9700 км). Однако Сан-Франциско и Лондон отстоят друг от друга на восемь часовых поясов, тогда как Лондон и Кейптаун разделяются лишь двумя часовыми поясами. Нетрудно представить, что члены команды в Сан-Франциско столкнулись с гораздо большими проблемами, чем члены команд в Лондоне и Кейптауне, поскольку рабочий день последних практически совпадал, что значительно облегчало организацию взаимодействия этих двух команд.

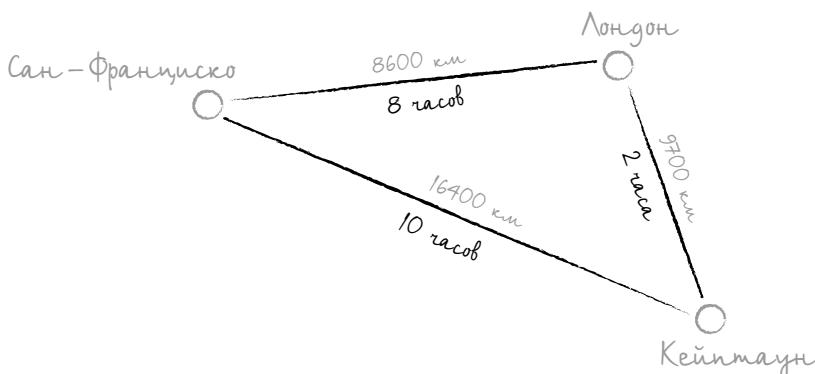


Рис. 18.2. Разница часовых поясов оказывает гораздо большее влияние на слаженность действий команды, чем географические расстояния

Разумеется, из сказанного не следует, что само по себе расстояние не порождает проблем. Команда, рассредоточенная между Осло и Франкфуртом, которые находятся в одном и том же часовом поясе, по-прежнему будет сталкиваться с проблемами, с которыми она не сталкивалась бы, если бы вся она была сосредоточена в одном и том же городе. Но по крайней мере их рабочий день полностью совпадает, а это облегчает им решение проблем, обусловленных рассредоточенностью команды.

Советы общего характера

В этом разделе мы рассмотрим, как связанные между собой проблемы времени и расстояния влияют на четыре типичных вида совещаний, проводимых при выполнении Scrum-проекта — ежедневные совещания разработчиков, планирование спринтов, ретроспективы спринтов и обзоры спринтов, — а также на объединенные Scrum-совещания, проводимые при выполнении многокомандных проектов. Сначала, однако, приведем ряд советов общего характера, применимых ко всем категориям совещаний.

Предусматривайте время для неформального общения

У команд, сосредоточенных в одном месте, есть множество возможностей для неформального общения, так называемой “светской болтовни”. Они могут позволить себе эту небольшую вольность. Что же касается рассредоточенных команд, то они должны использовать буквально каждую возможность для неформального общения своих членов. Мартин Фаулер пришел к выводу, что “начинать сеанс конференц-связи с обсуждения местных новостей — хорошая привычка. Последние новости местного уровня — политика, спорт, погода — помогают каждой из сторон, участвующих в сеансе конференц-связи, почувствовать более широкий жизненный контекст на другой стороне провода” (2006). Вспоминая свой опыт участия в проекте, рассредоточенном между Сан-Франциско, Бостоном и Торонто, Синик Янг (Cunick Young) и Хироки Терасима (Hiroki Terashima) высказывают точку зрения, полностью совпадающую с выводом Мартина Фаулера.

Когда разработчики впервые приступили к обсуждению технических аспектов проекта, мы почувствовали, что кое-кто из членов команды вел себя слишком профессионально и скованно. Было трудно обсуждать что-либо, кроме работы, и результатом каждого из совещаний было ощущение перегруженности информацией. Придя к выводу, что это — далеко не лучший способ наладить сотрудничество между членами команды, мы решили, что в начале каждого совещания нужно предусмотреть короткий период “разминки”. В ходе такой “разминки” мы обсуждали такие обыденные темы, как погода, здоровье каждого из участников совещания, а также все остальное, что приходило нам в голову. Такой способ начинать каждое совещание позволял расслабиться всем его участникам и лучше понять настроение каждого из них, что способствовало созданию более благоприятной атмосферы для проведения совещания (2008, 306).

Неплохой заменой этим импровизированным возможностям могут выступать современные технологии. Одна компания, в которой мне пришлось когда-то работать, широко использовала сеансы конференц-видеосвязи. В данном случае это означало, что в каждом из офисов, участвующих в проекте, комнаты, в которых проводились совещания, оснащались оборудованием видеосвязи, причем каждая команда могла пользоваться такой комнатой как помещением, выделенным специально для этой команды. Я всемерно призывал членов команды поглощать в этих комнатах бутерброды, устраивать перерывы в работе и т.п. Эти комнаты были оснащены видеооборудованием, работающим в режиме “он-лайн”. Учитывая то обстоятельство, что команды были разделены друг от друга тремя часовыми поясами, такой подход оказался весьма успешным. Члены команды, работавшие на западном побережье Соединенных Штатов, могли устроить короткий утренний перерыв примерно в то время, когда у членов

команды, работавших на восточном побережье Соединенных Штатов (с опережением на три часа), был обед; члены команды на восточном побережье могли устроить в своей комнате для проведения видеоконференций послеобеденный перерыв в то самое время, когда члены команды на западном побережье обедали. Неформальное общение, которое, естественно, имеет место в подобных случаях, помогало людям в обоих офисах чувствовать себя единой командой.

Неудобства должны разделять в равной мере все участники проекта

Если ваш проект рассредоточен так, что совещания приходится проводить или задолго до начала рабочего дня, или намного позже его начала, обязательно сделайте так, чтобы неудобства, связанные с этим, делились поровну между всеми участниками проекта. Не составляйте график совещаний таким образом, чтобы в выигрыше постоянно оказывались одни и те же люди. Если, например, команда рассредоточена между Калифорнией и Бангaloreм (разница во времени составляет 12,5 часов), не планируйте телефонный разговор на 8 часов утра по калифорнийскому времени и 8:30 вечера по индийскому времени. В то время как для большинства членов калифорнийской команды это окажется несколько раньше обычного начала рабочего времени (что никак не может считаться для них совершенно неприемлемым), то члены индийской команды посчитали бы это более чем неподходящим временем для регулярного проведения совещаний.

Поэтому поступайте так, как поступил Мэтт Траксоу (Matt Truxaw), когда он занимал пост руководителя разработки приложений в компании FirstAmerican CoreLogic (Санта-Фе, шт. Калифорния). Мэтт предложил калифорнийской и индийской командам, которыми он руководил, разделить неудобства поровну между собой, попарно соглашаясь на неудобное для себя время сеанса связи (например, в течение одного месяца сеансы связи должны были проводиться в вечернее время, а в течение другого месяца — в утреннее время).

Это помогло обеим командам поддерживать регулярную связь, не тая друг на друга обиду за то, что кому-то из них приходится нести большее бремя неудобств, чем другому. Это также помогло командам в Индии почувствовать, что они являются полноправными участниками процесса, т.е. не простыми наемниками, а настоящими членами команды.

Подобное чередование времени проведения сеансов связи помогает обеспечить надлежащий баланс власти между двумя местами сосредоточения исполнителей проекта. Существует тенденция к аккумуляции власти в том месте, где находится штаб-квартира соответствующей компании, где находится владелец продукта или где сосредоточено большинство разработчиков. Подобное нарушение баланса власти, т.е. ее аккумуляция в одном месте, может вызвать справедливое возмущение тех, кто пострадал от такого нарушения. Простые меры наподобие чередования времени проведения сеансов связи помогают предотвратить такой результат.

Участие в совещании по телефону может оказаться особенно проблематичным, если большинство других участников совещания присутствуют лично. Так зачастую бывает, когда в одном месте сосредоточена большая часть команды, а в другом — лишь два или три ее члена. Эта проблема оказывается особенно актуальной, когда эти два

или три члена работают в своих домашних офисах. Способ разделить это неудобство поровну между всеми заключается в том, чтобы поочередно вызывать по телефону то одного, то другого. Это дает возможность всем участникам совещания понять, насколько трудно бывает полностью сосредоточиться на общении по телефону.

Сообщите каждому, кто говорит по телефону

Очевидной проблемой использования телефонной связи при проведении совещаний является распознавание голосов тех, кто находится на другом конце провода. Кому-то такое распознаваниеается очень легко: они быстро запоминают голоса даже малознакомых им людей, находящихся на другом континенте. Я никогда не принадлежал к их числу, поэтому я отдаю должное командам, которые используют проверенное временем правило: всегда называйте свои имя и фамилию, перед тем как выразить по телефону то или иное мнение. Несмотря на безотказность этого метода многократные вступления типа “С вами говорит Майк...” требуют определенного времени, особенно ценного в случае международных телефонных переговоров. К тому же в ходе быстротечного и горячего обсуждения иногда трудно постоянно помнить о необходимости представляться, прежде чем высказать свою точку зрения.

Одна из команд, с которой мне пришлось когда-то работать, обнаружила интересный нюанс, позволяющий усовершенствовать описанный выше подход (который предполагает, что человек, взявший телефонную трубку, начинает свой комментарий с того, что называет свое имя). Члены команды назвали этот усовершенствованный метод “сеансом конференц-видеосвязи с низким качеством сигнала” и зачастую предполагали его обычному сеансу конференц-видеосвязи из-за неизбежных проблем и задержек, обусловленных использованием соответствующего оборудования. В случае проведения сеанса конференц-видеосвязи с низким качеством сигнала в каждом городе участие в сеансе принимал сотрудник, способный хорошо различать голоса людей на другом конце провода. Каждый раз, когда в разговор вступал новый участник сеанса, этот сотрудник показывал фотографию человека, говорящего на другом конце провода. Когда начинала говорить Сонали, соответствующий сотрудник показывал фотографию Сонали. Когда она заканчивала свой комментарий и в разговор вступал Маниш, этот сотрудник показывал фотографию Маниша. Фотографии лиц, которые должны были принять участие в таком сеансе конференц-видеосвязи с низким качеством сигнала, подбирались заранее, прикреплялись липкой лентой к линейкам и выкладывались таким образом, чтобы можно было побыстрее найти фотографию человека, вступающего в разговор.

Я понимаю, что в предложенном мною описании этот метод выглядит несколько глуповато, но уверяю вас, что на практике он существенно облегчает проведение сеансов конференц-видеосвязи. Наблюдая за тем, как команды этой компании проводят таким способом сеансы конференц-связи, я заметил две интересные вещи. Во-первых, в то время как одни команды предоставляют возможность одному и тому же человеку всегда поднимать фотографии говорящих по телефону, в других командах это право может быть предоставлено любому их члену. Это выливалось в своего рода соревнование, кто быстрее остальных распознает голос говорящего по телефону и первым поднимет фотографию. Такой подход заставлял всех присутствующих пристально следить за выступлениями своих товарищей по команде. Во-вторых, люди не ограничивались тем, что, мельком взглянув на фотографию, говорили: “А, это говорит

Ранджит". Напротив, они продолжали смотреть на фотографию, словно надеясь на то, что изображение Ранджит, смотревшее на них с фотографии, вот-вот заговорит.

Совещание, посвященное планированию спрингта

В этом разделе мы рассмотрим две общепринятые стратегии проведения совещания рассредоточенной команды, посвященного планированию спрингта. Эти стратегии обозначаются их описательными названиями: продолжительный телефонный разговор и два телефонных звонка. Рассмотрим преимущества и недостатки каждой из этих стратегий.

Продолжительный телефонный разговор

Этот подход, используемый по умолчанию большинством команд, предполагает участие в сеансе конференц-связи всех членов команды; во всех остальных отношениях совещание, посвященное планированию спрингта, проводится как обычно. Такой подход пытается имитировать формат и взаимодействие участников обычного совещания, посвященного планированию спрингта, т.е. совещания, в котором лично участвуют все члены команды. Во время этого телефонного разговора проводится вся работа, характерная для обычного совещания, посвященного планированию спрингта. Когда сеанс конференц-связи завершается, соответствующий спрингт оказывается полностью спланированным, т.е. все происходит так, как если бы вся команда была сосредоточена в одном месте.

Это идеальный пример того, почему разница во времени можетказываться даже хуже, чем сама по себе физическая рассредоточенность команды. Очевидно, что этот подход к рассредоточенному планированию спрингта возможен лишь в случаях значительного совпадения рабочего времени всех членов рассредоточенной команды. Никакая команда не согласится регулярно планировать свои спрингты, например, с 19:00 до полуночи.

В целом метод продолжительного телефонного разговора мне нравится. Однако практическая целесообразность этого подхода обычно определяется тем, насколько широко рассредоточена соответствующая команда. Если все члены рассредоточенной команды работают в одном и том же часовом поясе, может работать несколько дольше обычного рабочего дня или может несколько сдвигать часы своей работы, то метод продолжительного телефонного разговора может вполне подойти ей. Преимущества и недостатки этого подхода подытожены в табл. 18.2.

Таблица 18.2. Преимущества и недостатки планирования спрингтов в ходе одного продолжительного телефонного разговора

Преимущество	Недостаток
Может вести к плодотворному обсуждению, если в нем принимают активное участие все члены рассредоточенной команды	Иногда участникам столь продолжительного телефонного разговора нелегко поддерживать высокую концентрацию своего внимания
Планирование спрингта можно завершить за один день	Может использоваться только при условии значительного совпадения рабочего времени у всех групп рассредоточенной команды
Соответствует подходу, используемому сосредоточенными командами	Может предусматривать удлинение рабочего дня в одном или нескольких местах

Два телефонных звонка

Некоторым командам просто нет смысла в ходе совещания, посвященного планированию спринта, укладываться в один телефонный разговор: разница во времени слишком велика, чтобы обеспечить достаточное перекрытие рабочего времени разных групп рассредоточенной команды. Следующий подход к планированию спринта, под названием “два телефонных звонка”, позволяет решить эту проблему путем разделения совещания на два телефонных звонка, совершаемых в течение двух следующих друг за другом дней. Роджер Нессьеर (Roger Nessier), вице-президент Symphony Services, рассказывает о том, как его команда сегментировала такой телефонный разговор.

Замена первоначального восьмичасового сеанса связи двумя раздельными четырехчасовыми сеансами, проводимыми в течение двух следующих друг за другом дней, показалась нам весьма разумным решением. Например, первый сеанс связи посвящается определению важнейших задач, результатов работы, которые можно будет предъявить заказчику, и высокогорневых зависимостей. В течение второго сеанса связи каждый из членов команды определяет действия и предоставляет оценки по каждой задаче, которую он взялся выполнить (2007, 8-9).

Одни команды предпочитают заниматься планированием исключительно во время сеансов связи по телефону; а другие — использовать время между первым и вторым сеансами телефонной связи для индивидуальной подготовки ко второй части совещания. Именно таким подходом воспользовалась команда Vespa News Search из компании Yahoo!, продукт-менеджеры которой работали в двух калифорнийских городах, а разработчики — в Норвегии (разница во времени — девять часов). Поскольку они работали двухнедельными спринтами, члены команды решили ограничить первоначальный сеанс телефонной связи двумя часами. Они решили проводить его с 16:00 до 18:00 в Норвегии и с 7:00 до 9:00 в Калифорнии. Они также решили твердо придерживаться таких временных рамок. Вот как описывают это совещание члены команды Брайан Драммонд и Дж.Ф. Ансон.

Члены команды обдумывали каждый элемент журнала запросов на выполнение работ, обращаясь к продукт-менеджеру с вопросами, касающимися критерии приемки, областей применения каждой из функциональных возможностей, экономических ограничений и т.п. После завершения обсуждений пути команд расходились: члены норвежской команды возвращались к себе домой (в Норвегии уже был вечер), а члены калифорнийской команды отправлялись на свои рабочие места (в Калифорнии начинался рабочий день) (2008, 317).

Первое совещание посвящается обсуждению самых высокоприоритетных, с точки зрения владельца продукта, функциональных возможностей и ожиданий. Как указывают Драммонд и Ансон, для этого совещания характерно интенсивное обсуждение, которое ведут между собой команда разработчиков и владелец продукта. По завершении этого первого совещания части команды, сосредоточенные в разных городах, проводят собственные совещания, посвященные планированию их собственных частей предстоящего спринта. Эти совещания могут происходить в тот же день или на следующее утро в зависимости от мест дислокации тех или иных частей команды (субкоманд). Во время этих вторых совещаний субкоманды выявляют задачи, необходимые для реализации функциональных возможностей, которые обсуждались в ходе первоначального

сеанса связи, проводившегося с участием всех членов рассредоточенной команды. Планирование спринта завершается в ходе второго сеанса связи, который обычно проводится на следующий день и примерно в то же время, что и первоначальный сеанс связи. Целью этого второго сеанса связи является взаимная синхронизация задач, выполнение которых взяла на себя каждая из субкоманд. Допустим, что в ходе первоначального сеанса связи обсуждались четыре функциональные возможности. После этого первоначального сеанса связи субкоманда в Норвегии приходит к выводу, что может реализовать лишь три из этих четырех функциональных возможностей, тогда как субкоманда в Калифорнии берется реализовать все четыре функциональные возможности. В ходе второго сеанса связи вся команда в целом должна исследовать возможные способы реализации четвертой функциональной возможности в полном объеме или по крайней мере найти такой меньший элемент журнала запросов на выполнение работ, за выполнение которого могли бы взяться обе субкоманды. Драммонд и Ансон из Yahoo! пришли к выводу об эффективности такого подхода.

Наша команда смогла минимизировать и равномерно распределить неудобства, связанные с проведением совещания с участием субкоманд, действующих в разных часовых поясах. Актуальность обсуждаемой информации удавалось поддерживать на достаточно высоком уровне, не позволяя участникам совещания отклоняться в сторону от главной темы совещания и отводя каждому из участников обсуждения строго определенный интервал времени для изложения собственного мнения. Любые дискуссии, которые не имели никакого отношения к разработчикам, со-средоточенным в другом городе, переносились на время, более удобное для местной команды (2008, 318).

Преимущества и недостатки этого подхода подытожены в табл. 18.3.

Таблица 18.3. Преимущества и недостатки разделения планирования спринта на два сеанса телефонной связи

Преимущество	Недостаток
Может оказаться более эффективным вариантом использования времени	Полезность может колебаться в очень широких пределах в зависимости от того, как именно рассредоточена команда
Может использоваться даже в том случае, когда рабочее время субкоманд пересекается лишь незначительно (или если такое незначительное пересечение может быть достигнуто искусственно)	Поскольку многие обсуждения проводятся в рамках субкоманд, не всю информацию удается сделать достоянием всей команды. Впоследствии это может приводить к взаимному непониманию команд и неправильной интерпретации ими сведений, полученных по телефону
	Не может быть завершено в течение одного дня

ПРИМЕЧАНИЕ

Третий подход, который используется в отдельных случаях, заключается в том, что все планирование в каждом из мест дислокации поручается ведущим техническим специалистам. Несмотря на то что такой подход позволяет минимизировать проблемы, вызванные разницей во времени, я не могу рекомендовать его из-за следующих трех его недостатков.

- В планировании участвуют далеко не все члены команды, что снижает их личную заинтересованность в составлении эффективного плана, а также понимание ими работы, которую предстоит выполнять.
- Без личного участия в планировании каждого из членов команды высока вероятность того, что о каких-то задачах попросту забудут или оценка таких задач окажется неправильной.
- Использование такого подхода препятствует самоорганизации и не позволяет команде почувствовать свою ответственность за проблему, которую ей предстоит решить.

Ежедневное совещание разработчиков

Ежедневное совещание разработчиков влечет со собой совокупность проблем, совершенно непохожую на ту, которую влечет со собой совещание, посвященное планированию спринта. В то время как планирование спрингта требует проведения продолжительных, но не таких уж частых совещаний, ежедневное совещание разработчиков представляет собой непродолжительное, но *ежедневное* совещание. Поскольку продолжительность ежедневного совещания разработчиков ограничивается 15 минутами, проведение такого совещания не представляет проблемы для команд, рабочее время которых перекрывается. Однако ежедневные совещания разработчиков представляют проблему для широко рассредоточенных команд, рабочее время которых не перекрывается. Устраивать сеанс связи каждый день за рамками обычного рабочего дня — можно ли такое выдержать, например, несколько месяцев кряду? Такие команды пользуются тремя основными стратегиями: единый сеанс связи, проведение совещания в письменной форме и региональные совещания.

Единый сеанс связи

Возможно, самым распространенным подходом, который использует в первую очередь большинство рассредоточенных команд, является единый сеанс связи, в котором участвуют все члены рассредоточенной команды. Для команд, находящихся на расстоянии двух-трех часовых поясов друг от друга, такой подход является идеальным. К сожалению, эффективность такого подхода снижается пропорционально количеству часовых поясов, разделяющих субкоманды. В конечном счете большинство рассредоточенных команд приходит к выводу о необходимости использования какой-то другой стратегии для проведения своих ежедневных сеансов связи.

Некоторые из широко рассредоточенных команд пытаются преодолеть неудобство единого сеанса связи, проводя совещания разработчиков несколько реже (возможно, каждые два или три дня). Я полностью осознаю неудобства, вызванные необходимостью проводить ежедневные совещания разработчиков за рамками обычного рабочего дня. Однако каждый раз, испытывая соблазн сократить частоту ежедневных совещаний разработчиков, я вспоминаю следующую фразу Фреда Брукса (Fred Brooks) в книге *Мифический человеко-месяц*: “Каким образом завершение проекта может запоздать на целый год?.. День — на этот раз, день — в следующий раз, и т.д.” (1995, 153). Ежедневные совещания разработчиков я рекомендую проводить *ежедневно*. Если вы принимаете другое решение, постарайтесь по крайней мере заменить пропущенные совещания письменными версиями совещаний или сеансом связи, в котором будут

принимать участие по одному представителю от каждого из мест дислокации разработчиков. Каждый из таких подходов представляет собой оставшиеся две стратегии проведения ежедневных совещаний разработчиков, которые будут описаны ниже.

Команды, которые принимают решение использовать метод единого сеанса связи, должны рассмотреть возможность периодического изменения времени проведения такого сеанса связи, как описано выше в этой главе, в разделе “Неудобства должны разделять в равной мере все участники проекта”. Преимущества и недостатки метода единого сеанса связи подытожены в табл. 18.4.

Таблица 18.4. Преимущества и недостатки участия всех членов команды в ежедневных телефонных совещаниях

Преимущество	Недостаток
Подобен методу, используемому в случае команд, дислоцированных в одном городе, поэтому осваивать что-либо новое не приходится	Может оказаться очень неудобным для членов команды
Обсуждения проводятся в присутствии всей команды	Неприемлем, если людям приходится участвовать в сеансах связи за рамками своего обычного рабочего дня
Каждый узнает обо всех проблемах, что обуславливает большую информированность команды и большую ответственность за достижение общей цели	

Совещания в письменной форме

Чтобы сгладить неудобства, обусловленные проведением сеансов телефонной связи за рамками обычного рабочего дня, некоторые команды вообще отказываются от ежедневных совещаний разработчиков. В то же время, не желая полностью жертвовать выгодами ежедневного общения разработчиков, такие команды обычно заменяют ежедневные совещания разработчиков проведением совещания в той или иной письменной форме. Члены команды соглашаются отправлять электронную почту, обновлять вики-страницу или вводить свои комментарии с помощью какого-либо другого инструмента асинхронного сотрудничества, обеспечивающего обмен той же информацией, какой они делились бы друг с другом в ходе сеанса телефонной связи.

Разновидностью этого подхода является проведение сеанса телефонной связи во время, удобное для большинства членов рассредоточенной команды. При этом другие члены команды “участвуют” в совещании, подавая свои письменные отчеты. Это особенно распространенный вариант проведения ежедневных совещаний разработчиков, когда большая часть команды дислоцирована в одном городе, а в дистанционном режиме работает лишь пара-тройка разработчиков.

Обычно я не рекомендую такой подход в качестве основного. Его можно использовать лишь в качестве дополнения к ежедневным сеансам телефонной связи, если команда приходит к выводу, что ежедневные сеансы телефонной связи — это уж слишком и нужно проводить их пореже. У ежедневных сеансов телефонной связи есть важные побочные эффекты, которые утрачиваются, когда такой сеанс связи превращается в ежедневный обмен письменными посланиями. Например, обязательство выполнить определенный объем работы кажется более серьезным, когда соответствующий член команды говорит: “Я сделаю это сегодня”, чем когда та же фраза

предъявляется в письменном виде. Возможно, это объясняется тем, что устное сообщение представляет собой обязательство, взятое перед сотрудниками того, кто берет на себя это обязательство, тогда как письменное сообщение представляет собой обязательство, взятое частным образом и прочитанное, возможно, лишь кем-то из сотрудников того, кто берет на себя это обязательство. Преимущества и недостатки этого подхода подытожены в табл. 18.5.

Таблица 18.5. Преимущества и недостатки замены ежедневных совещаний разработчиков соответствующей письменной информацией

Преимущество	Недостаток
Может использоваться в течение длительного времени	Возникающие проблемы не обсуждаются и потому могут оставаться нерешенными в течение многих дней
Помогает преодолеть языковые барьеры, в том числе проблемы сильного акцента	Не позволяет налаживать отношения и делиться знаниями членам рассредоточенной команды в ходе ежедневного общения
	Нет гарантии, что письменные сообщения будут прочитаны
	Члены команды будут с меньшей ответственностью относиться к собственным обязательствам, взятым ими днем ранее

Региональные совещания

Третийм, и последним, подходом к проведению ежедневных совещаний разработчиков является проведение совокупности региональных совещаний, сопровождаемое определенными мерами по взаимному обмену информацией по ключевым вопросам, которые обсуждались на этих региональных совещаниях. Если команда рассредоточена между двумя достаточно удаленными друг от друга городами, то в каждом из этих городов может проводиться свое ежедневное совещание разработчиков. Это могло бы иметь место, например, в случае команды, рассредоточенной между Сан-Франциско и Лондоном, отстоящими друг от друга на расстоянии восьми часовых поясов.

Иногда у рассредоточенной команды есть несколько офисов, рабочие часы в которых перекрываются, плюс один офис, находящийся на большом удалении. В подобных случаях команды в местах с перекрывающимся рабочим временем могли бы проводить единное для них региональное ежедневное совещание разработчиков. Однако в более удаленном офисе пришлось бы проводить собственное ежедневное совещание разработчиков. Если бы, например, к нашим командам в Сан-Франциско и Лондоне подключилась команда в Лос-Анджелесе, то одним из приемлемых вариантов могло бы стать проведение по телефону ежедневного совещания разработчиков для двух калифорнийских команд и отдельного ежедневного совещания разработчиков в Лондоне, на котором должны были бы присутствовать все разработчики, дислоцированные в лондонском офисе.

Типичный подход заключается в проведении раздельных сеансов телефонной связи для Западного и Восточного полушарий. В сеансе телефонной связи для Западного полушария вполне могли бы участвовать все участники проекта, дислоцированные в Северной и Южной Америке, тогда как в телефонной связи для Восточного полушария могли бы участвовать остальные участники проекта, работающие на других континентах (возможно, за исключением Австралии и Новой Зеландии). Если какая-то команда рассредоточена настолько широко, что ее члены работают, например, в Сан-Франциско, Сиэтле, Торонто, Лондоне, Праге, Стокгольме, Пекине и Мельбурне,

то, возможно, ей придется проводить не два, а три сеанса телефонной связи в разное время. Но не следует забывать, что мы обсуждаем здесь ежедневные совещания разработчиков, а не объединенные Scrum-совещания (речь о которых будет идти ниже). Для большинства рассредоточенных команд вполне достаточно и двух сеансов телефонной связи, запланированных на разное время.

Эти региональные совещания (которые могут проводиться либо в присутствии всех членов команды, дислоцированных в соответствующем офисе, либо с использованием сеанса конференц-связи, объединяющего несколько городов) обычно сопровождаются дополнительным общением, чтобы каждая из субкоманд была в курсе того, чем занимаются другие субкоманды. Одним из способов такого дополнительного общения является телефонный разговор по крайней мере с одним представителем каждой субкоманды. Именно такой подход использовал Мартин Фаулер, который говорит: “Мы проводим сеансы связи с командой, дислоцированной на том или другом побережье, но не между разными побережьями. Тем не менее мы проводим ежедневные совещания между командами, дислоцированными на разных побережьях, однако *вся* команда в этих совещаниях не участвует” (2006).

Другой подход к поддержанию связи между субкомандами заключается в том, что команда поручает одному или нескольким своим членам участвовать во всех совещаниях. Этот “уполномоченный” член команды сначала участвует в обычном совещании своей субкоманды, а затем — в одном или двух других совещаниях, которые, как правило, происходят за пределами обычного рабочего дня этого “уполномоченного”.

Какой бы из этих подходов к обмену информацией между субкомандами и координации их работы ни использовался, удается в значительной мере избавиться от неудобств, вызванных необходимостью проводить сеансы связи за рамками обычного рабочего дня. Правда, кому-то из членов команды (а именно — сотруднику, которому поручено ежедневно участвовать в сеансах связи, проводимых за пределами обычного рабочего дня) все же приходится мириться с этими неудобствами. Степень этого недостатка можно снизить еще больше путем ротации сотрудников, которым поручено каждый день участвовать в сеансах связи.

Я пришел к выводу, что проведение региональных совещаний подходит для большинства широко рассредоточенных команд. Хотя я и предпочел бы, чтобы *все* члены команды каждый день принимали участие в едином сеансе телефонной связи, этот вариант далеко не всегда возможен, особенно на протяжении длительного времени. Несмотря на то что у этого подхода есть свои недостатки, они перевешиваются (хотя и не превосходят по численности) его преимуществами. Преимущества и недостатки этого подхода подытожены в табл. 18.6.

Таблица 18.6. Преимущества и недостатки использования региональных совещаний для проведения ежедневного совещания разработчиков

Преимущество	Недостаток
Удается в значительной мере избавиться от неудобств, вызванных необходимостью проводить сеансы связи за рамками обычного рабочего дня	Информация, передаваемая с одного совещания на следующее, может оказаться неточной или неполной
Позволяет местным субкомандам обмениваться информацией, представляющей для них наибольшую ценность	Может вызывать у субкоманд ощущения типа “мы — это одно, а они — это совсем другое” в отношении других субкоманд

Окончание табл. 18.6

Преимущество	Недостаток
	В обсуждениях принимают участие не все члены команды
	Информационный обмен между субкомандами может проходить нерегулярно

Объединенные Scrum-совещания

См. также Объединенные Scrum-совещания описаны в главе 17, “Изменение масштаба Scrum”.

Объединенные Scrum-совещания проводятся несколькими командами с целью координирования своей работы. В объединенном Scrum-совещании принимают участие по одному представителю от каждой команды, участвующей в совещании. Объединенные Scrum-совещания обычно проводятся два-три раза в неделю. Меньшая частота этих совещаний делает менее проблематичным их проведение в рассредоточенной команде. Такое совещание почти всегда продолжается не более одного часа. Следовательно, каким бы ни было пересечение рабочего дня у субкоманд, составляющих рассредоточенную команду, такая команда всегда сможет легко спланировать объединенное Scrum-совещание.

Проблема, конечно же, возникает в случаях, когда команда рассредоточена настолько широко, что рабочее время у субкоманд совершенно не пересекается. В таких случаях команды пытаются использовать одну из лучших стратегий для ежедневного совещания разработчиков: единый сеанс связи или региональные совещания. Когда в реализации проекта участвует лишь несколько команд, вполне приемлемым вариантом может оказаться единый сеанс связи, если участникам совещания удается выбрать интервал времени, создающий лишь минимальные неудобства для достаточно большого числа участников проекта. Этого легче добиться, чем в случае ежедневного совещания разработчиков, по двум причинам: во-первых, сеансы связи большинства объединенных Scrum-совещаний не проводятся ежедневно и, во-вторых, большинство команд время от времени меняют состав участников этих сеансов связи. Необходимость участвовать следующие четыре вторника и четверга в сеансах связи, назначенных на 19:00, может создавать определенные неудобства, но что значит эти неудобства в сравнении с необходимостью участвовать в сеансах связи, назначенных на 19:00 и проводимых пять дней в неделю, — и так до самого завершения проекта!

Более многочисленные команды, а также команды, сталкивающиеся с более трудными проблемами, вызванными разницей во времени, нередко прибегают к региональным совещаниям. Например, проект, в котором участвуют четыре команды, дислоцированные в Торонто, три команды в Бангалоре и две в Пекине, могут проводить объединенные Scrum-совещания с личным участием членов всех четырех команд, дислоцированных в Торонто. Это совещание проводилось бы во время, неудобное для команд, дислоцированных в Бангалоре и в Пекине, поэтому члены банглорских и пекинских команд могут запланировать сеанс конференц-связи на более подходящее для себя время. Информационный обмен между этими двумя группами может происходить либо

посредством одного-двух человек, участвующих в обоих совещаниях, либо посредством сеанса телефонной связи, включающего представителя каждого такого совещания.

Обзоры и ретроспективы спринтов

Обзорам и ретроспективам спринтов присущи атрибуты как ежедневного совещания разработчиков, так и совещания, посвященного планированию спринта. Подобно совещаниям, посвященным планированию спринтов, обзоры и ретроспективы спринтов проводятся не каждый день, поэтому у членов команды есть больше возможностей участвовать в таких совещаниях за рамками своего обычного рабочего времени. Подобно ежедневным совещаниям разработчиков, обзоры и ретроспективы спринтов несколько проще планировать, поскольку они требуют меньше времени, чем совещания, посвященные планированию спринтов. Это несколько упрощает поиск подходящего “окна” для проведения обзоров или ретроспектив спринтов.

Команды с перекрывающимся рабочим временем, естественно, планируют обзоры и ретроспективы спринтов таким образом, чтобы они приходились на перекрывающуюся часть их рабочего дня. Команды, рабочее время которых перекрывается лишь едва-едва, обычно планируют обзоры и ретроспективы спринтов таким образом, чтобы они приходились на конец их текущего рабочего дня и начало следующего. Например, разница во времени между Денвером и Хельсинки составляет девять часов. Таким образом, обзор спринта можно запланировать на 8 часов утра в Денвере; в Хельсинки в это время будет 17:00 (т.е. до окончания рабочего дня остается один час). Членам денверской команды придется несколько раньше явиться на работу утром следующего дня, а членам хельсинской команды придется задержаться на работе несколько дольше обычного, но в целом такой подход оказывается вполне приемлемым, учитывая то обстоятельство, что обзоры спринтов проводятся лишь один раз в несколько недель.

Командам, рассредоточенным более широко, приходится изыскивать время, которое создавало бы минимум неудобств для личной жизни членов субкоманд, дислоцированных в разных местах. Команда, рассредоточенная между Лондоном и Новой Зеландией (разница во времени — 12 часов), может проводить совещание в 8 часов утра в одном месте и в 8 часов вечера — в другом. Как и в случае всех таких совещаний, проводимых в нерабочее время, следует периодически варьировать, в каком из мест это нерабочее время будет приходить на утренние часы, а в каком — на вечерние.

Если и обзоры, и ретроспективы спринтов продолжаются недолго, то некоторые команды предпочитают планировать обзор и ретроспективу таким образом, чтобы они следовали непрерывно друг за другом. Остальные команды предпочитают планировать проведение обзора и ретроспективы в разные, но следующие друг за другом дни. Возможным компромиссным вариантом является проведение обзора и ретроспективы в промежутке между двумя следующими друг за другом днями (с использованием непродолжительных сеансов телефонной связи, проводимых в нерабочее время) и в один день, но с использованием более продолжительного сеанса телефонной связи.

Участие не может быть необязательным

Проблема с проведением обзоров и ретроспектив спринтов заключается в том, что некоторые члены команды считают свое участие в этих совещаниях необязательным. Это совсем не так. Но несмотря на то что я вовсе не считаю участие членов команды

в этих совещаниях необязательным, я считаю нереалистичным рассчитывать на то, что каждый из членов команды будет присутствовать на каждом из этих совещаний, особенно если эти совещания проводятся в нерабочее время. Я полагаю, что нужно требовать, чтобы все без исключения члены команды принимали участие в каждом из этих совещаний, но отдавать себе отчет в том, что время от времени тот или иной сотрудник будет отсутствовать во время их проведения. Если кто-то из членов команды не сможет присутствовать на каком-то из этих совещаний, он по крайней мере обязан заранее предупредить своих коллег об этом (по телефону или электронной почте).

Я сравниваю участие членов команды в обзорах и ретроспективах спринтов, проводимых в нерабочее время, с участием моих дочерей в тренировках команды по плаванию. Участвовать в таких тренировках обязательно: мои дочери не могут пропустить тренировку, а затем явиться на соревнования в полной уверенности, что их включат в состав команды. Чтобы быть допущенными к соревнованиям, они должны участвовать во всех тренировках. Но тренер, конечно же, понимает, что речь идет о школьниках, у которых может оказаться тысячауважительных причин (посещение врача, болезнь младшего брата или сестры, дорожные "пробки", какое-либо важное мероприятие в школе и т.п.) для того, чтобы время от времени пропускать тренировки. Пропуск двух-трех тренировок — не проблема. Слишком много пропусков — и тренер потребует объяснений. То же самое касается Scrum-команды.

Время от времени ретроспективы нужно проводить в одном городе

Вообще говоря, я не сторонник отсутствия во время ретроспективы кого-либо из участников проекта. Я, например, не рекомендовал бы команде не пригласить на какое-либо из совещаний тестеров (мол, поговорим и без них). Аналогично я не рекомендовал бы команде проводить ретроспективу без владельца продукта. Однако использование телефонной связи может создавать на совещании неподходящую атмосферу, поэтому я рекомендовал бы команде иногда исключать использование телефонной связи в ходе совещания, подразумевая под этим, что в каждом месте дислокации разработчиков должна периодически проводиться своя, местная ретроспектива. В ходе такой ретроспективы, проводимой в одном городе, может обсуждаться любая тема, но я настоятельно рекомендовал бы субкоманде, дислоцированной в определенном городе, сосредоточивать свое внимание на двух вещах: вопросах, уникальных для данного места дислокации разработчиков, и на том, что эта субкоманда может сделать, чтобы способствовать взаимодействию с другими местами дислокации разработчиков.

Действуйте с осторожностью

Никто не принимает решение рассредоточить команду исходя из потребностей самой команды. Решение рассредоточить членов команды географически принимается по другим причинам: чтобы сэкономить деньги, чтобы нанять исполнителей проекта в разных местах, чтобы освоить какой-то новый регион, в связи с поглощением одной компании другой компанией или в силу каких-либо других причин такого рода. Рассредоточение команды приводит к необходимости выполнения дополнительной работы, вызывает определенный стресс у участников проекта и порождает гораздо больший риск для организации.

См. также Яковоу (Iacovou) и Накацу (Nakatsu) представляют в своей статье перечень рисков, которым подвергаются рассредоточенные проекты (2008).

Советы, изложенные в этой главе, являются следствием моего собственного опыта, а также опыта тех людей, с которыми мне приходилось общаться. Рассредоточенная разработка может быть реализована на практике, но рассредоточенная команда никогда не сможет действовать так же эффективно, как команда, дислоцированная в одном месте. Тем не менее, поскольку не всегда есть возможность сосредоточить в одном месте всю команду, организациям приходится изыскивать способы (к числу которых относятся и описанные в этой главе), которые помогали бы рассредоточенным командам с максимально возможной для себя эффективностью. Вместе с тем считаю уместным рассмотреть вывод, к которому пришли Эммелина де Пиллис (Emmeline de Pillis) и Кимберли Фурумо (Kimberly Furumo) в своей статье, опубликованной в сборнике *Communications of the ACM*. После проведения экспериментов, в ходе которых выполнялось сравнение производительности, удовлетворенности и групповой динамики рассредоточенных и сосредоточенных команд, авторы статьи пришли к выводу, что “виртуальные команды демонстрируют значительно более низкую производительность, более низкую удовлетворенность и более низкое отношение «достигнутый результат/затраченные усилия». Вместе с тем виртуальные команды отличаются более низкой степенью ответственности, более низким моральным духом и производительностью” (2007, 95).

Дополнительная литература

Carmel, Erran. 1998. *Global software teams: Collaborating across borders and time zones*. Prentice Hall.

Д-р Кармел — профессор Американского университета. Он является признанным экспертом по технологической глобализации. Его книга прекрасно дополняет книгу Duarate и Снайдера в том отношении, что она охватывает, в частности, разработку программного обеспечения и IT-проекты. Более ранняя книга д-ра Кармела, *Global Software Teams*, посвящена исключительно разработке программного обеспечения, но я предпочитаю именно эту, новую, книгу.

Duarte, Deborah L., and Nancy Tennant Snyder. 2006. *Mastering virtual teams: Strategies, tools, and techniques that succeed*. 3rd ed. Jossey-Bass.

Наилучшая, на мой взгляд, книга общего характера, посвященная работе с рассредоточенными (или “виртуальными”) командами. Содержит полезную информацию о групповой динамике, культуре, совещаниях и многом другом. Несмотря на то что в этой книге не рассказывается о Scrum и даже о разработке программного обеспечения, большая часть содержащегося в ней материала применима и к Scrum, и к разработке программного обеспечения.

Fowler, Martin. 2006. Using an agile software process with offshore development. Личный сайт Мартина Фаулера, 18 июля. <http://martinfowler.com/articles/agile-Offshore.html>.

На этой веб-странице представлены соображения главного научного сотрудника ThoughtWorks Мартина Фаулера, касающиеся рассредоточенной разработки на основе гибкой методологии разработки. Здесь описаны уроки, полученные в результате такой разработки, комментарии по поводу издержек и выгод рассредоточенной разработки, а также прогнозы относительного дальнейших перспектив рассредоточенной разработки и гибкой методологии разработки.

Miller, Ade. 2008. *Distributed agile development at Microsoft patterns & practices*. Microsoft. (Загружайте с веб-сайта издателя по адресу <http://www.pnpguidance.net/Post/Distributed-AgileDevelopmentMicrosoftPatternsPractices.aspx>.)

Эди Миллер, сотрудник группы моделей и методов в компании Microsoft, описывает проблемы, с которыми столкнулась эта рассредоточенная группа, и способы их решения.

Sutherland, Jeff, Anton Viktorov, and Jack Blount. 2006. Adaptive engineering of large software projects with distributed/outsourced teams. В сборнике *Proceedings of the Sixth International Conference on Complex Systems*, ed. Ali Minai, Dan Braha, and Yaneer Bar-Yam. New England Complex Systems Institute.

Сазерленд, Викторов и Блаунт рассматривают практический пример особенно успешного Scrum-проекта, рассредоточенного между тремя городами на двух континентах.

Глава 19

Со существованием с другими подходами

Одно дело — изучать разработку программного обеспечения посредством гибкой методологии разработки, так сказать, в пробирке, и совсем другое дело — испытывать ее на практике. В пробирке, где в действие не вступают суровые реалии корпоративной политики, экономики и тому подобного, гибкие методологии разработки наподобие Scrum с готовностью принимаются всеми разработчиками. Однако в реальном мире все эти неприятные проблемы встают во весь рост. Принять решение об использовании Scrum не так уж сложно. Гораздо сложнее использовать Scrum в условиях, когда действует ряд жестких ограничений. Для какого-то одного проекта использование Scrum можно допустить, если это не вступает в противоречие с сертификацией CMMI третьего уровня для соответствующей организации. (CMMI — Capability Maturity Model Integration — интегрированная модель технологической зрелости организации, или модель CMMI. — *Примеч. пер.*) Для другого проекта использование Scrum можно допустить, если он прошел процедуру предварительного рассмотрения архитектуры и если совещание на этапе завершения проектирования прошло успешно.

У организации может быть достаточно оснований для наложения на проекты указанных ограничений, но в любом случае ограничения остаются ограничениями. Термину “ограничения” я не придаю в данном случае какого-то отрицательного смысла; я просто употребляю его, чтобы указать на то, что у команды отбирается определенная степень свободы, и показать, как команда выполняет свою работу. Не все ограничения плохи; например, в Соединенных Штатах (как и в большинстве других стран) разрешено лишь правостороннее движение. Я охотно соблюдаю это ограничение, поскольку знаю, что все остальные водители также ему подчиняются и, следовательно, вероятность столкновений на дорогах существенно снижается. Аналогично многим Scrum-командам приходится — по крайней мере поначалу — действовать в рамках правил и норм своей организации.

Из этой главы вы узнаете, какому влиянию подвергается Scrum-проект, когда он пересекается с последовательным (“водопадным”) процессом. Затем мы рассмотрим влияние надзора за выполнением проекта и покажем, как Scrum-проекты могут успешно сосуществовать с негибкими подходами к надзору за выполнением проекта. Наконец мы рассмотрим, каким образом можно обеспечить соответствие Scrum-проектов таким стандартам, как ISO 9001 или CMMI.

Совмещение Scrum и последовательной разработки

Лишь небольшое число крупных организаций может позволить себе роскошь выполнения всех своих проектов посредством Scrum. Большинству организаций придется пережить период, в течение которого какие-то проекты будут выполняться с помощью Scrum, а другие — с помощью метода последовательной разработки. Это может быть вызвано тем, что компании было бы нелегко одномоментно перейти к использованию Scrum, как нелегко по ходу выполнения какого-либо проекта перейти от последовательной его разработки к разработке посредством Scrum, а также в силу ряда других причин. Поскольку многим организациям на каком-то этапе приходится сталкиваться с проблемами совмещения Scrum и последовательной разработки, сейчас мы уделим внимание рассмотрению данной темы.

Три сценария взаимодействия

Встречаются разные типы пересечения Scrum и последовательной разработки. Проблемы, возникающие при выполнении проекта, будут зависеть от точек, в которых пересекаются Scrum и последовательная разработка. Преподаватель Scrum Мишель Слайджер (Michele Sliger) описывает три разных сценария, в которых могут взаимодействовать Scrum и последовательная разработка (2006).

Водопад вначале. Пересечение Scrum и последовательной разработки в самом начале проекта обычно возникает, когда организация создает определенные барьеры на пути к утверждению проекта. Для устранения этих барьера Scrum-команда на какое-то время должна забыть о своей неприязни к ведению документации и создать спецификацию, план проекта или какой-либо другой артефакт, который требуется для утверждения проекта. После того как проект типа “водопад вначале” будет утвержден, он выполняется, как обычный Scrum-проект. Слайджер рекомендует следовать совету Алистера Кокбурна (Alistair Cockburn) и создавать документацию на “минимально достаточном” уровне (2000). Слайджер приводит описание команды, которая составила спецификацию, минимально достаточную для утверждения проекта.

Они отложили эту спецификацию в сторонку и лишь изредка обращались к ней в ходе данного выпуска. Однако создание этой спецификации не рассматривалось как бесполезная тратка времени. Члены команды чувствовали, что единое для всех участников проекта представление о продукте, выработанное в результате составления спецификации, пошло им на пользу. К тому же финансовые менеджеры, представленные в совете по утверждению проектов, получили нужную им информацию (2006, 29).

Водопад в конце. Когда Scrum и последовательная разработка пересекаются в конце проекта, это обычно происходит на стадии тестирования. Иногда “водопад в конце” используется потому, что организация решила применить Scrum лишь в порядке эксперимента и оставила тестеров (или специалистов по контролю качества) в виде отдельной группы, которая в конце проекта приступает к проверке и контролю качества продукта. В других случаях “водопад в конце” используется, когда какая-либо сторонняя группа (например, отдел операций) требует проведения определенного тестирования в конце выполнения проекта. Обычной реакцией на требование использования “водопада в конце” является резервирование одного или более спринтов для выполнения соответствующей работы. К окончанию проекта команда уже успевает привыкнуть к новому способу работы с использованием гибкой методологии разработки, поэтому даже в самом конце она продолжает использовать Scrum по максимуму. Иными словами, она продолжает работать спринтами, проводит совещания, посвященные планированию спринтов и ежедневные совещания разработчиков и т.п.

Водопад в tandemе. Возможно, самым сложным способом взаимодействия Scrum и последовательной разработки является “водопад в tandemе”. Характерным его примером является ситуация, когда двум или более командам приходится работать вместе над созданием единого продукта и по крайней мере одна из них использует последовательный подход. Координирование работы и частое общение являются обычно главными источниками проблем, когда требуется “водопад в tandemе”. Команда, применяющая последовательный подход, предпочитает общаться посредством совещаний и документов, которые ограничивают функциональные возможности интерфейсов; Scrum-команда предпочтет оставить интерфейсы неопределенными и нечеткими и общаться неформально, но часто по мере нарастания определенности интерфейсов и обязательств, принимаемых на себя командами.

Scrum-команда, которая оказывается в подобной ситуации, обычно находит полезным приглашать менеджеров “водопадных” проектов на совещания, посвященные планированию спринтов, или на ежедневные совещания разработчиков. Слайджер пишет о своем опыте приобщения менеджеров “последовательных” команд к участию в совещаниях, посвященных планированию спринтов.

Поначалу “водопадные” менеджеры сокрушались, что все эти совещания, посвященные планированию спринтов, лишь ломают их рабочий график. Однако после посещения нескольких таких совещаний они начинали осознавать ценность взаимного обмена информацией, а также облегчения и координации работы (2006, 30).

Три сферы конфликта

Слайджер описывает пересечения между Scrum и последовательной разработкой в плане того, как быть в конкретных ситуациях, когда сталкиваются две непохожие одна на другую методологии. С другой стороны, Барри Бойм (Barry Boehm) и Ричард Тернер (Richard Turner) пишут в большей степени с точки зрения того, как избежать трех типов конфликтов, возникающих при сосуществовании Scrum и последовательного процесса.

- **Процесс разработки.** Конфликт процесса разработки проистекает из различий между Scrum и последовательными процессами.

- **Бизнес-процесс.** Конфликт бизнес-процесса порождается тем, что Scrum-команды и последовательные команды по-разному взаимодействуют с бизнесом. Организация, привыкшая к планам, создаваемым последовательной командой, не знакома с типом планирования, применяемым Scrum-командой.
- **Люди.** Конфликты, связанные с людьми, являются следствием новых ролей, вызванных использованием Scrum, а также следствием акцента Scrum на самоорганизации, командной работе и интенсивном общении (Boehm and Turner, 2005).

Некоторые из этих конфликтов возникают в рамках отдельно взятой команды. Например, Scrum полагается на облегченные требования в форме пользовательских историй или чего-либо вроде этого, тогда как последовательный процесс полагается на более тщательно и подробно документированные требования, которые должны быть сформулированы в самом начале проекта. Проблемы могут возникнуть, когда упреждающая деятельность в рамках гибкой методологии разработки питает последовательную деятельность ниже по потоку, но делает это с меньшей степенью детализации, чем ожидалось.

Остальные конфликты – это конфликты, которые возникают между двумя командами, которые используют разные типы процессов. Мы видим это, например, в необходимости синхронизировать работу команд. Бойм и Тернер пишут о проблеме совместной работы Scrum-команды и последовательной команды.

Если Scrum-команда разрабатывает собственные интерфейсы, то она может налечь на другие части команды риск ведения разработки на основе изменяющегося стандарта. Однако традиционный [последовательный] подход, заключающийся в создании спецификации интерфейса в самом начале проекта, может усложнить Scrum-команде задачу рефакторинга тех или иных частей своего дизайна (2005, 31).

В качестве возможного способа решения этих проблем Бойм и Тернер выдвигают следующие предложения.

- **Выполнять больший объем анализа, чем обычно предполагается методологией Scrum.** Если Scrum-команда рассчитывает на успешное взаимодействие с последовательной командой, то одним из компромиссов, на которые ей придется пойти, является выполнение большего объема предварительного анализа, чем она обычно предпочитает выполнять. Это необходимо для того, чтобы можно было распределить работу между командами, участвующими в проекте, и четко идентифицировать крупные интерфейсы.
- **Выстраивать едва достаточный процесс, вместо того чтобы разделять крупный процесс.** Практика показывает, что когда разделяется крупный процесс, то лишь в редких случаях его удается разделить в достаточной степени. Связанных с этим проблем можно легко избежать, начав с “пустого” процесса, а затем добавляя в него лишь то, что необходимо.
- **Определять архитектуру, в которой было бы место и для Scrum, и для последовательного подхода.** В ходе инициализации проекта и первых нескольких спринтov сосредоточьтесь на выявлении областей разрабатываемой системы, наиболее подходящих для Scrum и последовательного подхода. Области со стабильными, хорошо известными требованиями могут быть разработаны последовательными командами. Области с неопределенными требованиями или области,

где возможны разные подходы к проектированию, должны разрабатываться Scrum-командами.

- **Осваивать способы гибкой методологии разработки, которые работают успешно независимо от применяемого процесса.** Некоторые из способов гибкой методологии разработки хороши независимо от применяемого процесса. Непрерывная интеграция, интенсивное использование автоматизированного тестирования, парное программирование и рефакторинг — все это методы, которые вполне подходят не только для Scrum-проекта, но и для последовательного проекта.
- **Просвещать всех лиц, заинтересованных в успешном выполнении вашего проекта.** Поскольку некоторые из этих лиц будут взаимодействовать и со Scrum-командами, и с последовательными командами, очень важно их просвещение. Им необходимо в достаточной степени понимать каждый процесс, чтобы участвовать в нем, или хотя бы понимать его в той мере, в какой это требуется для исполняемых ими ролей.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Предложите последовательным командам выполнять меньшие пакеты работы. Вместо того чтобы требовать от последовательной команды, чтобы она передавала что-то Scrum-команде или объединяла выполненную ею работу с результатами работы Scrum-команды каждые два-три месяца, мы хотели бы, чтобы последовательная команда делала это каждые две-три недели.
- Попробуйте применять правило, в соответствии с которым Scrum-команда не должна включать в спринт тот или иной элемент журнала запросов на выполнение работ, если для реализации этого элемента требуется, чтобы во время того же спринта последовательная команда выполнила соответствующую работу лишь частично. Включайте в спринт тот или иной элемент журнала запросов на выполнение работ только в том случае, если последовательная команда уже завершила свою работу по этому элементу.
- Предложите, чтобы кто-либо из членов последовательной команды был также прикомандирован к Scrum-команде. Этот сотрудник должен в обязательном порядке посещать совещания, посвященные планированию спринта, обзоры, ретроспективы и ежедневные совещания разработчиков.

Могут ли Scrum и последовательная разработка сосуществовать вечно

Мнения о том, могут ли Scrum и последовательная разработка сосуществовать вечно, разделились. Несомненно, имеются организации, в которых такое сосуществование наблюдается уже сегодня. Также есть организации, которые успешно обеспечивали сосуществование нескольких процессов негибкой методологии разработки в прошлом. Однако имеется ли в Scrum какая-либо фундаментальная особенность, которая мешала бы этой методологии вечно сосуществовать с каким-либо последовательным процессом? Мишель Слайджер полагает, что такая фундаментальная особенность у Scrum есть.

Ранее я всегда утверждала, что компании могут до бесконечности сочетать гибкий и традиционный подходы к разработке, если у них нет желания применять гибкую методологию разработки к каждому своему проекту. Но после того, чему я стала свидетелем в прошлом году, я вижу, что была не права. Сейчас я считаю, что компании должны в конце концов сделать окончательный выбор в пользу того или иного подхода. Я называю этот поворотный пункт “точкой подвисания” (high-centering). Точка подвисания — это термин, применяемый к транспортным средствам с четырьмя ведущими колесами. Точка подвисания возникает, когда ваш джип наезжает на высоко выступающую остроконечную возвышенность (это может быть, например, большой камень) и повисает на ней, балансируя на своем шасси. При этом ни одно из колес автомобиля не может зацепиться за какую-либо поверхность, чтобы сдвинуть автомобиль вперед или назад. Когда компании, постепенно взираясь на выступающую возвышенность гибкой методологии разработки, достигают определенной точки и начинают балансировать на своем шасси, они обязаны принять осознанное и понятное всем сотрудникам решение двигаться вперед. В противном случае их команды соскользнут назад и снова окажутся в водопаде.

Я склонен согласиться с Мишель Слайджер. В крупной организации временное сосуществование Scrum с последовательным процессом зачастую является вынужденной мерой. Но важно помнить, что гибкая методология разработки — это вовсе не конечный пункт назначения. Применение гибкой методологии разработки означает непрерывное совершенствование. Когда организация достигает все большего и большего совершенства в деле применения гибкой методологии разработки, конфликты между Scrum и последовательной разработкой становятся все более болезненными. Если не устранить источники этих конфликтов, “организационная сила тяжести” потянет эту организацию обратно к тому процессу разработки программного обеспечения, который применялся ею до внедрения Scrum. Несколько безобидных методов гибкой методологии разработки, таких, как ежедневные совещания разработчиков или непрерывная интеграция, могут продолжать использоваться, но организация не сможет добиться неоспоримых преимуществ, которые обеспечивает эффективное применение гибкой методологии разработки.

Надзор за выполнением проекта

Одной из причин, заставляющих многие организации применять последовательный подход к разработке программного обеспечения, является естественное соответствие между строго определенной последовательностью фаз разработки и потребностью в надзоре за выполнением проекта. Цель надзора за выполнением проекта, который обычно называется контролем, состоит в том, чтобы гарантировать выполнение проекта в строгом соответствии с заранее выбранным направлением. Эффективный контроль, например, позволяет выявить проект, который превысил свой бюджет; это может послужить поводом к дискуссии о том, не следует ли вообще свернуть данный проект. Эффективный контроль проекта может также позволить выявить продукт, который отошел слишком далеко от своих первоначальных целей, или проект, который отклоняется от определенного архитектурного стандарта, либо проверить соответствие проекта ряду подобных соображений верхнего уровня, важных для данной организации.

Надзор за выполнением проекта не относится к числу новых концепций, но самым естественным для него является “процесс последовательных шлюзов” (stage-gate process), придуманный доктором Робертом Купером (Robert Cooper) и представленный на рис. 19.1. Центральная идея заключается в том, что после каждой стадии процесса разработки проект направляется в определенный шлюз. Каждый такой шлюз действует как формальный пункт проверки проекта: чтобы получить право продвигаться дальше, проект должен быть одобрен; в противном случае его отправляют на предыдущую стадию для повторной разработки или вообще сворачивают его выполнение (2001).

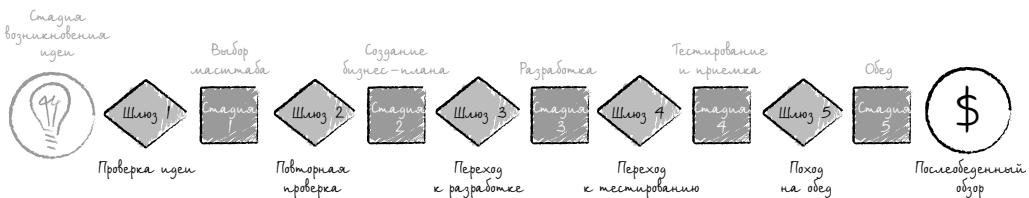


Рис. 19.1. Процесс последовательных шлюзов (stage-gate process). Stage-Gate® является зарегистрированной торговой маркой Product Development Institute

Шлюзы и пункты проверки могут встретиться команде разработчиков программного обеспечения в разные моменты — при первоначальном обзоре планов относительно масштаба, бюджета и графика выполнения работ, при обзоре архитектурных и проектных решений, при обзоре, который должен подтвердить готовность соответствующего приложения к системному тестированию или приемочному тестированию, проводимому заказчиком, при проверке возможности передачи продукта поддерживающей организации и т.д. Эти проверки зачастую не способствуют желанию команды разработчиков программного обеспечения использовать Scrum, поскольку не подходят для работы, выполняемой инкрементным способом. Например, Scrum-команда, которая предоставляет возможность дизайну системы выкристаллизовываться в процессе выполнения проекта, столкнется с проблемами при прохождении через какой-либо из ранних пунктов проверки, где проверяются адекватность и корректность архитектуры системы.

Первый шаг в приведении во взаимное соответствие необходимости осуществлять надзор за выполнением проекта и желания использовать Scrum заключается в том, чтобы уяснить, что надзор за выполнением проекта и управлением проектом — это не одно и то же. Желательно отделить надзор за выполнением проекта от управления последним. Но, осуществляя такое разделение, мы хотели бы получить возможность формировать пункты проверки верхнего уровня для обеспечения необходимого надзора за выполнением проекта, предоставив в то же время команде свободу управления самой собой и собственной деятельностью по выполнению проекта в соответствии с принципами гибкой методологии разработки.

В качестве подтверждения того, что надзор за выполнением проекта вовсе не обязательно является злом, допустим, что вас неожиданно назначили президентом или главным исполнительным директором вашей компании. Вы как новый босс хотите разобраться в том, как обстоят дела с выполнением основных проектов вашей компании. Для этого вы, возможно, установите новое правило, в соответствии с которым

вы лично будете утверждать начало выполнения любого проекта, стоимость которого превышает определенную сумму. Кроме того, поскольку вы намерены лично участвовать в как можно большем количестве обзоров спринтов, вы хотите, чтобы по любому проекту, продолжительность выполнения которого превышает три месяца, вам каждые три месяца предоставляли двухстороннюю подборку важнейшей информации. Такой подход — весьма упрощенная модель надзора за выполнением проекта, которую следовало бы внедрить в вашей компании. Так что возражения вызывает не сам по себе надзор за выполнением проекта, а ситуация, при которой надзор за выполнением проекта начинает влиять на то, *как мы выполняем* проекты.

Выполнение Scrum-проектов, когда надзор не связан с гибкой методологией разработки

Поскольку лишь немногие организации отваживаются поначалу зайти настолько далеко, чтобы полностью пересмотреть свои текущие подходы к надзору за выполнением проектов, командам требуются те или иные способы адаптации к негибкому надзору за выполнением проекта, применяемому их организацией. В таких случаях командам могут пригодиться перечисленные ниже меры.

Заранее согласуйте и сформулируйте ожидания. Несомненно, при осуществлении первого Scrum-проекта, который будет подвергнут традиционному для вашей компании процессу надзора за его выполнением, возникнет ряд проблем. Почти наверняка некоторые проблемы Scrum-команде будет очень нелегко решить. Например, Scrum-команде не удастся разработать детальный проект системы *до* получения разрешения начать кодирование, поскольку проектирование и кодирование должны выполняться параллельно. Единственным способом решения этой проблемы является заблаговременное проведение переговоров между командой и лицами, осуществляющими надзор за выполнением проектов. Чем большую поддержку в этом деле получит команда и чем выше будет уровень этой поддержки в организационной иерархии, тем лучше. Команде нет нужды настаивать на проведении постоянных изменений в политике надзора. Такие изменения могут быть осуществлены в виде однократного эксперимента.

Приведите свою отчетность в соответствие с текущими ожиданиями. Советы по рассмотрению проектов или комитеты по надзору, которые осуществляют надзор за выполнением проектов, исходят в своих решениях из определенных ожиданий относительно того, какую информацию должны предоставить исполнители проекта по каждому пункту проверки. Если они рассчитывают получить график Ганта, предоставьте им график Ганта. Если, однако, вы можете постепенно изменить эти ожидания, предоставляя дополнительную информацию, в большей степени соответствующую гибкой методологии разработки, попытайтесь сделать это. Если советы по рассмотрению проектов или комитеты по надзору согласны принимать к рассмотрению графики выполнения оставшихся работ, предоставьте им такие графики. Возможно, вы захотите заодно предоставить и отчет, в котором указывается, сколько раз сервер сборки начал непрерывно интегрируемые сборки, и приводятся ссылки на тысячи (возможно, даже на десятки или сотни тысяч) тестовых прогонов, которые были выполнены.

Вовлеките их в свой процесс. Scrum-команды могут дополнить менее детальные формальные пункты надзора, пригласив членов комитета по надзору принять участие

в регулярных совещаниях, которые они будут проводить. Команды в Yahoo! проводят обзоры с участием членов комитета по обзору архитектурных решений. Габриэль Бенефилд (Gabrielle Benefield), бывшая директор разработки продуктов по гибкой методологии в Yahoo!, вспоминает, как решали эту проблему ее первые команды, использовавшие гибкую методологию разработки.

Команды, использовавшие гибкую методологию разработки, с самого начала приглашали членов комитета по обзору архитектурных решений участвовать в обзورах спринтов, проводимых командами. Несмотря на это они проходили впоследствии один формальный пункт надзора, однако к тому времени большинство важнейших вопросов уже было решено. Такой подход был гораздо менее болезненным, а атмосфера доверия и сотрудничества возникала уже в самом начале выполнения проекта.

Мне нравится дополнять хорошо известный метод управления путем обхода объектов (*management by walking around*) управлением путем личного присутствия (*management by standing around*). Предлагайте менеджерам и руководителям, надзирающим за выполнением проекта, посещать ежедневные совещания разработчиков, где они смогут лично узнать, как обстоят дела с выполнением проекта. Такой же переход от документов к обсуждениям, который является следствием работы с пользовательскими историями, должен произойти с проектной отчетностью. Предлагайте людям лично посещать команду или участвовать в совещаниях, которые она проводит, чтобы увидеть собственными глазами, чем занимается команда и что она создает.

Рассказывайте об успехе. Ничто не убеждает лучше, чем успех. Сделайте все от вас зависящее, чтобы “протащить” один-два первых проекта через упрощенные или сокращенные пункты проверки. Затем указывайте на успех этих проектов как наглядное свидетельство того, что к будущим проектам также следует применять эти облегченные процедуры надзора. Габриэль Бенефилд указывает, что, “как только хотя бы одна-две команды, использующие гибкую методологию разработки, продемонстрируют благоприятные результаты, вы сформируете доверие. После этого вы сможете настаивать на том, чтобы и в дальнейшем процесс надзора за выполнением проектов применялся к командам, использующим гибкую методологию разработки, с учетом их особенностей”.

Концепции гибкости и надзора за выполнением проектов не являются принципиально несовместимыми. И то, и другое представляет собой попытку улучшить конечный продукт. Методология Scrum пытается достичь этого результата посредством тесного сотрудничества и коротких, строго ограниченных по времени циклов. Применение надзора за выполнением проектов пытается достичь этого же результата посредством того, что можно было бы назвать пунктами проверки “сравнить и утвердить (или отвергнуть)”, в которых продукт или проект сравнивается с некой совокупностью желаемых атрибутов. Однако, преследуя схожие цели, Scrum и надзор за выполнением проектов достигают этих целей совершенно по-разному. Проблемы в совмещении этих двух концепций возникают именно в связи с использованием ими совершенно разных путей. К счастью, достаточно лишь небольших компромиссов с той и другой стороны (дополненных использованием рекомендаций, которые были изложены в этом разделе) — и можно получить успешное сочетание гибкости и надзора.

Соответствие стандартам

Не каждая команда (или даже отдел разработки программного обеспечения) может позволить себе роскошь обладания полным управлением своим процессом разработки. Например, при разработке программного обеспечения с использованием субподрядчиков заказчики нередко выдвигают условие, чтобы этими субподрядчиками были организации, имеющие сертификат пятого уровня CMMI. Это означает, что указанные разработчики программного обеспечения должны использовать определенные “передовые” методы. Кроме того, отдельные продукты, в которых широко применяются компьютерные программы, поставляются в промышленные отрасли и должны соответствовать таким стандартам, как ISO 9001. Компании, производящие устройства для медицинской отрасли, должны соответствовать стандарту ISO 13458. Акционерные компании открытого типа, действующие на территории Соединенных Штатов Америки, должны соответствовать требованиям Закона Сарбейниса–Оксли (Sarbanes–Oxley Act). Список можно продолжить.

Ни один из этих стандартов не предписывает жизненный цикл, который бы полностью противоречил методологии Scrum. Некоторые из них, однако, в определенной мере противоречат этой методологии, поскольку настаивают на использовании последовательного процесса. Поскольку соответствие подобным стандартам лишь в редких случаях не является обязательным, Scrum-командам приходится заботиться о том, как соблюсти эти стандарты (начиная в ряде случаев с вопроса о том, возможно ли вообще соблюсти эти стандарты при использовании Scrum-процесса). В этом разделе мы рассмотрим, как Scrum-команды обеспечивают соответствие стандартам ISO 9001 и CMMI, двум наиболее распространенным стандартам, с которыми Scrum-команды уже научились сосуществовать. Проанализировав ISO 9001 и CMMI, можно обобщить некоторые стратегии обеспечения соответствия, которые могут оказаться полезными в других ситуациях, когда речь идет о необходимости соблюдать те или иные стандарты.

ISO 9001

Международная организация по стандартизации (International Organization for Standardization — ISO) поддерживает стандарт 9001 (полностью обозначается как ISO 9001:2000 или ISO 9001:2008; в том и другом случаях указывается год появления конкретной версии этого стандарта). Сертификация ISO 9001 не призвана гарантировать, что продукция соответствующей организации вышла на определенный уровень качества. Скорее эта сертификация указывает на то, что соответствующая организация придерживается определенной совокупности формальных методов при разработке своих продуктов. Значительная доля мер по соблюдению стандарта ISO 9001 связана с созданием определенной *системы управления качеством*, которая обычно представляет собой пространный документ или совокупность веб-страниц, описывающих методы обеспечения качества, которых придерживается соответствующая организация.

Primavera Systems, разработчик систем управления проектами и портфелями, создавала свою систему управления качеством на протяжении более чем десяти месяцев. Компания провела 30 семинаров, чтобы документировать уже используемые ею процессы, причем каждый такой семинар предусматривал межфункциональное представительство разработчиков.

К тому времени когда Primavera Systems инициировала меры по приведению своих процессов в соответствие с ISO 9001, эта компания уже располагала солидным опытом применения гибкой методологии разработки. В подобной обстановке было бы вполне естественно, если бы сотрудники Primavera Systems озабочились потерей гибкости, связанной с внедрением стандарта ISO 9001. Билл Макмайкл (Bill McMichael), сотрудник Primavera Systems, и Марк Ломбарди (Marc Lombardi), консультант, хорошо разбирающийся в ISO 9001, совместно работали над этим вопросом и пришли к выводу, что документация не снижает гибкости.

Возникали сомнения по поводу нарушения принципа “работоспособное программное обеспечение важнее, чем исчерпывающая документация”. Наша идея заключалась в том, чтобы обеспечить такой объем документации, который был бы достаточным для ее использования в качестве справочного материала, а также вспомогательного средства для обеспечения соблюдения существующих процессов (2007, 264).

Опыт компании Primavera Systems, которой потребовался примерно год, чтобы достичь уровня, при котором она смогла пройти аудит на сертификацию ISO 9001, подтверждается Грэхемом Райтом (Graham Wright). Райт, работающий наставником в компании Workshare (Лондон), внес немалый личный вклад в успешную сертификацию ISO 9001 своей организации, для проведения которой понадобилось немногим более года и которая началась спустя 13 месяцев после того, как компания Workshare внедрила у себя экстремальное программирование. Райт указывает, что “для получения сертификации ISO 9001 не потребовалось вносить никаких изменений в применяемые нами методы XP” (2003, 47).

Мой личный опыт использования ISO 9001

Описанный выше опыт внедрения ISO 9001 соответствует моему личному опыту. В 2002 году я руководил командой разработчиков, организация которых решила получить сертификат ISO 9001. Поскольку в то время моя команда имела меньший опыт применения Scrum, чем сотрудники Primavera Systems, я воспользовался другим подходом и самостоятельно написал большую часть нашей системы управления качеством.

За несколько месяцев до проведения официального аудита мы встретились с нашим аудитором, чтобы ознакомить его с тем, как мы разрабатываем программное обеспечение, и выяснить его конкретные ожидания в отношении нас. Поскольку это было в начале 2002 года, аудитор ничего не слышал о Scrum, а ни у кого из нас еще не было опыта применения ISO 9001. По результатам той встречи мы внесли несколько изменений в используемый нами процесс. Первое из этих изменений было связано с тем, что аудитор твердо заявил нам о том, что у нас нет шансов пройти аудит, если мы будем продолжать записывать пользовательские истории на индексных карточках. Он не возражал против формата пользовательских историй, но настаивал на необходимости создания “документа”. Мы согласились переносить пользовательские истории методом фотокопирования на бумагу блокнотного размера и хранить их в скоросшивателе с указанием порядкового номера на каждой пользовательской истории. Вторая претензия заключалась в следующем: несмотря на то, что наши неформальные процессы проектирования не вызывали никаких возражений, нам придется создавать больший объем проектной документации. Наш аудитор предложил нам делать

фотографии доски (обычной школьной доски) после каждого обсуждения проекта и хранить эти фотографии вместе с экземплярами любых рукописных заметок, которые вели для себя участники таких обсуждений.

Впоследствии мы успешно прошли процедуру аудита с нашим аудитором. Самым интересным в этом его повторном визите было то, какое глубокое впечатление произвели на него наши процессы автоматизированного тестирования. В дополнение к серверу сборки, выполняющему непрерывную интеграцию, мы проводили официальную ежевечернюю сборку, которая включала тысячи тестов, написанных главным образом в JUnit. Мы продемонстрировали результаты ежевечерних сборок за последний месяц; каждый вечер сборка и все тесты выполнялись успешно. Какой бы замечательной ни была эта команда, ей все же повезло, что за весь этот период ни один тест не оказался неудачным. Аудитор взглянул на все эти свидетельства успешных тестов и спросил: “Откуда вам известно, что эти тесты были удачными? Возможно, они вообще не выполняются и все дело лишь в том, что система формирования отчетов о тестировании построена таким образом, что в любом случае указывает успешное завершение тестов”. Разумеется, нам это было известно, поскольку в течение дня неизбежно встречались случаи неудачного завершения тестов. Зато вечером случаев неудачного завершения тестов не было. Но для нашего аудитора этого было недостаточно. Он настоял на том, чтобы мы включили в каждую вечернюю сборку какой-либо неудачно завершающийся тест. Мы добавили

```
assertTrue(false);
```

Этот тест завершается неудачно, поскольку “ложь” не является “истиной”. После добавления нашего неудачного теста мы прошли аудит на соответствие стандарту ISO 9001.

Я не считаю, что какое-либо из перечисленных мною изменений помогло нам создавать более качественное программное обеспечение; однако, с другой стороны, они не добавили и каких-либо существенных текущих накладных расходов. Документирование буквально всей информации о нашем процессе требовало определенного времени, но это были одноразовые меры (с планируемыми ежегодными обновлениями), причем основное бремя этих перемен легло именно на мои плечи. Хуан Габардини (Juan Gabardini), который пользовался Scrum в двух организациях, сертифицированных на соответствие стандарту ISO 9001, приходит к такому же выводу.

Компании пришлось нести в связи с этим определенные издержки, но для команды это оказалось не так уж плохо. Я не хочу сказать, что это было совершенно безболезненно. Вам понадобится прибегнуть к услугам какого-либо консультанта по ISO, обладающего достаточно широким мышлением, который поможет вам минимизировать издержки и не утратить при этом ни одного из преимуществ, которые обеспечивает гибкая методология разработки (2008).

Интегрированная модель технологической зрелости организации (CMMI)

Практически сразу же после того, как из первоматерии возник первый проект, выполненный с помощью гибкой методологии разработки, компании начали живо интересоваться, совместимы ли гибкие методологии разработки с

интегрированной моделью технологической зрелости организации (Capability Maturity Model Integration — CMMI), разработанной Институтом программной инженерии (Software Engineering Institute). Рассматриваемые как мера использования организацией некоего процесса (или хотя бы мера определенности этого процесса), CMMI и ее предшественница, модель технологической зрелости организации, занимающейся разработкой программного обеспечения (Software Capability Maturity Model — SW-CMM), зачастую считаются авторитетными средствами разработки программного обеспечения и антитезой гибкой методологии разработки. Ричард Тернер (Richard Turner), который входил в состав первой команды, разрабатывавшей CMMI, и профессор Апурва Джайн (Apurva Jain) утверждают, что “несмотря на существенные различия, указания на несовместимость между подходами CMMI и гибкой методологией разработки представляются несколько преувеличенными” (2002).

Ричард Тернер — не единственный из авторов СММ, которые рассматривали применимость СММ к проектам, выполненным с помощью гибкой методологии разработки. Марк Полк (Mark Paulk), ведущий автор исходной модели SW-CMM, сравнивал экстремальное программирование с 18 ключевыми областями процесса первоначальной SW-CMM. Мнение Полка сводилось к тому, что экстремальное программирование частично или в основном удовлетворяет 10 из 13 областей, необходимых для достижения уровня 3, и не является препятствием к остальным трем.

Таким образом, можно считать, что СММ и XP дополняют друг друга. SW-CMM в общих терминах рассказывает организациям, что делать, но не рассказывает, как делать. XP — это совокупность передовых методов, которая заключает в себе достаточно конкретную информацию типа “как делать” (т.е. метод практической реализации) для среды определенного типа. Методы XP могут быть совместимы с методами СММ (целями или КРА), даже если они не полностью согласуются с ними (2001, 26).

На возможность сочетания гибкой методологии разработки с CMMI указывает не только теория, но и практика. Сейчас многие компании успешно сочетают использование гибкой методологии разработки с SW-CMM или CMMI. Эрик Бос (Erik Bos) и Крист Вриенс (Christ Vriens) из Philips Research возглавляли один из первых проектов, выполненных с помощью гибкой методологии разработки, который был документирован с целью проведения аудита СММ. Они утверждают, что “особое впечатление на их экспертов-консультантов произвела прозрачная, легкодоступная и единообразная проектная информация” (2004).

Джо Фекаротта (Joe Fecarotta), проект которого, выполненный с помощью гибкой методологии разработки, был оценен как соответствующий уровню 3 CMMI, также пришел к выводу о совместимости CMMI и гибкой методологии разработки. Он говорит, что “CMMI и соответствующие аудиты не принуждали к использованию какой-то конкретной методологии, а лишь пытались помочь группе воспользоваться самыми эффективными методами” (2008).

Гибкие методологии разработки, подобные Scrum, внедрялись и в организациях, уже сертифицированных на соответствие уровню 5 CMMI. В компании Systematic (независимый разработчик программного обеспечения в Дании и Великобритании) трудится свыше 400 человек. Systematic разрабатывает программное обеспечение для организаций, работающих на оборону, здравоохранение, производство и сервис.

Поработав примерно два года на уровне 5, Systematic решила внедрить у себя Scrum. Компания указывает на то, что CMMI и Scrum удачно дополняют друг друга.

Сейчас Scrum сокращает каждую категорию работ (дефекты, переделка уже выполненной работы, совокупный объем требуемой работы и непроизводительные издержки процесса) почти на 59% в сравнении с нашей предыдущей реализацией уровня 5 CMMI, при этом обеспечивая тот же уровень дисциплины процесса (Sutherland, Jakobsen, and Johnson, 2007, 273).

Включение Scrum в процесс уровня 5 CMMI компании Systematic демонстрирует решение типичной проблемы, связанной с реализациями CMMI. Стремясь обеспечить соответствие конкретному уровню CMMI, многие организации забывают, что их конечная цель — повысить как эффективность разработки программного обеспечения, так и качество создаваемых ими продуктов. Вместо этого они сосредоточиваются на устраниении предполагаемых недостатков согласно документации CMMI, нимало не заботясь о том, приведут ли эти изменения к улучшению процесса разработки или продуктов, получаемых с помощью этого процесса. Проблему можно устранить, когда цели CMMI сочетаются с образом мышления, ориентированным на создание высокой ценности и присущим тем, кто использует методологию Scrum, т.е. образом мышления, который можно выразить фразой “Что вы сделали для меня в последнее время?” Джекф Сазерленд (Jeff Sutherland), Карстен Якобсен (Carsten Jakobsen) и Кент Джонсон (Kent Johnson), которые принимали участие во внедрении Scrum в компании Systematic, называют сочетание Scrum и CMMI “волшебным снадобьем”.

В результате сочетания Scrum и CMMI возникает “волшебное снадобье”, в котором Scrum гарантирует эффективную реализацию процессов и возможность осуществления изменений, тогда как CMMI гарантирует, что будут учитываться все значимые процессы (2007, 272).

Обеспечение соответствия

Мы уже выразили свою твердую убежденность в том, что методология Scrum совместима по крайней мере с ISO 9001 и CMMI. Эта убежденность имеет как теоретические, так и эмпирические основания. Обратимся к конкретным мерам, которые можно предпринять, чтобы обеспечить в вашей организации успешное сочетание ISO 9001 и CMMI с методологией Scrum.

- **Позаботьтесь о надлежащем ведении своего журнала запросов на выполнение работ.** Общей нитью, связывающей все проекты, при реализации которых приходилось соблюдать требования к обеспечению соответствия, является то, что важную положительную роль в успехе всех этих проектов сыграло надлежащее ведение журнала запросов на выполнение работ. Правильное ведение журнала запросов на выполнение работ не восполняет потребность в заглавовременном и подробном изложении всех требований к продукту. Однако команды, которые позаботились о надлежащем ведении своего журнала (позволяющем уточнять содержащуюся в нем информацию и дополнять ее все новыми и новыми подробностями, как описано в главе 13, “Журнал запросов на выполнение работ”), пришли к выводу, что это способствует обеспечению требуемого соответствия стандартам.

- **Отразите работу по обеспечению соответствия стандартам в журнале запросов на выполнение работ.** Если для обеспечения соответствия стандартам необходимо создать какой-либо документ или другой артефакт, то эту работу нужно отразить в журнале запросов на выполнение работ. Это не только гарантирует, что команда не забудет о данной работе, но и позволит постоянно держать в поле зрения стоимость обеспечения соответствия стандартам.
- **Рассмотрите возможность использования контрольных перечней.** Разработчики ряда проектов сообщают о большой пользе, которую приносило им использование контрольных перечней. Важно, чтобы такие контрольные перечни не содержали новых обязательных этапов. Контрольные перечни, напротив, должны включать этапы, уже выполняемые командой, а цель таких контрольных перечней заключается лишь в том, чтобы доказать аудитору или аттестующему, что соответствующие работы действительно выполняются. Например, в упоминавшейся выше компании Systematic, сертифицированной на соответствие уровню 5 CMMI, использовался контрольный перечень реализации историй (объемом в одну страницу), который начинался с указания, была ли соответствующая история оценена, и заканчивался интеграцией данной истории в систему. Сформулированное командой определение того, что подразумевается под “выполнено” (соответствующее описание содержится в главе 14, “Сprintы”), можно без труда преобразовать в контрольный перечень.
- **Автоматизируйте.** Автоматизация сборок и тестов играет важную роль в успехе любого Scrum-проекта. Она вдвойне актуальна для проектов с требованиями к соответствию.
- **Используйте инструменты управления проектами.** Для большинства стандартов соответствия важным фактором является постоянное слежение за проектом. Хотя лично я предпочитаю материальные артефакты — индексные карточки, надписанные от руки, а также крупноформатные плакаты, развешиваемые на стенах, — командам, которым приходится заботиться о соблюдении требований к соответствию, следует как минимум рассмотреть возможность использования одного из программных инструментов управления проектами.

См. также С обзорами инструментов управления проектами, выполняемыми с применением гибкой методологии разработки, можно ознакомиться на веб-сайте www.userstories.com.

- **Продвигайтесь медленно, но уверенно.** За одну ночь невозможно изучить какой-либо важный элемент процесса (например, руководство по системам управления качеством ISO 9001). Поэтому следует воспользоваться одним из самых эффективных методов Scrum-команд и сделать это инкрементно. Постепенно перерабатывайте руководство по системам управления качеством таким образом, чтобы оно все больше соответствовало гибкой методологии разработки. Стандарт ISO 9001 направлен главным образом на то, чтобы обеспечить следование компанией своей собственной системе управления качеством. Поэтому компания вполне может переработать эту систему так, чтобы она поддерживала Scrum.
- **Работайте со своим аудитором.** Если позволяет ситуация, встретьтесь со своим аудитором заранее. Обсудите с ним ваш процесс разработки программного

обеспечения в неформальной обстановке и попросите его высказать свои замечания по этому процессу. По мере возможности обращайтесь к опытным аудиторам, которые понимают, что своеобразие используемого вами процесса еще не означает, что он не в состоянии достичь целей соответствующего стандарта.

- **Воспользуйтесь посторонней помощью.** Если никогда раньше вы не пытались получить сертификат того типа, который вас сейчас интересует, воспользуйтесь услугами стороннего консультанта, которому уже приходилось заниматься подобными вопросами. Если у вас еще нет достаточного опыта применения Scrum, воспользуйтесь услугами опытного Scrum-мастера. Наличие опытного специалиста и по тому, и по другому вопросу, несомненно, поможет вам.

Что дальше

Вряд ли можно рассчитывать на то, что методология Scrum будет применяться в идеальной среде, свободной от вмешательства окружающего реального мира. В этой главе мы рассмотрели три типа таких вмешательств: необходимость работать с каким-либо другим последовательно управляемым проектом (или выполнять часть Scrum-проекта последовательным методом); необходимость работать в рамках некоторой системы корпоративного управления; и необходимость соблюдать определенные законы, нормативы или стандарты. В следующей главе мы продолжим рассмотрение проблем, которые препятствуют успешному применению Scrum. Мы рассмотрим возможные способы влияния на Scrum-команды и Scrum-проекты других групп или подразделений организаций, включая отдел кадров, отдел технического обеспечения и отдел управления проектами.

Дополнительная литература

Boehm, Barry, and Richard Turner. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, September/October, 30–39.

В 1988 году Барри Бойм представил спиральную модель, одну из первых эффективных альтернатив “водопадному” процессу. В этой книге, соавтором которой является Ричард Тернер, Барри Бойм представляет точку зрения, суть которой заключается в сосуществовании гибкого и “дисциплинированного” процессов, которые при необходимости могут использоваться для соответствующего проекта один в сочетании с другим (конечно, с учетом определенных факторов риска для этого проекта).

Glazer, Hillel, Jeff Dalton, David Anderson, Mike Konrad, and Sandy Shrum. 2008. *CMMI or agile: Why not embrace both!* Software Engineering Institute at Carnegie Mellon, November. <http://www.sei.cmu.edu/pub/documents/08.reports/08tn003.pdf>.

В этой статье излагается мнение, суть которого заключается в том, что передовые методы CMMI и гибкие методологии разработки не противоречат друг другу и могут вполне успешно сочетаться друг с другом.

McMichael, Bill, and Marc Lombardi. 2007. ISO 9001 and agile development. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 262–265. IEEE Computer Society.

В этом коротком отчете об опыте применения гибкой методологии разработки в условиях действия стандарта ISO 9001 содержатся конкретные советы, основанные на опыте, полученном компанией Primavera в ходе внедрения стандарта ISO 9001. Этот опыт особенно ценен тем, что к моменту внедрения стандарта ISO 9001 компания Primavera уже использовала Scrum-процесс.

Paulk, Mark. 2001. Extreme programming from a CMM perspective. *IEEE Software*, November, 19–26.

Эта статья, опубликованная в 2001 году, несколько устарела, поскольку экстремальное программирование в ней сравнивается с моделью СММ, на смену которой уже пришла модель СММI. Однако поскольку автор этой статьи является ведущим автором СММ, к его мнению все же стоит прислушаться.

Sliger, Michele. 2006. Bridging the gap: Agile projects in the waterfall enterprise. *Better Software*, July/August, 26–31.

Авторы этой статьи доказывают, что гибкие и “водопадные” процессы могут успешно сосуществовать в одной и той же организации. В статье излагаются конкретные советы относительно выполнения “водопадного” процесса до гибкого процесса, в конце гибкого процесса или одновременно с ним.

Sutherland, Jeff, Carsten Ruseng Jakobsen, and Kent Johnson. 2007. Scrum and CMMI level 5: The magic potion for code warriors. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 272–278. IEEE Computer Society.

Авторы этого отчета об одном высокопроизводительном проекте пришли к выводу, что сочетание Scrum и СММI приводит даже к лучшему результату, чем применение того или другого по отдельности. В отчете приводятся рекомендации о том, как добиться успешного сочетания Scrum и СММI.

Глава 20

Кадры, техническое обеспечение и отдел управления проектами

Чтобы достичь долговременного успеха в деле использования Scrum, последствия перехода разработчиков к использованию гибкой методологии разработки должны стать фактом, который необходимо учитывать другим отделам организации. Если этого не сделать, организационная гравитация — т.е. влияния, которые придали организации тот вид, который она имела до начала перехода к Scrum — возьмет свое. Я лично был свидетелем того, как переход к Scrum в некоторых организациях тормозился или вообще прекращался, поскольку они игнорировали влияние перехода к использованию гибкой методологии разработки на группы, не имеющие непосредственного отношения к разработке. Игнорирование этого влияния приводит к возникновению перечисленных ниже проблем с теми или иными отделами и группами.

- **Отдел кадров.** Scrum-команды работают чрезвычайно успешно до момента очередной ежегодной аттестации. Здесь разработчики начинают понимать, что их снова будут оценивать, основываясь исключительно на личных достижениях каждого из них. Ежегодная аттестация, конечно же, может включать оценку того, насколько хорошо аттестуемый сотрудник взаимодействует с другими членами команды, но в конечном счете повышение по службе определяется индивидуальным вкладом работника.
- **Группа технического обеспечения.** Гораздо легче придерживаться гибкой методологии разработки, когда вся команда локализована в одной комнате. Но когда группа технического обеспечения чинит препятствия такому сосредоточению всей команды в одном помещении или когда она не позволяет членам команды развешивать на стенах графики выполнения оставшихся работ и другие важные проектные данные, “боевой дух” команды падает. В этом случае — когда членам команды начинает казаться, что все настроены против них — бывает очень легко заставить команду совершенствовать свою технику применения Scrum.

- **Отдел управления проектами (Project Management Office — PMO).** Не задумываясь над тем, как выполняемый Scrum-командой проект соотносится с требованиями PMO, команда начинает полностью игнорировать “бумажную волокиту” и перестает придерживаться определенного процесса разработки. Таким образом, команда собственными руками создает себе врага в лице PMO — группы, которая и без того относилась с большим подозрением к первым робким экспериментам с применением Scrum в своей организации. PMO реагирует на это, убеждая руководство отдела, что методология Scrum хороша лишь в той степени, в какой она дополняется внушительной кипой документов и предписанных методов.

Когда внедрение Scrum ошибочно рассматривается как изменение, касающееся исключительно разработчиков, организационная гравитация, создаваемая отделами, не имеющими отношения к информационным технологиям, может потянуть группу разработки обратно, на ее исходные позиции. В этой главе мы рассмотрим меры, которые можно принять, чтобы помочь усилиям по переходу вашей организации к Scrum достичь “второй космической скорости” отхода от применения традиционного способа разработки, которая оказалась бы достаточной для невозврата группы разработки на ее исходные позиции. В частности, мы рассмотрим влияние Scrum на три упоминавшиеся выше группы: отдел кадров, группу технического обеспечения и отдел управления проектами.

Отдел кадров

Многие проблемы, касающиеся отдела кадров, являются результатом перехода к коллективной ответственности. В своей книге *The Wisdom of Teams* Катценбах (Katzenbach) и Смит (Smith) описывают, почему это так трудно.

Большинство организаций изначально предпочитают индивидуальную ответственность коллективной (групповой, командной). Должностные инструкции, схемы оплаты труда, карьерные возможности и оценки достигнутых результатов сосредоточиваются на отдельных работниках... Наша культура отдает предпочтение индивидуальным достижениям и вызывает у нас неприятие ситуации, когда наши карьерные устремления попадают в зависимость от результатов деятельности других людей... Сама мысль о переносе акцента с индивидуальной ответственности на коллективную вызывает у нас ощущение дискомфорта (1993, 3–4).

Рассмотрим в качестве примера человека по имени Чак. Когда я сказал Чаку и его товарищам по команде, что хотел бы, чтобы они в течение нескольких спринтов попробовали воспользоваться методом парного программирования, Чак встал и сказал: “Я немедленно сообщу об этом в отдел кадров!”, после чего покинул комнату, где мы проводили ретроспективу спринта. Что он собирался сделать? Попытаться уволить меня? Но я не был сотрудником этой организации, а потому терялся в догадках. Всматриваясь в лица остальных членов команды, я увидел на них выражение озадаченности. Тем не менее мы продолжили ретроспективу спринта.

Несколько позже в тот же день, но еще до того как у меня появилась возможность поговорить с Чаком, чтобы понять его точку зрения, мне позвонила Урсула, начальник отдела кадров этой компании, и попросила меня зайти к ней в кабинет. Наш разговор с Урсулой оказался первым в череде практически не отличающихся один

от другого разговоров, которые впоследствии состоялись у меня в других компаниях. Чак пришел к Урсуле с жалобой, что в случае, если в команде будет практиковаться парное программирование, он окажется в проигрыше, что было бы явной несправедливостью по отношению к нему. Чак, один из лучших программистов компании и один из наиболее ревностных борцов за качество, объяснил Урсуле, что его ежегодные повышения зарплаты, как правило, оказывались выше среднего уровня по компании, поскольку он стабильно пишет самый качественный код в своей группе. Если в группе будет практиковаться парное программирование, сказал он, то руководитель Чака не сможет адекватно аттестовать его, поскольку будет непонятно, какой код написал Чак, а какой — его напарник. В результате, как утверждал Чак, он будет несправедливо обделен при повышении зарплаты. Этот довод убедил Урсулу и она сказала, что не разрешает мне практиковать в этой группе парное программирование, поскольку это может создать проблемы с оценкой индивидуального вклада каждого работника и привести к несправедливой аттестации работников.

Из-за подобных ситуаций и наличия в командах таких работников, как Чак, самыми трудными проблемами, с которыми вам придется столкнуться, наверняка окажутся проблемы, связанные с политикой отдела кадров. Сотрудники отдела кадров могут оказаться либо вашими помощниками, либо вашими противниками, которые лишь усугубляют указанные проблемы. В этом разделе мы рассмотрим потенциальные проблемы, связанные с политикой отдела кадров. Такими проблемами являются структуры подчиненности, периодические аттестации работников, решение вопросов об индивидуальном вкладе каждого из работников и определение путей совершения карьеры.

Структуры подчиненности

Не существует ни одной структуры подчиненности, которая может принести успех при использовании Scrum. Мне приходилось иметь дело с функциональными, проектно-ориентированными и матричными организациями, каждая из которых была успешной. Матричной организации придется столкнуться с большими проблемами, но это не должно оказаться неожиданностью для организации, которая выбирала именно эту структуру из-за других обеспечиваемых ею выгод. Хотя я и не выступаю решительно в пользу какого-то конкретного типа организационной структуры, могу сказать, что организация должна быть как можно более “плоской”, т.е. количество иерархических уровней в организации должно быть как можно меньшим. Чем больше иерархических уровней разделяет членов команды и высшее руководство компании, тем больше появляется возможностей для возникновения дисфункциональности.

Подчиненность Scrum-мастера

При обсуждении уровней управления зачастую возникает вопрос о том, могут ли члены команды находиться в подчинении у Scrum-мастера. Обычно считается, что это плохая идея. Однако в данном случае я возьму на себя смелость не согласиться с общепринятой точкой зрения. Иными словами, я не выступаю категорически против того, чтобы члены команды находились в подчинении у своего Scrum-мастера. Эта моя точка зрения может объясняться тем, что я сам многие годы выступал одновременно в роли и Scrum-мастера, и начальника в небольших организациях, которые не могли позволить себе роскошь разделить эти две роли. Более того, мне встречалось множество других людей, которые успешно совмещали в себе роли Scrum-мастера и начальника.

Обычное возражение сводится к тому, что член команды, который находится в подчинении у Scrum-мастера, не сможет свободно высказывать свою точку зрения в ходе ежедневных совещаний разработчиков. Разработчик, например, не отважится открыто заявить о возникшей у него проблеме, опасаясь, что эту проблему ему припомнят в ходе очередной аттестации. Разумеется, здесь есть определенный риск. Но этот риск может быть нивелирован назначением на роль Scrum-мастера человека, способного понять, как важно, когда разработчики открыто обсуждают возникшие у них проблемы. К тому же, когда в ролях Scrum-мастера и начальника выступает один и тот же человек, то для самой команды есть определенные преимущества. В частности, такому человеку подчас легче устраниТЬ некоторые препятствия, с которыми сталкивается команда.

Бывают ли Scrum-мастера, подчиняться которым я не пожелал бы членам ни одной команды? Разумеется. Вообще говоря, я предпочитаю, чтобы члены команды находились в подчинении у функциональных менеджеров, а не у своего Scrum-мастера. Однако среди проблем, связанных с использованием гибкой методологии разработки, подчинение членов команды своему Scrum-мастеру — не самая большая (если, конечно, Scrum-мастер в состоянии справиться с ролью начальника).

Подчиненность владельцу продукта

Учитывая мою готовность допустить подчинение членов команды своему Scrum-мастеру, вы, наверное, удивитесь, узнав о моем категорическом неприятии подчинения членов команды своему владельцу продукта. Все дело в том, что в самых благополучных командах существует естественная напряженность между владельцем продукта и командой. Одна из обязанностей владельца продукта заключается в том, чтобы заставлять команду реализовывать как можно быстрее как можно большее количество функциональных возможностей. Хорошая команда, как правило, ничего против этого не имеет. Более того, хорошим командам это даже нравится. Но иногда полезно осадить владельца продукта, если команда кажется, что его чрезмерные требования приведут лишь к ухудшению внутреннего качества продукта. Я полагаю, что, когда команда подчиняется своему владельцу продукта, естественная — и необходиМая! — напряженность между владельцем продукта и командой постепенно пропадает. Одно дело, когда члены команды иногда противятся желанию владельца продукта реализовывать как можно быстрее как можно больше функциональных возможностей, и совсем другое дело, когда им приходится делать то же самое, находясь *в подчинении* у своего владельца продукта.

В силу тех же причин нецелесообразно, чтобы владельцу продукта подчинялся Scrum-мастер. Вовсе не обязательно, чтобы Scrum-мастер и владелец продукта находились на одном иерархическом уровне в организационной схеме компании, но, участвуя в реализации одного и того же проекта, они должны относиться друг к другу как к равным партнерам.

Периодические аттестации работников

Многие призывают организации отказаться от системы проведения ежегодных аттестаций работников. Я обращался с этим призывом к отделам кадров разных организаций, однако нашел понимание лишь у очень маленьких организаций, начальники отделов кадров которых были слишком занятыми людьми, чтобы категорически настаивать на проведении ежегодных аттестаций. Поэтому вместо того чтобы просто советовать вам бороться против традиции, которую вам вряд ли по силам ликвидировать, предлагаю

рассмотреть влияние периодических аттестаций работников на ваши Scrum-команды, а также обсудить способы, с помощью которых можно минимизировать отрицательное влияние периодических аттестаций и использовать их положительное влияние.

Попытайтесь изъять из периодических аттестаций наиболее индивидуальные факторы. Не секрет, что поведение работников определяется в первую очередь тем, что больше всего ценится во время аттестаций. Я беру в руки старую аттестационную форму, которая предлагает мне оценивать работника по “степени, в которой данный работник справляется со своими заданиями в рамках установленного бюджета и календарного плана”. Как, по-вашему, будет вести себя работник с учетом этого требования, если в прошлый раз он был не лучшим образом аттестован в соответствии с этим критерием? Станет ли он уделять время своему коллеге, который обратится к нему за помощью? Наверняка нет. Индивидуальные факторы оценки обуславливают поведение, ориентированное исключительно на индивидуальный результат. Мы же хотим побуждать людей действовать так, чтобы это приносило максимальную пользу команде и продукту. В большинстве западных культур удаление из аттестации всех индивидуальных факторов успеха неминуемо столкнется с сопротивлением множества членов команды. В подобных ситуациях следует добиваться хотя бы примерного паритета (50/50) между индивидуальными и командными факторами успеха.

Включайте командные факторы. В большинстве систем аттестации предусмотрены пункты, в которых руководитель может отразить качество взаимодействия аттестуемого работника с другими членами команды. Правильно построенная система аттестации должна, не ограничиваясь наличием одного-двух таких пунктов, обеспечить сфокусированность на командной работе. Рассмотрим случай, когда работники оцениваются по тому, насколько эффективно каждый из них “справляется со своими заданиями в рамках установленного бюджета и календарного плана”. Начальное усовершенствование может заключаться в том, чтобы сформулировать этот пункт по-другому: “насколько эффективно работник помогает команде выполнить поставленные перед ней задачи в рамках установленного бюджета и календарного плана”. Но даже такая формулировка не является решением проблемы, поскольку слова “помогает команде выполнить” по-прежнему являются показателем индивидуального вклада. На самом же деле речь здесь должна идти о том, “насколько эффективно команда справляется со своими заданиями в рамках установленного бюджета и календарного плана”; при этом все члены команды должны получить по этому пункту одинаковые оценки.

ВОЗРАЖЕНИЕ

“Команда может функционировать, как единое целое, но в любом случае она состоит из отдельных работников. Когда мы просим одного из членов своей команды о помощи, почти всегда слышим примерно такой ответ: “Ничего не поделаешь, приятель, у меня полно своей работы”. В лучшем случае он действительно занимается своей работой, а в худшем — продолжает, как ни в чем не бывало, шарить в Интернете. Конечно, обычно находится кто-то, кто соглашается помочь, но по результатам ежегодной аттестации он получает такую же оценку, как и тот, кто никогда никому не помогает. Разве это не демотивирует остальных членов команды?”

Я согласен с тем, что это будет демотивировать остальных членов команды. Кто-то обязан поговорить с таким эгоистом и рассказать ему о разлагающем влиянии на членов

команды. В идеальном случае такое поведение должно быть подвергнуто публичному осуждению в ходе очередной ретроспектины спринта. Как альтернативный вариант такую воспитательную беседу с эгоистом может провести Scrum-мастер. Кроме того, поскольку какая-то часть ежегодной аттестации почти наверняка будет основываться на индивидуальных факторах достигнутого результата, такое поведение одного из членов команды может послужить прекрасным поводом для снижения его оценки.

Проводите аттестации гораздо чаще, чем один раз в году. Работники и их руководители должны встречаться как можно чаще для проведения неформальных обсуждений, касающихся достигнутых результатов, ожиданий и целей. Но если вы решили ограничиться лишь формальными аттестациями, проводите их гораздо чаще, чем один раз в году. Хотя это относится даже к организациям, не использующим гибкую методологию разработки, в случае Scrum это становится критически важным, поскольку выполнение Scrum-проектов продвигается гораздо быстрее и работникам приходится осваивать новые навыки и приемы работы, особенно в течение первых двух лет.

Предлагайте участвовать в обзорах как можно более широкому кругу людей. Когда вы приступаете к проведению очередной аттестации, маловероятно, что вы знаете все обо всех. Вот почему так важно получать информацию от других людей, и желательно, конечно, чтобы круг этих людей был как можно более широким. Функциональный менеджер должен получать соответствующую информацию от Scrum-мастера аттестуемого работника, от владельца продукта, других членов команды (по крайней мере от некоторых из них), от ряда сотрудников, работающих в соответствующей функциональной группе, а также от некоторых пользователей или заказчиков, с которыми приходилось иметь дело аттестуемому работнику. Я обычно выбираю нескольких таких потенциальных "информаторов" и рассылаю им по электронной почте письма с просьбой описать, что аттестуемый работник должен начать делать, прекратить делать или продолжить делать, чтобы улучшить свои результаты. Просматривая полученные ответы, я стараюсь обнаружить в них нечто общее и на основании этого общего сформулировать конструктивные предложения.

Информируйте отдел кадров и привлекайте его сотрудников к участию в аттестациях. Многие из обсуждавшихся нами изменений требуют участия или утверждения со стороны вашего отдела кадров. Кроме того, пытайтесь информировать его о том, какие изменения предполагается осуществить в организации разработки. Если вы проводите занятие по обучению Scrum, рассчитанное на половину рабочего дня, попросите кого-нибудь из отдела кадров присутствовать на этом занятии. Именно так поступила Габриэль Бенефилд (Gabrielle Benefield), директор по разработке с применением гибкой методологии компании Yahoo!. Представитель отдела кадров, присутствовавший на занятии по обучению Scrum, был настолько заинтригован услышанным, что отдел кадров Yahoo! начал использовать Scrum для управления собственным проектом обновления процесса ежегодной аттестации. Вот как описывает результаты Габриэль Бенефилд.

С помощью Scrum им удалось вовремя выполнить этот проект, который оказался весьма успешным. Им понравились ритм, в котором проводятся итерации, а также частое проведение совещаний, в ходе которых участники проекта знакомились с ходом его выполнения: дело в том, что проект выполняла рассредоточенная команда, которой приходилось параллельно заниматься другими делами, не связанными с этим проектом.

Увольнение членов команды

Когда во время конференции я увидел, что ко мне направляется Дерек, я обрадовался. Я познакомился с ним примерно год назад, когда проводил курс обучения в его компании. После этого я неоднократно бывал в этой компании и с удовольствием общался с Дереком. Правда, последние три месяца я не встречал его и сейчас был рад возможности вновь пообщаться с этим приятным собеседником. Когда мы поприветствовали друг друга и начали разговор, я заметил, что он чем-то озабочен. Дерек рассказал мне, что во время обзора спринта, состоявшегося на прошлой неделе, его команда предложила ему покинуть пост Scrum-мастера и вообще уйти из команды. Он был вынужден подчиниться этому требованию и сейчас пытается найти в своей компании какую-нибудь другую Scrum-команду, которая согласилась бы принять его в свой состав. Но потрясение, которое он испытал, когда его попросили уйти из его прежней команды, все еще давало знать о себе.

Несмотря на то что ситуации, похожие на возникшую у Дерека, встречаются достаточно редко, они не являются чем-то совсем уж неслыханным. Вопрос о том, есть ли у команды право исключить кого-либо из своего состава, обсуждался неоднократно. К увольнению кого-либо из членов команды, известному как “изгнание из стаи”, нельзя относиться слишком поверхностно и воспринимать это как рядовое событие. Прежде чем принимать столь ответственное решение, нужно попытаться уладить проблемы, которые заставили часть (или всех) членов команды решить, что для всех будет лучше, если кто-то из членов команды покинет ее.

Сама команда не должна иметь права увольнять из своих рядов кого бы то ни было. Вспомним, о чем говорилось в главе 12, “Управление самоорганизующимся коллективом”. В ней говорилось о том, что самоорганизация возникает не на пустом месте. Чтобы возникла самоорганизация, необходимы соответствующие предпосылки. Люди самоорганизуются в рамках, установленных соответствующей организацией. Мы определили это как модель CDE, которая гласит: чтобы возникла самоорганизация, необходимы контейнер, который ограничивает людей, участвующих в этой самоорганизации, определенные различия между ними и трансформирующие обмены. В главе 12, “Управление самоорганизующимся коллективом”, говорилось и о том, что лидеры в организации оказывают влияние на самоорганизующуюся команду, корректируя ее контейнеры, различия и обмены. Например, со временем в результате “притирки” команда может оказаться чересчур однородной. Дальновидный и проницательный владелец продукта, функциональный менеджер или даже Scrum-мастер может противодействовать этому процессу нивелирования различий, добавив двух новых членов команды, имеющих совершенно другой опыт работы, квалификацию, навыки, стиль принятия решений и т.п.

Но разве не может случиться так (а в приведенном нами примере именно так, скорее всего, и случится), что команда примет в штыки этих новых, не вписывающихся в нее членов, уволит их, а заодно и выразит свое недоверие лидеру, который преднамеренно включил этих чужаков в состав команды? Таким образом, непререкаемым авторитетом в отношении состава команды должен выступать кто-либо из высшего руководства организации. Этот лидер должен, конечно же, выслушать мнение членов команды, которым может показаться, что кто-то из членов команды мешает ей добиться более высокой производительности, и учесть это мнение при вынесении окончательного решения. Однако ни в коем случае нельзя допускать, чтобы вопрос увольнения был целиком и полностью отдан на усмотрение членов команды.

Пути карьеры

Одни сотрудники могут беспокоиться по поводу своего возможного исключения из команды, а другие — по поводу очередной ступеньки в их карьере. В большинстве организаций исторически сложилось так, что каждый сотрудник может отчетливо представить свою потенциальную служебную карьеру. Достигнув определенного уровня технической квалификации, вы становитесь лидером небольшой группы работников, имеющих примерно такую же квалификацию, как у вас. Затем вы становитесь менеджером, затем — старшим менеджером и т.д. На каждой очередной ступеньке этой служебной лестницы вы немножко теряете в своей технической квалификации, но зато подчиненных у вас становится все больше и больше. При этом количество подчиненных может напрямую зависеть от вашей важности в данной организации.

С уплощением организационной схемы, к которому приводят внедрение Scrum и ликвидация некоторых ролей или должностей, многие сотрудники перестают понимать, какими окажутся их новые карьерные пути. Они хотят знать, какие работы они будут выполнять на этом пути и как они сами (и все остальные) будут знать, что их работа стала более ценной для организации. После того как организация внедрит у себя Scrum, успех любого сотрудника этой организации уже не может измеряться количеством людей, находящихся у него подчинении. Теперь успех сотрудника может измеряться степенью ответственности, которая на него возлагается. Например, Scrum-мастер, являющийся в своем деле новичком, может быть назначен в небольшую и, возможно, достаточно опытную команду. Успешно справившись с проектом, который выполняла данная команда, этот Scrum-мастер может быть назначен в другую команду, не имеющую достаточного опыта работы со Scrum, однако выполняющую более важный проект. Это может продолжаться до тех пор, пока наш Scrum-мастер не начнет работать одновременно с несколькими командами, возглавит сообщество практиков для Scrum-мастеров и т.д.

Тот же путь свершения карьеры (успех в одном проекте приводит к увеличению степени ответственности при выполнении следующего проекта) относится ко всем ролям в Scrum-команде, в том числе к программистам, тестерам, дизайнерам и т.д. В самом начале своей карьеры программист может быть включен в состав команды лишь для написания кода и, возможно, выполнения еще каких-то малозначительных работ. Впоследствии этот программист может быть включен в состав какой-то другой команды для передачи своего опыта разработки веб-сайтов с высокой степенью доступности. В дальнейшем он может быть включен в состав какой-то команды, которой могут понадобиться его навыки, связанные с решением проблем и умением поддерживать хорошие отношения с остальными членами команды. Успех приводит к увеличению степени ответственности.

Подобный подход преобладает в SAS, частной компании, специализирующейся на разработке программного обеспечения (число сотрудников SAS превышает 4 тысячи). SAS неизменно оказывалась среди двадцати лучших компаний в рейтинге Best Companies to Work For (“Лучшие компании-работодатели”), ежегодно публикуемом журналом *Fortune*. В одной из статей, опубликованных в *Harvard Business Review*, описывается мотивационная культура в SAS.

Мотивационная культура SAS базируется на убежденности в том, что вдохновляющая умственная работа способствует повышению производительности и в конечном счете приводит к созданию более качественных продуктов. Компания не пытается "подкупать" работников опционами на акции, она никогда не предлагала им ничего подобного. В SAS самой подходящей благодарностью за хорошо выполненную работу является предоставление права выполнить еще более сложный и ответственный проект (Florida and Goodnight, 2005, 126).

ВОЗРАЖЕНИЕ

"Но если речь идет о самоорганизующихся командах, то как ответственность за решение проблем или проектирование системы с высокой степенью доступности может возлагаться на одного человека?"

Ответственность не возлагается на одного человека, ответственность несет вся команда. Но лидеры могут сообщить о своих повышенных ожиданиях в отношении одного из членов команды: "Мы хотим, чтобы вы работали в этой команде, потому что вы умеете ладить с людьми. Мы помним, как вы уладили конфликт между Франсуа и Джеймсом год назад. Подобные конфликты могут возникнуть и в этой команде". Нет никаких причин хранить в тайне ожидания лидера в отношении одного из членов команды или причины ввода в состав команды какого-то конкретного сотрудника. Конечно, если говорить о приведенном выше примере, то причину, заставившую лидера включить в состав команды сотрудника, умеющего ладить с людьми, разглашать вовсе не обязательно. Но когда кого-то из сотрудников включают в состав команды в связи с его высокой технической квалификацией, нет смысла держать это в тайне от остальных членов команды.

Команда состоит из людей, а между людьми всегда возникают проблемы

Поскольку разработка программного обеспечения является сферой деятельности, в которой человеческий фактор играет особенно важную роль, для этой сферы деятельности весьма характерны человеческие проблемы. Какие именно проблемы возникнут в той или иной команде, предугадать заранее невозможно. Проблемы, о которых говорилось выше, относятся к числу типичных. Другие человеческие проблемы, которые могут возникнуть в вашем случае, наверняка удастся решить, следя тем же принципам, которые лежат в основе предложенных в этом разделе мер по устранению препятствий на пути успешной реализации проекта.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Встретитесь с кем-либо из сотрудников вашего отдела кадров. Кратко объясните ему сущность методологии Scrum и причины, заставившие ваш отдел или команду пользоваться этой методологией. Поясните, какие конфликты с существующей кадровой политикой могут возникнуть в результате внедрения Scrum. Спросите у своего собеседника, может ли он предвидеть какие-то другие конфликты. Попросите его оказать вам помощь в решении этих проблем.

Группа технического обеспечения

Любая команда, которая пыталась применять Scrum в неподходящей рабочей обстановке, знает, как иногда трудно это делать. Идеальная рабочая обстановка может поддерживать членов команды в процессе освоения ими этой новой для них методологии, но, к сожалению, во многих случаях рабочая обстановка, в которой приходится действовать Scrum-команде, отнюдь не способствует ее производительной работе. Вообще говоря, физическая рабочая обстановка, в которой приходится действовать Scrum-команде, может оказывать столь значительное влияние на ее производительность, что не кажется таким уж преувеличением высказывание Джеральда Уайнберга (Gerald Weinberg), спросившего: “Кто является самой важной персоной процесса? Тот, кто расставляет мебель” (Dinwiddie 2007, 208).

Физическая среда, в которой приходится работать команде, может оказывать настолько глубокое влияние на успешность освоения командой гибкой методологии разработки, что во втором издании книги *Extreme Programming Explained* Кент Бек (Kent Beck) и Синтия Andres (Cynthia Andres) подняли “Информативное рабочее пространство” (“Informative Workspace”) до уровня основного условия (2004). Учитывая влияние, оказываемое физической средой, в которой приходится работать команде, на ее способность эффективно использовать гибкую методологию разработки, мы рассмотрим в этом разделе два аспекта этой среды: физическое пространство и обстановку в этом пространстве.

Физическое пространство

Традиционный “хай-тек”-офис с отсеками высотой около двух метров является очевидным препятствием для общения между членами команды. Самой распространенной его альтернативой у команд, которые сами участвовали в обустройстве своего рабочего пространства, является то, что на языке коммерческих дизайнеров интерьера принято называть “пещерами и залами” (“caves and commons”). Этот подход представляет собой сочетание небольших уютных мест уединения (“пещер”) и общих пространств (“залов”), т.е. пространств, где члены команды общаются между собой.

Типичный подход “пещер и залов”, применяющийся до внедрения Scrum, мог включать отсеки, выделяемые для каждого из членов команды, и некую центральную зону, содержащую, возможно, пару диванов, обычную школьную доску и книжный стеллаж. Идея заключалась в том, что члены команды могли собираться в этой центральной зоне (в общем пространстве) для непринужденного обсуждения тех или иных вопросов. Когда Scrum-командам предоставляют такой шанс, они используют идею “пещер и залов”, изменяя, однако, соотношение между пещерами и общим пространством в пользу общего пространства. Рабочее помещение типа “пещер и залов”, предназначенное специально для Scrum-команды, обычно предполагает полный отказ от использования индивидуальных отсеков, вместо которых создается большое общее рабочее пространство, соседствующее с парой небольших офисов (или комнат для проведения совещаний), в которых могут собираться те, кому это необходимо.

Вот что говорят о своем переходе к открытому рабочему пространству Scrum-команды компании 3M: “Мы пришли к выводу, что открытое рабочее пространство

является превосходным способом стимуляции импровизированного сотрудничества. Каждому сразу же видно, с кем из членов команды можно обсудить ту или иную проблему". Дух и энергия сотрудничества, присущие открытому рабочему пространству, как говорят они, придает команде огромный заряд энергии. Они заключают: "Обустройство помещения, способствующего сотрудничеству, позволило создать обстановку, исключительно благоприятную для работы Scrum-команды. Эта обстановка способствовала сплоченности команды и ее высокой сосредоточенности на выполняемой работе" (Moore et al., 2007, 176).

Еще одним преимуществом открытого рабочего пространства этого типа является легкость, с которой можно менять его планировку. Когда у членов команды возникают идеи по поводу дальнейшего совершенствования своей совместной работы, они нередко начинают экспериментировать с разными вариантами организации этого пространства. Кроме того, если время от времени меняется численный состав команды, очень важно, чтобы у команды была возможность быстро реконфигурировать открытое рабочее пространство, приспособливая его к своим изменяющимся потребностям.

ВОЗРАЖЕНИЕ

"Я не желаю работать в общем помещении, где полно людей. Там слишком шумно, а мне нужны тишина и покой, чтобы сосредоточиться".

При разработке программного обеспечения иногда действительно требуются абсолютная тишина и покой, чтобы сосредоточиться. При разработке Scrum-проекта чаще возникают ситуации, когда гораздо важнее предоставить людям возможность сотрудничать, обсуждать общие для них проблемы, делиться знаниями и добиваться взаимопонимания. Когда же кому-то действительно требуются тишина и покой, то к желанию работника скрыться на какое-то время в уединенной "пещере" команда должна относиться с пониманием. Возможен еще один вариант — беруши или музыкальный плеер с наушниками. Вообще говоря, этот вариант не должен приветствоваться, но в исключительных случаях, когда требуется абсолютная концентрация, он может оказаться вполне приемлемым.

К счастью, большинство людей полагает, что преимущества более частого общения со своими товарищами по команде перевешивают такие недостатки, как повышенный уровень общего шума в помещении. На это обстоятельство указывают, в частности, Сайед Райхан (Syed Rayhan) и Нимат Хаке (Nimat Haque).

Как ни странно, но концепция открытого пространства с самого начала понравилась команде. Нам не понадобилось долго убеждать команду в ее преимуществах. Правда, кое-кто беспокоился о том, сможет ли он сосредоточиться на своих задачах, если поблизости кто-то будет обсуждать совершенно иные проблемы или если коллеги будут отвлекать его вопросами или просьбами о помощи. Однако в конечном счете все пришли к выводу, что такое взаимодействие в действительности помогает им решать свои проблемы быстрее и позволяет учиться друг у друга. Теперь они пришли к выводу, что работа в индивидуальных отсеках контрпродуктивна и что они уже не согласились бы вернуться к старому способу работы (2008, 354).

Комната для планирования становится открытым рабочим пространством

До внедрения Scrum многие команды обожали проводить время в так называемой “комнате планирования боевых действий” (war room), роль которой исполнял обычный конференц-зал, который команде разрешалось занимать и использовать для всех своих совещаний. Наличие особой комнаты планирования для Scrum-команды становится менее актуальным, поскольку ее роль исполняет все открытое рабочее пространство, предоставленное команде. Ежедневные совещания разработчиков и другие совещания чаще всего проводятся в привычном для Scrum-команды открытом рабочем пространстве, а не в конференц-зале.

Одним из преимуществ традиционной комнаты планирования является то, что ее можно использовать как удобное место для проведения незапланированных совещаний. Четыре члена команды, которым внезапно пришло в голову провести расширенное обсуждение определенной проблемы, могли просто прийти в такую комнату, не обращая внимания на то, что это их совещание не предусмотрено общим календарным планом. Поскольку комната планирования принадлежала команде, последняя почти наверняка могла пользоваться ею всякий раз, когда в этом возникала необходимость. Scrum-командам также требуется место для проведения незапланированных совещаний. Хотя этим местом по-прежнему может быть небольшой конференц-зал, выделенный данной команде (или совместно используемый двумя-тремя Scrum-командами), для проведения незапланированных совещаний вполне может подойти небольшой стол, установленный посреди открытого рабочего пространства данной команды. Будет ли выбрано место проведения импровизированных совещаний “за дверью” или непосредственно в открытом рабочем пространстве Scrum-команды, в значительной мере зависит от того, предпочитает ли команда, чтобы все дискуссии проводились во всеуслышание (причем каждый из членов команды сам решает, принимать ли ему участие в дискуссии), или она предпочитает проводить длительные дискуссии “за дверью”, чтобы не создавать дополнительных помех тем, кто работает в открытом рабочем пространстве.

Если вы собираетесь заняться реконфигурированием рабочего пространства с целью создания крупной открытой площадки, на которой могла бы трудиться Scrum-команда, позаботьтесь о том, чтобы места в этом открытом рабочем пространстве хватило для всех, в том числе для Scrum-мастера и даже для владельца продукта. Нет ничего хуже, когда в открытом рабочем пространстве кому-то из членов команды не хватает места. Если, например, проектировщиков придется посадить отдельно от остальных членов команды, это наверняка вызовет у них чувство обиды. Еще хуже то, что участники проекта, которым придется работать отдельно от остальных членов команды, будут чувствовать себя чужаками в этом проекте, в то время как остальные члены команды, сидя в непосредственной близости друг от друга, будут ощущать себя единым коллективом.

Я говорю это вовсе не к тому, что все сто человек, участвующих в проекте и составляющих, возможно, дюжину команд, непременно нужно разместить в одном большом помещении. Когда речь идет о крупных проектах, чаще всего создают несколько открытых рабочих пространств, каждое из которых вмещает в себя примерно 20 человек. Таким открытым рабочим пространством могут совместно пользоваться три или четыре команды. При этом нужно расположить людей таким образом, чтобы

они сидели рядом со своими Scrum-командами, а не со своими функциональными командами. Нужно, например, избегать ситуаций, когда все программисты данной компании сидят в одном помещении, а все тестеры — в другом.

Как важно получать поддержку со стороны высшего руководства компании

Одна из основных обязанностей Scrum-мастера — устранять любые препятствия на пути повышения производительности Scrum-команды, а рабочее пространство, которое препятствует общению и командной работе, является несомненной помехой. Однако в деле совершенствования рабочего пространства Scrum-мастеру зачастую требуется помочь со стороны сообщества в поддержку перехода к Scrum или кого-либо из высшего руководства компании. Именно к такому выводу пришла преподаватель методологии Scrum Габриэль Бенефилд, когда возглавляла внедрение Scrum в компании Yahoo!.

При взаимодействии с группой технического обеспечения, которая, как правило, настаивает на неукоснительном соблюдении установленного ею процесса, предельно забюрократизирована и обладает значительными властными полномочиями, не обойтись без помощи со стороны кого-либо из высшего руководства компании. Вам все время отвечают “нет”; вам постоянно чинят какие-то препятствия и подозревают в желании нарушить те или иные важные правила. Некоторые из команд пытались действовать с упреждением: сами убирали лишнюю мебель, а оставшуюся переставляли так, как им было удобно. Разумеется, все это совершалось вопреки политике компании. Иногда это сходило команде с рук, иногда — нет. Именно в таких случаях требуется помочь со стороны кого-либо из высшего руководства компании, кто помог бы устраниТЬ эти препятствия. Членам команды бывает очень трудно сделать это самим (подчас это может обернуться для них увольнением). Когда вам отвечают “нет”, ваша задача состоит в том, чтобы понять истинную причину такого ответа. Иногда это может быть финансовая причина — в таких случаях нужно попытаться найти более дешевое решение проблемы или изыскать способ финансирования такого решения, которое кажется вам более приемлемым. Если речь идет о мерах пожарной безопасности, то изменить что-либо вам, скорее всего, не удастся. Если же речь идет о времени или каких-либо других ресурсах, попытайтесь разобраться, не можете ли вы сами решить эту проблему.

ВОЗРАЖЕНИЕ

“Я не желаю расставаться со своим отсеком, особенно сейчас, когда я наконец-то заслужил право работать в отсеке с окном”.

Типичной проблемой при переходе к Scrum является необходимость расстаться с определенными выгодами и привилегиями тем разработчикам, которые оказались в наибольшем выигрыше от разработки программного обеспечения традиционным методом. Например, те, кому в свое время были присвоены солидные должности и звания, становятся просто “членами команды”. Тем, у кого были персональные отсеки (а тем более кабинеты) с большими светлыми окнами, приходится переселяться в общее рабочее пространство. Во многих случаях солидные должности и звания, а также

более привлекательные отсеки становятся символами определенного статуса в организации. Неудивительно, что обладатели этих благ и привилегий противятся их отъему и переходу в статус “как у всех”.

В подобных случаях иногда помогает призыв “сделать это для общего блага”. Иногда разумнее согласиться, что в жизни действительно случаются те или иные несправедливости, но если переход к Scrum окажется успешным, то в выигрыше окажется организация в целом, а это создаст более благоприятные возможности для каждого из сотрудников (например, возможность участия в более сложных и интересных проектах).

Мебель

Некоторые команды проявляют большие творческие способности в отношении использования своей мебели, особенно если их творческие идеи подкрепляются соответствующим бюджетом. Типичным подходом в этом случае является сочетание передвижных рабочих столов с большим открытым рабочим пространством. Это позволяет командам формировать рабочие пространства с использованием таких вариантов расположения мебели, которые кажутся им наиболее подходящими. Члены одних команд предпочитают сидеть лицом друг к другу — при этом рабочие столы расставляются вдоль стен помещения (параллельно стенам). Членам других команд кажется не совсем удобным смотреть целый день в лицо друг другу, и в таком случае рабочие столы также расставляются вдоль стен помещения, но перпендикулярно стенам — так, что члены команды видят спины друг друга. Позволяя легко реконфигурировать рабочее пространство применительно к изменяющимся потребностям команд, передвижные рабочие столы олицетворяют в себе важное “послание” членам команды: команда сама может организовывать свое рабочее пространство так, как ей кажется удобным и полезным с точки зрения максимизации производительности труда.

Вероятно, даже более важным, чем сами по себе передвижные рабочие столы, являются форма и ширина рабочих поверхностей. Большинство хороших Scrum-команд со временем начинают практиковать парное программирование (или, в более общем случае, парную работу двух любых членов команды). Даже если они прибегают к такой парной работе лишь при выполнении самых критических задач, для такой работы требуется соответствующая рабочая поверхность. Небольшие по площади или фигурные рабочие поверхности затрудняют совместную работу двух человек за одним монитором. Эта проблема усугубляется, когда под столом помещаются ноги только одного человека.

Проблемы, порождаемые “мелочами”

Много внимания приходится уделять куда большим мелочам, чем рабочие столы. Типичным источником проблем являются телефоны. Переместить рабочий стол с одного места на другое, как правило, не составляет труда. Гораздо больше проблем возникает с переносом телефонных аппаратов. Некоторые компании пытаются использовать VoIP-телефоны, но, насколько мне известно, даже в таких компаниях, команды, как правило, жалуются на аналогичные проблемы.

Джон Корнелл (John Cornell), директор по использованию гибкой методологии разработки в компании Kofax, столкнулся с совершенно другой “телефонной” проблемой при организации открытого рабочего пространства.

Первоначальные планы по использованию открытого рабочего пространства предусматривали совместное использование телефонных аппаратов всеми членами команды (при использовании традиционного метода разработки программного обеспечения телефонные аппараты были установлены в каждом индивидуальном отсеке). Руководство компании не думало, что это станет серьезной проблемой в наши дни, когда почти у каждого есть мобильный телефон. К тому же подавляющее большинство технического персонала не пользуется телефоном для служебных целей. Однако у работников на сей счет было совершенно другое мнение: они рассматривали офисные телефонные аппараты как свой важный рабочий инструмент. Членам команды показалось, что действия руководства компании, направленные на сокращение количества телефонных аппаратов, ведут к снижению их производительности.

Я уверен, что дома у некоторых из этих разработчиков вообще нет стационарных телефонов и они пользуются исключительно мобильными телефонами. Но вместе с тем я подозреваю, что некоторые из членов команды, лишившись персональных телефонных аппаратов, чувствуют себя чем-то вроде граждан второго сорта. Несмотря на то что руководство компании, вознамерившееся оптимизировать использование телефонных аппаратов, вовсе не подразумевало этого, нетрудно понять, его действия вполне можно интерпретировать именно таким образом.

Как рассадить людей

Как именно будут рассажены люди в совместно используемом открытом рабочем пространстве, как правило, не так важно, как их распределение по индивидуальным отсекам. Когда люди не разделены перегородками, каждый может наслаждаться видом своих коллег. Частое применение парного программирования не позволяет людям сидеть целый день на одном и том же месте. А способность перемещаться из одной части открытого пространства в другую (даже при отсутствии передвижных рабочих столов) ослабляет ощущение постоянства.

Выступая в роли защитника команды, Scrum-мастер часто сидит ближе всех к главному входу на территорию, занимаемую командой. Наставник по гибкой методологии разработки Джордж Динвайдди (George Dinwiddie) вспоминает об одной команде, Scrum-мастер (и одновременно руководитель) которой выступал в роли сторожевого пса при своей команде. Один из разработчиков называл впечатление, производимое этим Scrum-мастером, “доберманским впечатлением”, поскольку этот руководитель “резко прерывал свою работу, чтобы остановить каждого, кто пытался войти в комнату, и выяснить, что нужно этому человеку” (2007, 208). Если руководитель мог предоставить необходимую информацию, он так и делал, и команда была таким образом защищена от нежелательного вмешательства в свою работу. Если же он не мог предоставить информации, а необходимость в ее получении была настоятельной, посетитель получал разрешение войти на территорию, занимаемую командой.

О том, что должно быть хорошо видно в вашем рабочем пространстве

Теперь, когда мы рассмотрели собственно рабочее пространство и мебель, которой должно быть обставлено правильно организованное рабочее Scrum-пространство,

рассмотрим, что должно быть видно каждому в рабочем пространстве, идеально приспособленном для гибкой методологии разработки.

- **Крупноформатные, хорошо просматривающиеся диаграммы.** Хорошая Scrum-команда обязательно заполняет свое рабочее пространство совокупностью крупноформатных, хорошо просматривающихся диаграмм. Одной из наиболее типичных диаграмм такого рода является график выполнения оставшихся работ, на котором отображается оставшийся объем работ по состоянию на каждый день текущего спринта. Диаграммы такого рода служат наглядным визуальным напоминанием о текущем состоянии выполняемого проекта. Информация, представленная на диаграммах такого рода, постоянно привлекает внимание членов команды, поэтому ее нужно подавать таким образом, чтобы продемонстрировать членам команды именно то, что является самым важным в текущем спринте. Рон Джейфрис (Ron Jeffries) предлагает использовать определенную совокупность диаграмм, в том числе диаграммы, на которых отображаются количество выполненных приемочных тестов заказчика, состояние тестов (закончился удачно/закончился неудачно) по дням, графики выполнения оставшихся работ для спринта и выпуска, количество новых историй, занесенных в журнал запросов на выполнение работ в каждом спринте, и т.п. (2004а).
- **Дополнительные устройства обратной связи.** Помимо крупноформатных, хорошо просматривающихся диаграмм, Scrum-команда обычно использует в своем рабочем пространстве дополнительные визуальные устройства обратной связи. Одним из типичных устройств такого рода является лавовая лампа¹, которая включается каждый раз, когда аварийно завершается автоматизированная сборка. Кроме того, я работал с командами, которые применяют проблесковые красные огни для индикации исключительных ситуаций, таких как возникновение проблем, связанных с производственным сервером. Светодиодные табло могут быть запрограммированы для отображения сообщений из Twitter. Немалой популярностью пользуются так называемые ambient orbs² и Nabaztag rabbits (“Кролики Nabaztag”), представляющие собой беспроводные программируемые устройства в форме игрушечных кроликов, которые по желанию команды можно настроить таким образом, чтобы они изменяли свой цвет, воспроизводили звуковые сообщения или даже шевелили ушами. Архитектор программного обеспечения Йоханнес Броулл (Johannes Brodwall) отдает предпочтение более простым решениям (что, кстати говоря, вполне отвечает духу гибкой методологии разработки) и рекомендует использовать устройства, подключаемые посредством USB (например, устройства компании Delcom), которыми он пользуется для мониторинга тестирования, контроля прохождения разных стадий процесса и слежения за производственными серверами. Подобные устройства оживляют рабочее пространство, ненавязчиво демонстрируя информацию, которая может оказаться полезной для команды.

¹ Лавовая лампа — декоративный светильник в виде прозрачной стеклянной емкости, как правило, цилиндрической, с прозрачным маслом и полупрозрачным парафином. Снизу расположена электрическая лампочка, нагревающая и подсвечивающая содержимое цилиндра; при этом происходит “лавообразное” перемещение парафина в масле. — Примеч. пер.

² “Обтекаемые сферы” — особые устройства для отображения информации. — Примеч. пер.

- Каждый из членов вашей команды.** В идеальном случае все члены команды должны постоянно находиться на виду друг у друга. В первую очередь, это относится к Scrum-мастеру, а в идеале — и к владельцу продукта. Разумеется, я понимаю, что у владельца продукта могут быть обязанности не только по отношению к данной команде разработчиков, но и по отношению к другим группам. Это означает, что он не сможет постоянно находиться в рабочем пространстве данной команды. Но если речь идет об идеальном случае, то владелец продукта также должен постоянно находиться в поле зрения всех членов команды.
- Журнал запросов на выполнение работ данного спринта.** Один из наилучших способов гарантировать, что все запланированные для данного спринта работы будут своевременно выполнены, — это позаботиться о том, чтобы журнал запросов на выполнение работ постоянно находился на виду у всех членов команды. Этого можно добиться, разместив его на стене в рабочем пространстве данной команды (лучше всего — в форме доски с заданиями, т.е. диаграммы выполнения задач спринта). Диаграмма выполнения задач спринта обычно состоит из строк и столбцов, причем каждая строка содержит конкретную пользовательскую историю и одну индексную карточку или стикер для каждой задачи, связанной с данной историей. Соответствующий пример приведен на рис. 20.1. Карточки задач организованы по столбцам, включая как минимум столбцы “Предстоит сделать”, “Выполняется” и “Выполнено”³. Диаграммы выполнения задач спринта позволяют членам команды буквально с первого взгляда понять ход выполнения работ, а также быстро оценить оставшийся объем работ.

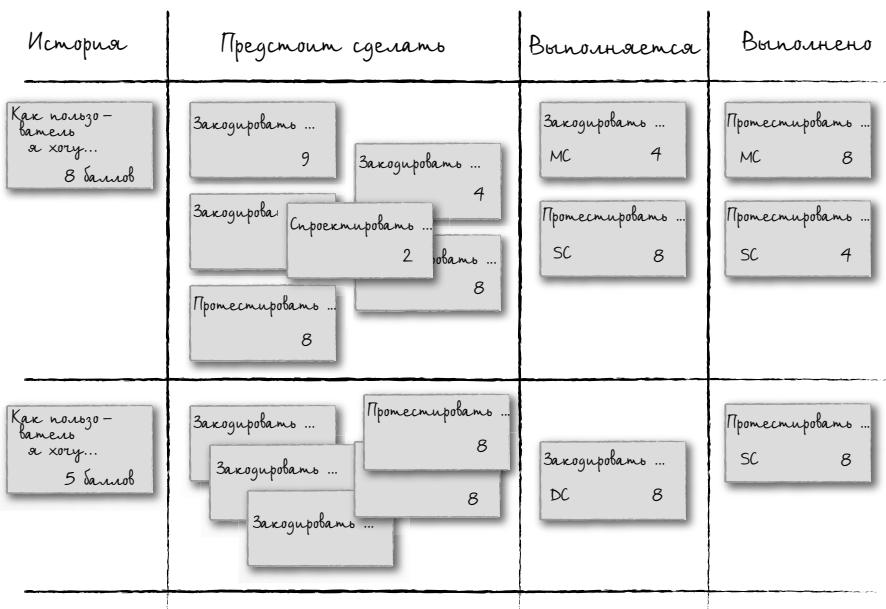


Рис. 20.1. Диаграмма выполнения задач спринта делает журнал запросов на выполнение работ данного спринта видимым каждому из членов команды

³ Фотографии разных диаграмм выполнения задач спринта представлены на сайте <http://www.succeedingwithagile.com>.

- **Журнал запросов на выполнение работ.** Одна из проблем, возникающих в связи с выполнением бесконечной последовательности спринтов, заключается в том, что каждый может почувствовать себя изолированным или отчужденным от того целого, что представляет собой запланированная к реализации совокупность новых функциональных возможностей. Хорошим способом снизить отрицательное влияние этой проблемы является выstellung журнала запросов на выполнение работ на всеобщее обозрение. Для этого можно воспользоваться таким простым решением, как помещение на стол, установленный посередине рабочего пространства команды, коробки из-под обуви, наполненной индексными карточками с пользовательскими историями. Еще лучше — прикрепить индексные карточки с пользовательскими историями, которые предстоит реализовать команде, на стене, где они будут видны каждому. Это даст возможность членам команды видеть, каким образом пользовательские истории, над которыми они работают в текущем спринте, связаны с другими пользовательскими историями, над которыми им предстоит вскоре работать.
- **По меньшей мере одна крупноформатная школьная доска.** Каждой команде нужна хотя бы одна крупноформатная школьная доска. Само по себе размещение такой доски в общем рабочем пространстве команды стимулирует спонтанные совещания. Инициатором проведения такого спонтанного совещания может выступить кто-то из разработчиков, решивший воспользоваться доской для обдумывания той или иной проблемы; другие могут заметить это и предложить свою помощь.
- **Необходимо предусмотреть спокойное и тихое место, где может уединиться тот, кто испытывает в этом потребность.** Как бы ни было важно открытое общение членов команды, иногда человеку нужно на какое-то время уединиться в спокойном и тихом месте. Это бывает необходимо хотя бы для того, чтобы поговорить по телефону о своих личных делах, которые касаются только его одного. Иногда это необходимо для обдумывания какой-то особенно сложной проблемы, когда нежелательны любые отвлекающие факторы.
- **Еда и питье.** Всегда неплохо, когда у вас под рукой есть что-нибудь пожевать и выпить. Вовсе необязательно, чтобы это были какие-то изысканные блюда. Необязательно даже, чтобы еду и питье закупала организация. Многие из команд, с которыми мне приходилось работать, покупали для себя небольшой холодильник, где хранился запас еды и питья, купленных членами команды в складчину. Другие команды покупают кофеварку. В третьих практикуется поочередная покупка членами команды бутербродов и сухих завтраков — как полезных для здоровья, так и не очень.
- **Окна.** Во многих организациях окна оказываются дефицитным ресурсом, которым наделяются лишь лучшие сотрудники. Открытое рабочее пространство удобно, в частности, тем, что окна в таком помещении принадлежат не кому-то конкретно, а всем членам команды. Даже если из окна открывается не самый привлекательный вид (например, на стоянку для автомобилей) и даже если ваш рабочий стол отделен от окна еще несколькими рабочими столами, то вы можете утешить себя хотя бы тем, что естественный свет, льющийся из окна, все же лучше искусственного света от ламп накаливания или ламп так называемого “дневного света”.

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Составьте приоритизированный список вопросов, касающихся физического рабочего пространства команды, которые могут влиять на ее производительность. Договоритесь о получении помощи со стороны сообщества в поддержку перехода к Scrum. Выясните, не согласится ли кто-либо из данного сообщества оказать вам помощь при проведении переговоров с группой технического обеспечения.
- ❑ Если вы можете что-то сделать для улучшения рабочего пространства своей команды без предварительного получения разрешения, сделайте это. Ни в коем случае не загромождайте проходы, устроенные на случай пожара, и не нарушайте никаких других правил, но помните, что подход “сначала сделай, а затем спроси” зачастую позволяет все же произвести кое-какие улучшения.

Отдел управления проектами

Наличие отдела управления проектами (Project Management Office — PMO), одной из целей которого является оказание помощи во внедрении Scrum, может оказаться великим благом для команды. Сотрудники PMO зачастую рассматривают себя как активных помощников в деле внедрения тех или иных методов, поэтому PMO может оказать существенную помощь в деле внедрения гибкой методологии разработки в масштабе всей организации. Однако, если участие PMO не будет организовано должным образом, оно может привести к возникновению дополнительных проблем, поскольку PMO будет пытаться защитить уже используемый процесс, вместо того чтобы способствовать его совершенствованию.

Одной из причин, по которым первой естественной реакцией большинства сотрудников PMO является сопротивление переходу к Scrum, является боязнь перемен (как в личном, так и в профессиональном плане), которые влечет за собой такой переход. Использование Scrum ведет к распределению традиционных обязанностей по управлению проектами между Scrum-мастером, владельцем продукта и командой, оставляя руководителей проекта в недоумении относительно своей роли в этой новой ситуации. Отсутствие упоминаний о PMO в большей части литературы по Scrum и гибкой методологии разработки явно не способствует росту оптимизма у сотрудников PMO.

См. также Роль руководителя проекта обсуждалась в главе 8, “Изменившиеся роли”.

В данном разделе я постараюсь развеять эти страхи, рассмотрев работу, выполняемую PMO в организациях, которые успешно внедрили у себя Scrum. Мы рассмотрим здесь три аспекта деятельности PMO и вклада этого отдела в дело внедрения гибкой методологии разработки: люди, проекты и процесс.

Люди

Хотя речь идет об отделе управления *проектами*, PMO оказывает огромное влияние на людей, участвующих во внедрении Scrum. Гибкий PMO должен выполнять следующие функции.

- **Разрабатывать программы обучения.** Во внедрении Scrum есть много нового и неизвестного для многих членов команды. РМО может оказать огромную помощь в разработке программы обучения, поиске сторонних преподавателей, которые будут воплощать в жизнь эту программу обучения, или в реализации этой программы собственными силами.
- **Оказывать наставнические услуги.** Помимо обучения, огромную пользу может принести индивидуальное и групповое (в составе небольших групп) наставничество. В ходе программы подготовки преподаватель говорит, например: “Вот как проводится совещание, посвященное планированию спринта” и, возможно, проводит практическое занятие, на котором моделируется совещание, посвященное планированию спринта. При использовании наставничества кто-либо, обладающий большим опытом применения гибкой методологии разработки, помогает команде провести реальное совещание, посвященное планированию спринта (или какое-то другое мероприятие, предусмотренное гибкой методологией разработки). Поначалу сами сотрудники РМО могут не располагать достаточной квалификацией в области гибкой методологии разработки, но они должны позаботиться о ее приобретении с помощью сторонних наставников, а затем сами оказывать наставнические услуги членам команды.
- **Подбирать и обучать наставников.** Успешная Scrum-инициатива со временем породит большую потребность в наставнических услугах, чем будет в состоянии обеспечить РМО, опираясь лишь на собственные силы. РМО должен заниматься подбором и обучением наставников путем наблюдения за командами, которым помогает, а затем предоставляя обучение и помочь, необходимые для того, чтобы отобранные таким образом сотрудники стали квалифицированными наставниками. За этими наставниками обычно закрепляются их текущие должности, но при этом на них возлагаются дополнительные обязанности (например, каждую неделю уделять по пять часов на оказание помощи какой-то определенной команде).

См. также Наставники, подобные описанным здесь, могут играть важную роль в деле внедрения Scrum в масштабе всей организации. Один из способов использования наставников описан в главе 3, “Модели внедрения Scrum”.

- **Изживание сложившихся стереотипов и привычек.** Когда организация приступает к внедрению Scrum, сотрудники РМО отслеживают команды, которые скатываются к старым привычкам, мешающим успешно осваивать гибкую методологию разработки. Впоследствии сотрудники РМО могут время от времени напоминать командам, что Scrum предполагает непрерывное совершенствование, и помогать им предотвратить появление самодовольства и самоуспокоенности.

Проекты

Несмотря на то что в результате трансформации традиционного РМО в гибкий РМО некоторые проектно-ориентированные обязанности исчезают, все же отдельные обязанности остаются актуальными, включая перечисленные ниже.

- **Помогать в составлении отчетности.** В большинстве организаций, достаточно крупных для того, чтобы они могли позволить себе содержание собственного РМО, как правило, предусматривается что-то наподобие еженедельных отчетов или встреч с главами соответствующих отделов, посвященных обсуждению текущего состояния каждого проекта. Если речь идет о встрече, то в ней должны участвовать соответствующие участники проекта, такие как владелец продукта или Scrum-мастер. Но если речь идет о еженедельном стандартизированном отчете о текущем состоянии проекта, то РМО может оказывать помощь в подготовке такого отчета.
- **Удовлетворять потребности, касающиеся соблюдения тех или иных стандартов.** Многие проекты должны соответствовать определенным стандартам (ISO 9001, Сарбейниса–Оксли (Sarbanes–Oxley) и т.п.) или внутрикорпоративным правилам (например, правилам обеспечения безопасности данных). Гибкий РМО может оказывать помощь командам, информируя их о необходимости соблюдения тех или иных стандартов или внутрикорпоративных правил, давая им соответствующие рекомендации и играя роль информационного центра, консультирующего команды по вопросам соблюдения стандартов и другим подобным вопросам.

См. также Вопросы соблюдения стандартов обсуждались в главе 19, “Сосуществование с другими подходами”.

- **Управлять темпом поступления новых проектов.** Одним из самых важных видов помощи, которую может оказывать гибкий РМО, является управление темпом поступления новых проектов в проектную организацию. Как указывалось в главе 10, “Структура команды”, важно ограничить объем работы реальной производительностью команды. В противном случае будет накапливаться незавершенная работа и связанные с этим проблемы. Когда завершается выполнение очередного проекта, можно приступать к новому проекту примерно такого же объема. Гибкий РМО может играть роль “швейцара” и помогать организации сопротивляться желанию браться за выполнение слишком большого числа проектов.

Процесс

Выступая в роли “хранителей процесса”, сотрудники РМО тесно сотрудничают со Scrum-мастерами своей организации, заботясь о том, чтобы использование методологии Scrum было максимально эффективным. Ниже перечислены виды деятельности, связанные с поддержанием процесса в надлежащем состоянии.

- **Предоставлять инструментарий и поддерживать его в надлежащем состоянии.** Вообще говоря, решения, касающиеся использования того или иного инструмента, нужно по мере возможности оставлять на усмотрение каждой отдельной команды. Если же это невозможно, то решение о выборе какого-то одного инструмента для всех проектов, обусловленное наличием у этого инструмента достаточного числа преимуществ, может принять одно из сообществ практиков. В качестве крайней меры решения, касающиеся использования того или иного

инструмента, могут иногда приниматься РМО, хотя это должно быть скорее исключением, чем правилом. Но гибкий РМО может оказывать командам помощь, приобретая соответствующие инструменты и выполняя любое необходимое конфигурирование или настройку.

- **Оказывать помощь в деле определения и сбора показателей.** Как и до перехода к использованию гибкой методологии разработки, РМО может определять и собирать количественные показатели, связанные с выполняемыми проектами. Scrum-команды даже еще более осмотрительны в отношении метрических программ, чем традиционные команды, поэтому именно в данной области РМО должен действовать с особой осторожностью. Гибкий РМО должен, в частности, заниматься сбором информации о том, насколько успешно действуют команды с точки зрения обеспечения ценности для пользователей и заказчиков.

См. также В главе 21, “Как далеко вы продвинулись в деле внедрения Scrum”, обсуждается совокупность показателей и подходов, позволяющих оценивать прогресс команды в освоении ею гибкой методологии разработки.

- **Сокращать непроизводительные затраты времени и прочих ресурсов.** РМО должен активно помогать команде в деле устранения из ее процесса всех действий и артефактов, не способствующих достижению ею конечного результата. Гибкий РМО должен избегать узаконивания документов, собраний, утверждений, одобрений и тому подобного, если только они не вызваны абсолютной необходимости. РМО должен также помогать командам выявлять в их действиях то, что препятствует повышению ценности создаваемых ими продуктов.
- **Оказывать помощь в создании и поддержке деятельности сообществ практиков.** Одной из самых важных задач гибкого РМО является оказание помощи в создании сообществ практиков и последующая поддержка их деятельности. Сообщества практиков не только способствуют внедрению Scrum в масштабе всей организации, но и помогают распространять ценные идеи от одной команды к другой.
- **Обеспечивать надлежащую степень единобразия среди команд.** У большинства команд (в особенности у Scrum-команд) “шерсть становится дыбом” от одной лишь мысли о принудительно насижданном единобразии. Наилучший тип единобразия среди команд — являющийся следствием согласия всех этих команд на использование какого-то определенного подхода. Гибкий РМО упрощает процесс достижения единобразия, способствуя быстрому распространению ценных идей от одной команды к другой. Большую пользу в этом отношении могут принести сообщества практиков и наставники, работающие одновременно с несколькими командами.
- **Координировать деятельность команд.** Поскольку сотрудники РМО работают с людьми из многих команд, важной их функцией является координация работы отдельных команд. Кто-либо из сотрудников РМО может первым заметить, что какие-то две команды начинают работать вразнобой или дублировать работу друг друга. В случае возникновения подобных ситуаций сотрудники РМО могут поставить команды в известность об этой проблеме.

- **Моделировать использование Scrum.** Активное участие большинства гибких РМО во внедрении Scrum достаточно быстро приводит их к пониманию полезности Scrum как универсального концептуального каркаса управления проектами. Когда к сотрудникам РМО приходит такое понимание, часто они решают сами использовать Scrum для управления деятельностью РМО. Они планируют ежемесячные спринты, проводят ежедневные совещания разработчиков и тому подобное, т.е. занимаются тем же, что и любая другая Scrum-команда.
- **Работать с другими группами.** РМО может оказывать неоценимую помощь командам в работе с другими группами, в частности с отделом кадров и группой технического обеспечения, как уже описывалось в этой главе.

Переименование РМО

Многие РМО считают целесообразным сменить свое название, чтобы лучше соответствовать своей новой роли в организации. Пока еще не выработалось какое-то стандартное название для группы, которая в прежние времена называлась РМО, но чаще всего мне встречались такие названия.

- Центр совершенствования применения Scrum
- Центр распространения знаний о Scrum
- Scrum-офис
- Поддержка разработки

В главе 2, “ADAPТация к Scrum”, я предостерегал против присвоения тех или иных названий процессу внедрения Scrum. Многие люди с большим подозрением относятся к сменам названий и к звонким названиям как таковым. Такая подозрительность может коснуться и РМО, если сменится лишь вывеска этого отдела, а его суть останется прежней. Поэтому, как бы ни назывался РМО (Центр совершенствования применения Scrum, Центр распространения знаний о Scrum или еще как-нибудь), чтобы способствовать успешному внедрению Scrum, РМО, который в прежние времена поддерживал процесс последовательной разработки в организации, придется изменить не только свое название, но и свою деятельность. Но, как уже указывалось в этом разделе, гибкий РМО может принести организации немалую пользу.

Подведем итоги

До поры до времени вы можете игнорировать влияние Scrum на отдел кадров, группу технического обеспечения и отдел управления проектами — и работать вполне успешно. Однако рано или поздно вам придется наладить сотрудничество с этими группами для обеспечения успешного долговременного перехода к использованию Scrum. Было бы нереалистичным полагать, что в организации можно внедрить тот или иной процесс, базирующийся на главенствующем положении “людей и взаимодействий, а не процесса и инструментов”, не привлекая к решению этой задачи отдел кадров. В типичной организации отдел кадров может оказывать на то, как люди воспринимают и выполняют свою работу, даже большее влияние, чем функциональные менеджеры этих людей.

Аналогично физическое окружение, в котором мы выполняем свою работу, также оказывает на нас непосредственное влияние. Представьте себе, например,

одновременное заявление компании о том, что “люди — самый ценный наш актив”, и запрет развешивать диаграммы на стенах. Понятно, какой вывод должны сделать в этом случае работники компании: “Самым ценным активом компании являются стены”.

PMO зачастую имеет колоссальное политическое влияние и огромный опыт в управлении проектами. Привлекая сотрудников этого отдела к внедрению Scrum, вы не только избавляесь от потенциального источника сопротивления, но и используете опыт, которым они располагают. Сотрудники PMO становятся поборниками самоорганизации, постоянного совершенствования, обладания командой правом собственности на создаваемый ею продукт, интенсивного общения, экспериментирования, сотрудничества и других не менее полезных вещей.

Нетрудно рассматривать отдел кадров, группу технического обеспечения и отдел управления проектами как досадные помехи, с которыми приходится бороться. Однако гораздо продуктивнее рассматривать каждую из этих групп как союзника в деле внедрения Scrum. Несмотря на то что отношения противостояния и соперничества могут приносить нужный результат в течение какого-то времени, для долговременного успеха требуется поддержка со стороны всей организации. Путь к освоению гибкой методологии разработки может быть долгим, и преодолевать его желательно вместе с друзьями, а не с врагами.

Дополнительная литература

Cockburn, Alistair. 2006. *Agile software development: The cooperative game*. 2nd ed. Addison-Wesley Professional.

Алистер Кокбурн, автор этой книги, получившей престижную награду *Jolt Award*, рассматривает широкий спектр тем, но самой ценной, на мой взгляд, является глава 3, “Команды, деятельность которых основана на общении и сотрудничестве”. В этой главе содержится исключительно ценная информация о влиянии физического окружения на команду, выполняющую проект. Текст этой главы из первого издания представлен по адресу <http://www.informit.com/articles/article.aspx?p=24486>. Разумеется, будет лучше, если вы прочтете всю книгу, а не одну эту главу.

Jeffries, Ron. 2004. Big visible charts. *XP*, October 20. <http://www.xprogramming.com/xpmag/BigVisibleCharts.htm>.

Превосходное описание некоторых крупноформатных, хорошо заметных диаграмм, которые обязательно должны присутствовать в рабочем пространстве команды, практикующей гибкую методологию разработки.

Nickols, Fred. 1997. Don't redesign your company's performance appraisal system, scrap it! *Corporate University Review*, May-June.

Здесь опубликовано немало замечательных материалов о проблемах, порождаемых периодическими аттестациями работников. Данная статья может служить хорошей отправной точкой для изучения этих проблем главным образом из-за ее краткости и убедительности приведенных в ней аргументов.

Seffernick, Thomas R. 2007. Enabling agile in a large organization: Our journey down the yellow brick road. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 200–206. IEEE Computer Society.

Сефферник описывает успешный переход к гибкому РМО в компании KeyCorp, крупном финансовом учреждении, в проектной организации которого трудится 1500 человек. Здесь, в частности, описано, как в ходе преобразования старого РМО в гибкий РМО удалось значительно сократить его состав (высвободившихся при этом сотрудников включили в команды-разработчики) и как гибкий РМО был переименован в Центр поддержки разработки программного обеспечения.

Tengshe, Ash, and Scott Noble. 2007. Establishing the agile PMO: Managing variability across projects and portfolios. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 188–193. IEEE Computer Society.

Тенгши и Нобль сформировали в компании Capital One Auto Finance гибкий отдел управления проектами. В этой статье описано, как происходило формирование этого отдела, и приводятся ценные рекомендации по преобразованию традиционного РМО в гибкий РМО.

ЧАСТЬ V

Что дальше

Когда позади у вас осталось 95% пути, можете считать, что половину пути к своей цели вы уже преодолели.

— Японская пословица

Глава 21

Как далеко вы продвинулись в деле внедрения Scrum

После того как вы приступите к внедрению Scrum, пройдет не так уж много времени, и у кого-нибудь наверняка возникнет вопрос “И каковы же наши успехи?” Этот вопрос не из разряда тех, на которые можно отдохнуться ответом типа “У нас все в порядке”. Точно так же (на ваше счастье!) вы не можете ответить предельно точно и лаконично: “Мы находимся сейчас на третьем уровне Scrum”. Внедрение Scrum — процесс сложный, и ответить на вопрос о степени овладения этой методологией также достаточно сложно. К счастью, многие компании, уже достаточно давно внедрившие у себя Scrum, экспериментировали с разными способами оценки прогресса, достигнутого в освоении Scrum, и выработали ряд эффективных подходов. Эти подходы задокументированы и могут с успехом использоваться другими компаниями.

В последующих разделах мы рассмотрим способы измерения прогресса, достигнутого командой в процессе освоения Scrum. Начнем с трех универсальных оценок освоения гибкой методологии разработки, которые используются многими компаниями. Затем мы рассмотрим возможности адаптации одного из этих способов оценки к особенностям вашей организации. Наконец, мы покажем, почему внедрение Scrum так важно рассматривать со сбалансированной точки зрения, и продемонстрируем оценочную таблицу, в которой реализована такая сбалансированная точка зрения.

Цель измерения

Прежде чем погрузиться в тему “что измерять”, рассмотрим для начала, почему мы пытаемся что-либо измерить. Если спросить у большинства людей, в чем цель измерения, они наверняка ответят вам, что цель измерения — определить объем, вес, длину и другие подобные характеристики объекта измерения. Это чрезмерно амбициозное определение *измерения*. Подлинная цель измерения заключается в том, чтобы снизить степень неопределенности. Чтобы то или иное измерение помогло снизить

степень неопределенности, оно вовсе необязательно должно быть точным. Рассмотрим в качестве примера суп, который я сегодня съел на обед. Я был в неизвестном мне ресторане, и томатный суп показался мне привлекательным на вид. Чтобы выбрать подходящую порцию, я спросил у официантки, чем отличается чашка супа от тарелки супа. Вместо того чтобы ответить, что разница между ними составляет три унции, официантка показала мне примерные объемы чашки супа и тарелки супа на пальцах. Это объяснение настолько снизило степень моей неопределенности, что я заказал тарелку супа.

Это очень важный момент, поскольку дискуссии о показателях, позволяющих оценивать объем работы, выполненной разработчиками, зачастую заходят в тупик именно из-за стремления участников дискуссии предложить показатель, обеспечивающий максимальную точность измерения. Между тем мы вовсе не нуждаемся в идеально точных измерениях. Мы нуждаемся в измерениях, которые помогали бы нам отвечать на вопросы. Самыми типичными вопросами, касающимися успешности внедрения Scrum, являются подобные перечисленным ниже.

- Оправданны ли наши инвестиции во внедрение Scrum?
- На улучшении чего нам нужно сосредоточиться в ближайшее время?
- Следует ли нам продолжать использовать Scrum?
- Лучше ли нам удаётся разрабатывать программное обеспечение, чем год назад?
- Повысилось ли качество наших продуктов в результате внедрения Scrum?
- Меньше ли в наших продуктах стало дефектов, чем прежде?
- Работаем ли мы быстрее, чем прежде?

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Поговорите с командами, которые уже приступили к освоению Scrum, и выясните, какими показателями они пользуются. Спросите у них также, какими показателями они хотели бы пользоваться.
- Прежде чем инициировать собственную программу показателей, составьте перечень вопросов, на которые вы пытаетесь ответить.

Универсальные оценки освоения гибкой методологии разработки

На многие из указанных вопросов можно ответить непосредственно. Например, чтобы определить, содержится ли в ваших нынешних продуктах меньше дефектов, чем до внедрения Scrum, сравните количество дефектов, о которых сообщил заказчик в течение первых 90 дней после выпуска нового продукта, с соответствующими данными, которые были собраны по прошлым проектам, выполнявшимся еще до внедрения Scrum. Можете даже нормализовать соответствующие данные в пересчете на количество строк кода, количество человеко-месяцев, затраченных на выполнение проекта, или на количество пользователей. Однако иногда нас интересует более глобальный вопрос: “В какой степени мы освоили гибкую методологию разработки?”

См. также Ниже в этой главе, в разделе “Сбалансированная система показателей Scrum-команд”, мы рассмотрим некоторые примеры измерения таких показателей, как влияние Scrum на качество продуктов.

Я понимаю аргументы тех, кто говорит, что не нужно задумываться, в какой степени мы освоили гибкую методологию разработки; нас должно интересовать лишь, выпускаем ли мы более качественные продукты, делаем ли мы это быстрее и с меньшими издержками. Поэтому, если нам и нужно что-то измерять, так это то, насколько успешно организация разработчиков достигает указанных целей. На определенном уровне я предпочел бы сказать так: для того чтобы понять, работает ли организация разработчиков в этом году лучше, чем в предыдущем, нужно посмотреть, принесли ли этой организации больший доход продукты, разработанные ею в этом году. Однако у такого подхода есть много проблем. Например, может наблюдаться достаточно большое расхождение во времени между периодами, когда организация совершенствует свою работу, и когда в результате этого совершенствования возникает рост доходов. Кроме того, внешние факторы, такие как рецессия или изменение спроса на продукцию соответствующей компании, могут перевесить результат улучшений, достигнутых разработчиками. Кроме того, изменение системы оплаты труда маркетологов соответствующей организации может заставить их переключить свое внимание на другие продукты.

Очевидно, что анализ такого показателя, как доход, полученный организацией от продажи конкретного продукта, в определенной степени дает информацию о процессе разработки продукта в целом, но в конечном счете он не позволяет нам ответить на вопрос о том, насколько эффективно действовала команда разработчиков программного обеспечения. Чтобы ответить на этот вопрос, мы можем воспользоваться замещающими (прокси) показателями, которые обычно являются ведущими индикаторами, заменяющими другие показатели, определение которых обходится нам слишком дорого, которые охватывают слишком много факторов или определяются слишком поздно, чтобы представлять собой хоть какую-то ценность. Степень освоения командой или организацией гибкой методологии разработки является весьма полезным прокси-показателем.

Чтобы понять, почему это так, допустим, что мы рассматривали деятельность вашей команды разработчиков в прошлом году и рассматриваем ее деятельность сейчас. Воспользовавшись некоторой формой оценки, мы пришли к выводу, что эта команда добилась лучших результатов с точки зрения работы спринтами. Возможно, она научилась лучше планировать объем работы, которую она в состоянии выполнить в течение одного спринта. Возможно, члены команды научились работать в более тесном контакте друг с другом в течение спринта. Возможно, команда настолько хорошо освоила инкрементный метод работы, что способна завершать *каждый* спринт созданием продукта, потенциально готового к использованию. Как бы то ни было, ваша команда гораздо эффективнее работает спринтами, чем год назад. Означает ли это, что ваша команда создает более качественные продукты, делая это быстрее и дешевле, чем в прошлом году? Нет! Но это говорит о том, что так *может* быть. Измеряя степень освоения командой гибкой методологии разработки, мы пытаемся понять, улучшает ли команда свою работу таким образом, который, по нашему мнению, должен

привести к действительно желанному для нас улучшению — к улучшению экономических результатов. Но при этом мы используем прокси-показатель, “степень освоения командой гибкой методологии разработки”, поскольку его можно измерить задолго до появления экономических результатов и поскольку он позволяет сосредоточиться лишь на одном факторе экономического успеха — на команде разработчиков.

Рассмотрим три универсальных подхода к измерению степени освоения командой гибкой методологии разработки.

Опрос на соответствие уровню “шодан”

Одним из первых подходов к оценке степени освоения командой гибкой методологии разработки (который продолжает использоваться и по сей день) является опрос на соответствие уровню “шодан”¹ (Shodan Adherence Survey), предложенный Биллом Кребсом (Bill Krebs) (Williams, Layman, and Krebs, 2004). Подход, предложенный Кребсом, представляет собой самоадминистрируемый опрос, включающий в себя 15 вопросов, которые охватывают весь спектр методов экстремального программирования. Пример вопроса представлен в табл. 21.1. Как нетрудно заметить, каждый вопрос включает краткое описание и перечень фактов, которые относятся к команде, полностью освоившей соответствующий метод. Ответы на вопросы оцениваются по шкале от 0 (“не согласен с использованием этого метода”) до 10 (“являюсь фанатом этого метода”).

Таблица 21.1. Пример вопроса на соответствие уровню “шодан”

Приемочные тесты заказчика

Приемочные тесты заказчика призваны гарантировать, что и разработчики, и заказчики знают, чего хотят. Все приемочные тесты должны проводиться до передачи продукта заказчику. *Насколько важны приемочные тесты заказчика для разработки вашего продукта?*

- Приемочные тесты используются для проверки функциональности системы и выполнения требований заказчика.
 - Заказчик сообщает приемочные критерии.
 - Заказчик использует приемочные тесты, чтобы определить, какой результат был получен к концу той или иной итерации.
 - Приемочное тестирование должно быть автоматизированным.
 - Пользовательская история не может считаться завершенной до тех пор, пока не пройдут ее приемочные тесты. Приемочные тесты выполняются автоматически каждый вечер.
-

В результате опроса на соответствие уровню “шодан” получается число в диапазоне от 0 до 100%, указывающее на степень использования командой 15 методов. Этот конечный показатель определяется путем умножения ответов респондентов на фиксированные весовые коэффициенты Кребса, учитывающие важность каждого из этих 15 методов. Меньше всего сказываются на этом конечном показателе ежедневные совещания разработчиков (менее одного процента), парное же программирование оценивается выше всего остального (12,5%).

¹ Шодан в переводе с японского означает “первая степень”. Обычно этот термин используется применительно к боевым искусствам и обозначает бойца, который получил право носить черный пояс самой низкой степени. Этот термин используется также игроками в игру “то”, которым они обозначают сильного игрока-любителя (непрофессионала).

Использование опроса на соответствие уровню “шодан”

Этот опрос, включающий в себя перечень из 15 вопросов, достаточно компактен, чтобы его можно было проводить по завершении каждого спрингта. Однако лично мне кажется, что так часто проводить его не следует. Этот опрос нужно проводить для отслеживания тенденций и поиска областей, которые нуждаются в улучшении. То и другое бывает труднее выявлять при слишком частом проведении опроса. Еще одной проблемой, которую мне приходилось наблюдать, являются команды, которые включают этот опрос в свою ретроспективу спрингтов вместо более развернутого обсуждения, которое должно быть неотъемлемой частью ретроспективы.

На мой взгляд, правильное использование опроса на соответствие уровню “шодан” предполагает ответы всех членов команды на все вопросы примерно раз в полгода и уж никак не чаще одного раза в квартал. Это позволит проводить полезные сравнения результатов, демонстрируемых в разные моменты времени.

Периодически администрируя опрос на соответствие уровню “шодан”, организация может выявлять определенные тенденции, например насколько успешно мы справляемся с приемочными тестами заказчика. Компания может проводить опросы, учитывающие ее специфику (например, включить в опрос дополнительные методы гибкой методологии разработки, имеющие важное значение для данной компании, или изменить весовые коэффициенты Кребса, присваиваемые отдельным методам).

Преимущества и недостатки

Преимуществом опроса на соответствие уровню “шодан” является использование всего лишь 15 вопросов, хотя это количество является обманчивым из-за составной природы каждого вопроса. Недостатком опроса на соответствие уровню “шодан” является то, что его результаты нельзя применить непосредственно. Невысокий результатирующий показатель может оказаться следствием недостаточного соответствия какому-то одному из нескольких пунктов, составляющих данный вопрос. Прежде чем наставник или консультант примет решение о том, как помочь команде добиться улучшения показателя по тому или иному методу, может понадобиться дополнительное исследование.

Схема оценки степени освоения командой гибкой методологии разработки

Кребс, первым предложивший опрос на соответствие уровню “шодан”, и его коллеги из IBM предложили также дополнительные подходы к оценке того, насколько успешно справляются со своей работой команды, применяющие гибкую методологию разработки. Самым известным среди этих подходов является схема оценки степени освоения командой гибкой методологии разработки (Agile Evaluation Framework, или Agile:EF) (Krebs and Kroll, 2008). Agile:EF следует совету гибкой методологии разработки, суть которого заключается в том, чтобы делать все как можно проще. Таким образом, Agile:EF представляет собой не столько схему как таковую, сколько процесс оценки деятельности команд.

Кребс и его соавтор Пер Кролл (Per Kroll) предлагают при использовании этого метода просить членов команды заполнить очень короткую анкету (это можно делать, например, в конце каждого спрингта). Вопросы в этой анкете сформулированы более

лаконично, чем при проведении опроса на соответствие уровню “шодан”. Но, как и в случае опроса на соответствие уровню “шодан”, каждый вопрос относится к какому-то одному из методов гибкой методологии разработки. Ответ на каждый вопрос представляется в форме оценки по шкале от 1 до 10, причем оценка 10 соответствует постоянному (100% случаев) применению рассматриваемого метода, 1 указывает на то, что рассматриваемый метод вообще не применяется, а 5 — что рассматриваемый метод применяется лишь в 50% случаев. Agile:EF не предлагает какую-то определенную совокупность вопросов. Вместо этого авторы данного подхода рекомендуют использовать уже существующие совокупности вопросов. Можно, например, пользоваться вопросами, включенными в опрос на соответствие уровню “шодан”.

На рис. 21.1 показано, в каком виде могут быть представлены результаты оценки Agile:EF. Столбцы на этом рисунке отражают средний результат команды. Более темная и тонкая линия отражает широту разброса мнений. Если вы оцениваете деятельность достаточно большой группы людей, можете вычислить среднеквадратическое отклонение и отображать его этими линиями. Если вы оцениваете деятельность только одной команды, тонкую линию можете использовать для отображения самых низких и самых высоких оценок.

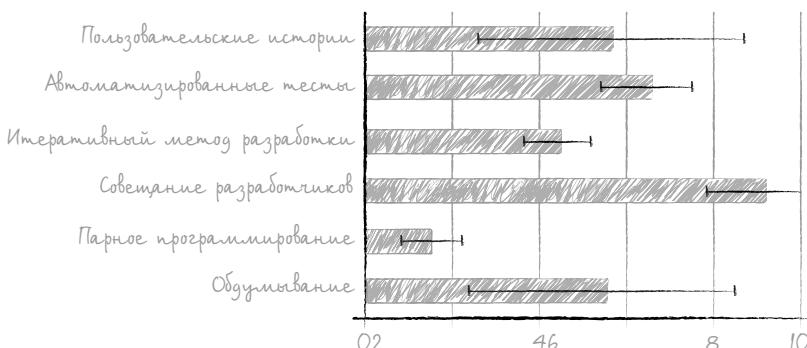


Рис. 21.1. Пример результатов, полученных с помощью опроса Agile:EF. На этом рисунке представлены средний результат и его среднеквадратическое отклонение

Преимущества и недостатки

Поскольку Agile:EF является универсальным подходом, присущие ему преимущества вполне позволяют рекомендовать его к применению. Периодические, быстрые, ненавязчивые оценки, скорее всего, будут характеризоваться более высоким процентом ответивших на вопросы, чем более подробные (и соответственно, требующие больше времени) опросы. Несмотря на то что возможность использовать в случае Agile:EF любую совокупность вопросов является преимуществом этого подхода, она же является и его недостатком, который не позволяет назвать Agile:EF схемой в прямом смысле этого слова.

Несмотря на то что лично мне нравится ориентированность Agile:EF на простоту и быстрые опросы, мой собственный опыт свидетельствует о том, что трудно рассчитывать на получение полезных результатов, задавая команде в конце каждого спринта одни и те же 15 (или что-то около того) вопросов. Вместо этого я рекомендую

пользоваться несколькими совокупностями вопросов, каждый из которых фокусируется на каком-то определенном аспекте процесса разработки программного обеспечения. Используйте это множество совокупностей вопросов циклически, задавая по окончании каждого спринта (или раз в месяц) другую совокупность вопросов. К тому моменту, когда вы будете готовы повторить ту или иную совокупность вопросов, пройдет уже достаточно времени для того, чтобы команда добилась существенных улучшений в областях, охватываемых соответствующим опросом.

Сравнительная оценка степени освоения командой гибкой методологии разработки

Несколько лет тому назад кто-то из моих клиентов начал донимать меня вопросом “Каковы наши успехи?” Каждый раз я отвечал примерно так: “Вы добились впечатляющих успехов в парном программировании; мне нравится также, что команды перешли от составления документов с требованиями к обсуждению пользовательских историй. Но команды еще не усвоили по-настоящему идею автоматизированного тестирования и вот на этом-то вам нужно сейчас сосредоточиться”. Но такой ответ их не устраивал: они хотели знать, “каковы наши успехи *в сравнении с успехами наших конкурентов?*”

Поначалу этот вопрос не давал мне покоя. Я думал: “Ну какая вам разница, каких успехов в освоении гибкой методологии разработки добились ваши конкуренты?” Если вы еще не достигли совершенства в этом деле, продолжайте совершенствоваться. Со временем я, однако, понял, в чем заключается ущербность моего хода мыслей. Компания не стремится быть идеальной. Главное для нее — превзойти (и все время опережать) конкурентов. В наши дни Google является доминирующей поисковой машиной вовсе не потому, что демонстрируемые ею результаты являются идеальными, а потому что эти результаты, как правило, оказываются лучше результатов, демонстрируемых ее конкурентами.

Это означает, что гибкой методологии разработки не нужна модель зрелости уровня 5, подобная СММI. Организации не стремятся к совершенству в плане некоторого идеализированного перечня принципов и методов гибкой методологии разработки. Скорее всего, они пытаются добиться больших успехов в освоении гибкой методологии разработки, чем их конкуренты. Это не означает, что освоение гибкой методологии разработки является самоцелью. Целью остается создание более качественных продуктов, чем у конкурентов. Но если организация добилась больших успехов в освоении гибкой методологии разработки, чем конкуренты, это указывает на ее способность создавать более качественные продукты, делая это быстрее и дешевле.

См. также Пример пятиуровневой модели освоения гибкой методологии разработки можно найти в описании Sidky Agile Maturity Index по адресу <http://www.agilejournal.com/content/view/411/33/>.

Учитывая это обстоятельство, мы с Кенни Рубином (Kenny Rubin) разработали сравнительную оценку степени освоения командой гибкой методологии разработки (Comparative Agility assessment — CA), с которой можно бесплатно ознакомиться в Интернете. Подобно опросу на соответствие уровню “шодан” и Agile:EF, оценка CA может основываться на индивидуальных ответах на определенную совокупность

вопросов. Однако оценка СА разработана таким образом, что может выполняться опытным Scrum-мастером, наставником или консультантом от имени команды или компании, а ее основой является интервью или наблюдение.

ПРИМЕЧАНИЕ

С оценкой СА можно ознакомиться в Интернете на сайте www.ComparativeAgility.com.

Ответы на вопросы могут быть обобщены в масштабе всей организации и сопоставлены с базой данных СА в целом. Эти ответы можно также сопоставить с каким-то из подмножеств базы данных СА. Вы можете, например, сравнить свою команду со всеми другими компаниями, занимающимися веб-разработками, со всеми компаниями, которые приступили к освоению гибкой методологии разработки примерно шесть месяцев назад, либо с компаниями определенной отрасли или использовать какое-то сочетание указанных факторов. Вы можете также сравнить свою команду с ее собственными данными за некоторый прошедший период и выяснить таким образом, каких улучшений команде удалось достичь с тех пор.

На самом высоком уровне подход СА оценивает степень освоения командой гибкой методологии разработки по семи параметрам: командная работа, требования, планирование, технические методы, качество, культура и создание знаний. Частичные результаты, отражающие оценку некоторой команды по трем параметрам, представлены на рис. 21.2. На этом рисунке приведено сравнение одной конкретной команды с совокупностью других команд, представленных в базе данных СА (в данном случае речь идет о других командах, занимающихся веб-разработками). Нуль представляет среднее значение всех соответствующих команд в базе данных СА. Каждая из вертикальных линий, помеченных от -2 до 2, представляет одно среднеквадратическое отклонение от среднего значения. Из рис. 21.2 видно, что эта команда демонстрирует гораздо лучшие результаты, чем в среднем, по параметру "Планирование", несколько лучшие результаты, чем в среднем, по параметру "Требования" и значительно худшие результаты, чем в среднем, по параметру "Качество".

Каждый из трех параметров, представленных на рис. 21.2, включает в себя три-четыре характеристики. Чтобы оценить команду по каждой из них, задается определенная совокупность вопросов. Например, характеристики параметра "Планирование" включают следующие вопросы.

- Когда выполняется планирование
- Кто участвует в планировании
- Выполняется ли планирование как выпусков продукта, так и спринтов
- Фиксируются ли критические переменные (такие, как масштаб, календарный план и ресурсы)
- Как отслеживается достигнутый прогресс

Вопросы для характеристики "когда выполняется планирование" представлены в табл. 21.2. Как видно из этой таблицы, ответы на вопросы указываются в соответствии со следующей шкалой: *да; скорее да; ни да, ни нет; скорее нет; нет и нет ответа*.

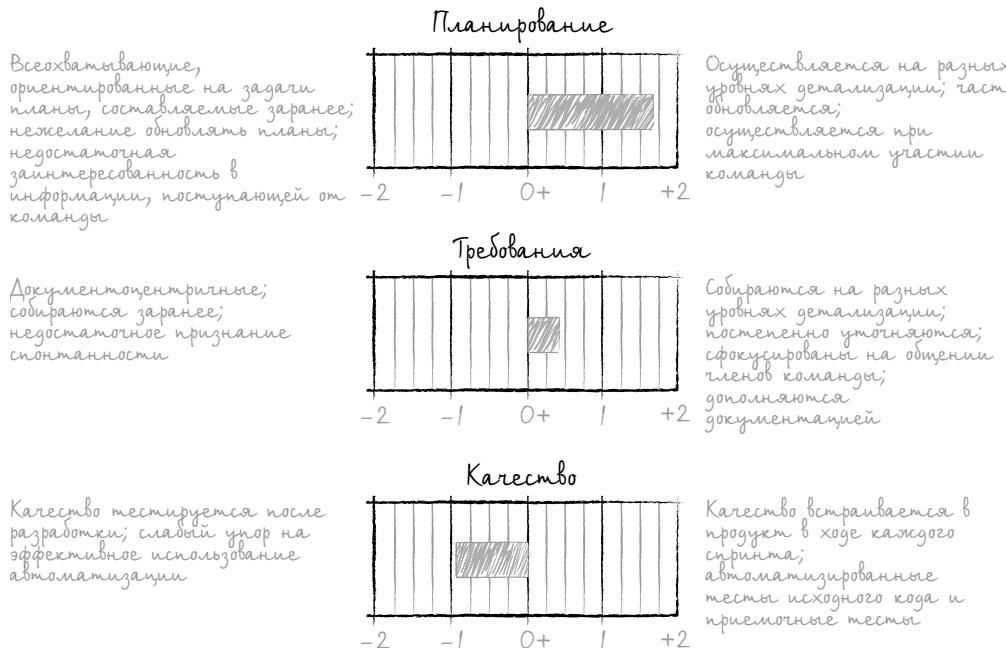


Рис. 21.2. Результаты сравнительной оценки степени освоения командой гибкой методологии разработки показывают, что эта команда действует лучше, чем в среднем, по параметрам “Планирование” и “Требования”, но хуже по параметру “Качество”

Таблица 21.2. Одной из характеристик, рассматриваемых при получении сравнительной оценки степени освоения командой гибкой методологии разработки, является характеристика “когда выполняется планирование”

	Да	Скорее да	Ни да, ни нет	Скорее нет	Нет	Нет ответа
Заблаговременное планирование полезно и не является излишеством						
Члены команды покидают совещания по планированию, зная, что должно быть сделано, и в уверенности в том, что им удастся справиться со своими обязательствами						
Команды обсуждают необходимость изменения даты выпуска или масштаба, как только в этом возникает потребность						
Усилия, затраченные на планирование, распределяются приблизительно равномерно по всему времени выполнения проекта						

Преимущества и недостатки

Преимуществом сравнительной оценки степени освоения командой гибкой методологии разработки является то, что она быстрее приводит к результатам, имеющим

реальную практическую ценность. Подобно другим оценкам, эти результаты указывают, в первую очередь, на тот или иной недостаток в одном из семи параметров, но, в отличие от других оценок, углубленный анализ этого параметра позволяет выявить конкретную характеристику, которая является проблемной для данной организации. Это должно помочь команде или ее Scrum-мастера определить меры, которые нужно предпринять для исправления ситуации. Например, проанализировав низкую оценку качества на рис. 21.2, можно увидеть, что у параметра “Качество” есть три характеристики: автоматизированные тесты исходного кода, приемочные тесты заказчика и временные рамки. Поскольку подход СА позволяет оценить каждую характеристику индивидуально, есть возможность понять, какая из этих характеристик снижает оценку качества для организации в целом.

Предполагалось, что сравнительная природа оценки СА должна стать ее основным преимуществом. Сравнив свою организацию с другими организациями, действия по совершенствованию можно сосредоточить на самых многообещающих областях.

Самым заметным недостатком СА является масштаб опроса. Опрос включает примерно 125 вопросов, касающихся процесса разработки. Существует два типичных решения.

- Выполнять всестороннюю оценку лишь один-два раза в году (для некоторых организаций приемлемой и подходящей может быть ежеквартальная оценка)
- Каждый месяц оценивать лишь один из семи параметров

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- Если вам никогда не приходилось проводить опросы какой-либо из своих команд, использующих гибкую методологию разработки, проведите такой опрос сейчас. Не ожидайте шесть недель, пока вы разработаете “идеальный” опрос. Проанализируйте вопросы, содержащиеся в сравнительной оценке степени освоения командой гибкой методологии разработки или в опросе на соответствие уровню “шодан”, и выберите 15 вопросов, которые, как вам кажется, дадут вам полезную или интересную информацию и послужат отправной точкой.
- Ожидая поступления результатов от этой первой совокупности вопросов, продумайте свою долгосрочную стратегию и разработайте самостоятельно (или выберите) подход, которым вы будете пользоваться регулярно.

Разработка собственной оценки

Некоторые организации предпочитают разрабатывать собственные оценки. Для этого, разумеется, придется дополнитель но поработать, однако преимущество такой оценки заключается в том, что ее можно полностью приспособить к особенностям вашей организации. Организации с масштабными Scrum-инициативами, такие как Ultimate Software, JDA Software, Yahoo! и Salesforce.com, воспользовались этим подходом. Если вы предпочтете такой вариант, обязательно начните с вопросов, содержащихся в таких подходах, как опрос на соответствие уровню “шодан” и сравнительная оценка степени освоения командой гибкой методологии разработки. Исключите вопросы, неактуальные для вашей организации или неспособные, по вашему мнению, дать интересные для вас результаты. Если, например, автоматизированное тестирование является в вашей организации устоявшейся практикой,

успешно используемой всеми вашими командами, вы можете сэкономить время, исключив соответствующие вопросы.

Большинство известных мне оценок, специфических для той или иной конкретной компании, включают значительное число вопросов, призванных выявить мнение работников о соответствующем изменении. В Yahoo!, например, задавали вопросы относительно перехода к Scrum, подобные указанным ниже.

- Как, на ваш взгляд, изменилась производительность вашей команды после перехода к использованию Scrum?
- Как, на ваш взгляд, изменился моральный климат в вашей команде после перехода к использованию Scrum?
- Как изменилось ваше чувство ответственности и чувство вашей личной заинтересованности в успешном выполнении проекта после внедрения Scrum?
- Как изменилась степень сотрудничества и взаимопомощи в команде после внедрения Scrum?
- Что вы думаете в целом о качестве продуктов, разработанных после внедрения Scrum?

Шкала ответов была такой: *Scrum гораздо хуже традиционного метода разработки, Scrum хуже традиционного метода разработки, Scrum практически ничем не отличается от традиционного метода разработки, Scrum лучше традиционного метода разработки и Scrum гораздо лучше традиционного метода разработки.*

Аналогичные вопросы задавала и компания Salesforce.com во время своего перехода к Scrum.

- Считаете ли вы Scrum эффективной методологией разработки?
- Как повлияло внедрение Scrum на качество наших продуктов?
- Порекомендовали бы вы Scrum своим коллегам (как в вашей компании, так и в других компаниях)?

Помимо этих качественных показателей, Salesforce.com занималась сбором ряда довольно простых, но эффективных количественных показателей. Это позволило компании рассматривать свой переход к использованию Scrum с разных точек зрения (впрочем, это является темой следующего раздела).

ВОЗРАЖЕНИЕ

“Мне не нравятся подобные вопросы. Кому какое дело до того, что думают работники о своей производительности и о том, улучшилось ли качество?”

Именно те, кто непосредственно занимаются разработкой программного обеспечения, в состоянии по достоинству оценить изменения в методах их работы. Конечно, они могут проявить необъективность, если, например, руководство заранее объявит о выплате премий в случае, если после внедрения Scrum удвоится производительность. Но я и не предлагал ничего такого. Ничего подобного не предлагали также компании, которые использовали эти вопросы. В отсутствие каких-либо стимулов, способных повлиять на правдивость ответов, большинство работников будет отвечать честно. А их ответы на вопросы, подобные рассмотренным, могут оказаться не менее полезными, чем ответы заказчиков на столь же субъективные вопросы.

Сбалансированная система показателей Scrum-команд

Хорошо известно, что если мы вводим какой-либо новый показатель и сообщаем командам о том, что их работа будет оцениваться по этому показателю, они постараются изменить свое поведение таким образом, чтобы оптимизировать этот показатель. Сообщите команде, что ее работа будет оцениваться по количеству обнаруженных системой отслеживания дефектов, и этот показатель начнет снижаться — возможно, вследствие разумных усовершенствований или потому, что команде удастся изыскать способы неформального информирования о дефектах, позволяющего “обойти” систему отслеживания. Даже если бы мы могли придумать показатель, который невозможно обойти тем или иным способом, один показатель не в состоянии отразить полную картину деятельности команды. Возможно, команде удалось снизить количество дефектов за счет резкого снижения производительности, и она разрабатывает на 90% меньше функциональных возможностей, чем прежде, потому что каждая строка кода подвергается доскональной проверке. Нам требуется более *сбалансированный* взгляд, чем тот, который обеспечивается каким-то одним показателем.

Идея сбалансированного взгляда на организацию привела Роберта Каплана (Robert Kaplan) и Дэвида Нортона (David Norton) к созданию так называемой *сбалансированной системы показателей*. Их идея заключается в следующем: чтобы как можно лучше уяснить функционирование той или иной компании, нельзя ограничиваться лишь анализом отчета о прибылях и убытках и балансового отчета, которые представляют собой лишь два показателя того, как функционирует соответствующая компания. Рассмотрение одних лишь финансовых отчетов компании дает ничуть не более полное представление об успешности функционирования этой компании, чем представление об успешности функционирования организации-разработчика, получаемое на основе лишь количества дефектов в базе данных выявленных дефектов. Каплан и Нортон предлагают рассматривать любую компанию с четырех точек зрения: финансы, клиент, бизнес-процессы, а также обучение и рост. Эти четыре точки зрения составляют в совокупности сбалансированную систему показателей (1992).

Начиная с 2000 года я использую сбалансированные системы показателей как способ, с помощью которого группы разработчиков программного обеспечения могут оценивать свою деятельность с разных точек зрения. Первоначальные четыре точки зрения, предложенные Капланом и Нортоном, вовсе необязательно являются оптимальными в случае их непосредственного применения к отделу разработки программного обеспечения, группе информационных технологий или в особенности к отдельно взятой команде. В течение ряда лет я экспериментировал с разными точками зрения и пришел к выводу, что наилучшими из них являются перечисленные ниже четыре точки зрения, предложенные Лиз Барнетт (Liz Barnett) из Forrester Research.

- **Операционное совершенство.** Команды стремятся создавать высококачественные продукты с максимальной для себя производительностью, не выходя при этом за рамки указанных для них затрат и сроков.
- **Ориентация на пользователя.** Команды сосредоточиваются на разработке функциональных возможностей, указанных пользователями и заказчиками.

- **Экономическая ценность.** Команды обеспечивают ценность для соответствующего бизнеса в форме экономии затрат, повышенных доходов или какими-либо иными подобными способами.
- **Ориентация на будущее.** Создавая новые продукты и разрабатывая новые функциональные возможности сегодня, команды приобретают квалификацию, которая пригодится им в будущем (Barnett, Schwaber, and Hogan, 2005).

Если сбалансированная система показателей создается примерно в то же время, когда вы приступаете к внедрению Scrum, а команда, отдел или организация затем периодически оценивается по этой системе показателей, то прогресс должен быть налицо. Команда, которая действует успешно в условиях использования Scrum, должна проявить способность к одновременному улучшению по каждой из перечисленных выше точек зрения. Более того, сбалансированная система показателей переключает внимание команды с полной сосредоточенности на освоение гибкой методологии разработки на достижение именно тех целей, которые заставили данную организацию приняться за освоение гибкой методологии разработки путем внедрения Scrum.

Построение сбалансированной системы показателей

Каждая из точек зрения на сбалансированную систему показателей подкрепляется, как правило, одной-четырьмя стратегическими целями. Прогресс в достижении каждой цели измеряется как опережающими, так и запаздывающими индикаторами, для которых заранее указываются целевые значения. Если, например, мы рассматриваем точку зрения “Операционное совершенство”, то можно было бы определить такие цели, как повышение производительности, повышение качества, получение более точных оценок, снижение совокупных затрат на разработку и т.д. Совокупность выбранных вами целей не должна представлять собой подробный перечень замечательных целей; напротив, выбирать нужно лишь те цели, на которых можно сосредоточиться.

Для каждой цели важно определить такие показатели, которые скажут нам, достигаем ли мы этой цели (или, может быть, уже достигли ее). Хотя вы должны определить по меньшей мере один показатель для каждой цели, лучше все же определить по меньшей мере один опережающий индикатор и один запаздывающий индикатор. Опережающий индикатор — это показатель, который, как вы рассчитываете, изменится до достижения соответствующей цели. Если, например, речь идет о такой цели, как повышение качества, то опережающим индикатором является количество записанных контрольных примеров. Наличие большего количества контрольных примеров не гарантирует повышения качества соответствующего продукта, но это количество может быть хорошим показателем повышения качества данного продукта.

Напротив, запаздывающий индикатор представляет собой показатель, который изменяется уже после достижения цели, или показатель, который можно измерить только в это время. Продолжая наш пример с повышением качества, отметим, что запаздывающим индикатором могло бы быть количество дефектов, выявленных заказчиками в официально выпущенной версии продукта. Запаздывающие индикаторы обычно представляют собой показатели, используемые для выяснения, была ли в действительности достигнута цель. Однако их недостатком является, конечно же, то обстоятельство, что измерить их можно лишь после достижения соответствующей цели. Именно поэтому оптимальным вариантом представляется сочетание опережающих и запаздывающих индикаторов. В табл. 21.3 приведены примеры целей, опережающие индикаторы и запаздывающие индикаторы для точки зрения “Операционное совершенство”.

Таблица 21.3. Сбалансированная система показателей позволяет получить разные точки зрения на функционирование команды

Точка зрения	Цель	Опережающие индикаторы	Запаздывающие индикаторы
Операционное совершенство	Повышение производительности	Процент нереализованных элементов журнала запросов на выполнение работ (цель — 5–15%) Процент проверок исходного кода в выходные дни (цель — менее 5%)	Количество разработанных функциональных возможностей в расчете на одного разработчика (цель — 20-процентное увеличение)
	Предсказуемость календарных планов		Количество проектов, завершенных в пределах от -1 до +2 sprintов, по сравнению с тем, что было предсказано в средине процесса реализации проекта (цель — 95%)
	Повышение качества	Процент успешно выполненных тестов в непрерывных сборках (цель — 95%)	Количество дефектов, выявленных в течение первых 30 дней после официального выпуска (цель — 50-процентное сокращение)
	Увеличение времени функционирования		Время простоя сервера (запланированное + незапланированное) — менее 120 минут в год
Ориентация на пользователя	Повышение степени удовлетворенности пользователей	Увеличение процента ответивших в фокус-группе заказчика (цель — увеличение процента ответивших по электронной почте на 20%)	Нетто-оценка промоутера (цель — повысить на 25%)
	Более частые официальные выпуски	Графики выполнения оставшихся до выпуска работ составляются и отображаются для всех проектов (цель — 100%)	Улучшенные оценки по результатам ежеквартального опроса заказчиков (цель — 80% отвечают "превосходит ожидания" или "намного превосходит ожидания")
	Больше функциональных возможностей в выпусках		По меньшей мере один основной выпуск ежеквартально (цель — интервал между основными выпусками составляет не более 90 дней)
	Повышение степени удовлетворенности работников	Количество жалоб на работников компании (цель — 1 в месяц)	Количество видимых пользователю элементов журнала запросов на выполнение работ в расчете на один выпуск (цель — 300)
Ориентация на будущее	Повышение степени удовлетворенности работников		Процент работников, утверждающих, что им доставляет удовольствие (или огромное удовольствие) работать здесь (цель — 80%)
	Улучшение понимания Scrum и гибкой методологии разработки	Участие во всевозможных конференциях, посвященных гибкой методологии разработки (цель — обеспечить в этом году участие по меньшей мере 40 человек в конференциях, посвященных гибкой методологии разработки)	Количество работников, готовых порекомендовать Scrum кому-либо из своих знакомых в какой-либо другой компании (цель — 80%)

Отдавайте предпочтение простым показателям

Знакомясь с табл. 21.3, вы, наверное, заметили, что некоторые показатели довольно просты. Наверняка вы подумали, что количество разработанных функциональных возможностей в расчете на одного разработчика (первый из запаздывающих индикаторов в табл. 21.3) слишком просто, чтобы быть полезным. Что мы подразумеваем, когда говорим о функциональной возможности? Можно ли говорить о “функциональной возможности вообще”? Ведь бывают простые функциональные возможности и сложные функциональные возможности. Вообще говоря, простые показатели наподобие этого могут быть полезны, если их сравнение проводится за достаточно продолжительный период времени и если они рассматриваются в сочетании с другими простыми показателями.

Количество разработанных функциональных возможностей в расчете на одного разработчика является одним из простых показателей, которые, как уже упоминалось, использовались Salesforce.com для оценивания выгод от внедрения Scrum в этой компании. Спустя двенадцать месяцев после начала перехода к Scrum компания выявила 38-процентный рост количества разработанных функциональных возможностей в расчете на одного разработчика за предшествующий год. Можно ли допустить, что одной из причин такого увеличения явилось то, что сами функциональные возможности оказались меньше? Разумеется, можно. Но столь простой показатель, как этот, позволил им охарактеризовать количественно общее ощущение роста производительности по сравнению с предшествующим годом. Сопоставим этот показатель с целью измерений, о которой мы говорили в начале этой главы, — снижением неопределенности. До выполнения этого измерения члены сообщества в поддержку перехода к Scrum (ETC) компании Salesforce.com, возможно, не были уверены в том, что применение Scrum будет способствовать росту производительности команд. Выполнив это измерение, они получили ответ на свой вопрос.

Увеличение потока функциональных возможностей к заказчикам продемонстрировал еще один простой показатель, которым воспользовалась Salesforce.com. На рис. 21.3 представлена диаграмма *кумулятивной ценности*, построенная Стивом Грини (Steve Greene) и Крисом Фраем (Chris Fry) (эта же диаграмма была приведена в главе 1, “Гибкая методология разработки: нелегко, но перспективно”). Учитывая каждую из функциональных возможностей безотносительно ее величине, сложности или важности, эта диаграмма показывает, сколько новых функциональных возможностей было разработано и когда именно они были разработаны. Общая идея заключается в том, что область под кривыми отражает совокупную ценность соответствующих функциональных возможностей для пользователей — чем раньше разработана функциональная возможность, тем большую ценность она представляет для пользователя. В 2006 году, т.е. до внедрения Scrum, Salesforce.com не поставила своим пользователям ни одной новой функциональной возможности вплоть до января 2007 года. Сравните общую площадь под линией 2006 года с общей площадью под линией 2007 года: применение компанией Salesforce.com методологии Scrum в течение всего этого времени привело к более частым выпускам и увеличению совокупного количества функциональных возможностей, поставленных заказчикам.

Было бы не так уж сложно указать на недостатки простых показателей: не все функциональные возможности характеризуются одинаковой сложностью; не все функциональные возможности одинаково важны для всех заказчиков; в 2007 году

в составе команды было больше разработчиков, чем в 2006 году, и т.д. Несмотря на справедливость каждого из этих замечаний, ни одно из них не в состоянии серьезно преуменьшить полезность диаграммы, подобной той, которая представлена на рис. 21.3, для демонстрации выгод, которые обеспечило компании Salesforce.com внедрение Scrum.



Рис. 21.3. Спустя год после перехода к Scrum Salesforce.com воспользовалась этой весьма полезной диаграммой, чтобы проиллюстрировать 568-процентное улучшение простого показателя, названного этой компанией кумулятивной ценностью

ПОПРОБУЙТЕ ПРЯМО СЕЙЧАС

- ❑ Составьте список простых показателей, которые вам будет нетрудно собрать и которые позволят получить ответы на ряд развернутых вопросов, касающихся внедрения Scrum в вашей организации. Приступите к сбору этих показателей.
- ❑ Составьте сбалансированную систему показателей. Она может касаться определенного проекта, отдела или даже перехода к использованию Scrum. Начните с определения целей (каких целей вы хотели бы достичь?), затем определите для каждой из них один или более опережающих индикаторов, которые могут быть использованы для выяснения, находитесь ли вы на пути к достижению соответствующей цели. Наконец, определите для каждой цели по меньшей мере один запаздывающий индикатор, который может использоваться для выяснения, была ли достигнута соответствующая цель.

Должно ли это нас волновать

Сбор метрик (даже если речь идет о простых показателях) связан с затратами сил и времени и явно не относится к числу достижений, которыми могут похвастаться компании, занимающиеся разработкой программного обеспечения. Учитывая

необходимость дополнительных затрат сил и времени, есть ли смысл заботиться о сборе данных, который позволит понять, насколько успешно мы осваиваем гибкую методологию разработки? Да, смысл в этом есть. Я могу указать три основные выгоды, которые дает такой сбор данных.

- **Показатели помогают преодолевать силу организационной гравитации.** Как уже указывалось в главе 2, “ADAPТация к Scrum”, существуют определенные причины сохранения ситуации, имевшейся до начала внедрения Scrum. Организационная гравитация стремится вернуть нас к этому статус-кво. Периодические оценки и показатели, демонстрирующие выгоды перехода к Scrum, окажутся одним из наилучших средств, которые помогут преодолеть силу организационной гравитации.
- **Показатели помогают “рекламировать” переход к использованию Scrum.** Разумная программа оценивания помогает не только преодолевать силу организационной гравитации, но и демонстрировать преимущества Scrum другим командам, которые проявляют интерес к внедрению Scrum. Вспомните еще раз о модели ADAPТ, о которой рассказывалось в главе 2, “ADAPТация к Scrum”. Мы говорили о важности пропаганды использования Scrum путем распространения опыта успешного применения этой методологии. Чтобы заинтересовать другие команды в переходе к Scrum, очень важно рекламировать первые успехи вашей команды на этом пути. Показатели помогают выразить эти успехи количественно.
- **Показатели помогают понять, на чем следует сосредоточить усилия для дальнейшего совершенствования.** Показатели должны приводить к определенным действиям. Если собираемые вами данные не приводят к каким-то действиям и решениям, нужно прекратить сбор этих данных и начать сбор других данных. Разумная программа оценивания поможет выявить области, нуждающиеся в совершенствовании (а также области, в которых достигнуты несомненные успехи). Если вы обнаружите, что по тем или иным показателям команда начала работать хуже, чем прежде, постарайтесь понять причину этого. Если окажется, что какие-то команды приняли на вооружение тот или иной новый метод, который позволил добиться более высоких результатов, чем у других команд, изыщите способ внедрить этот метод и в других командах.

Несмотря на то что я уверен в полезности и важности сбора показателей, характеризующих успешность освоения командой Scrum и гибкой методологии разработки, я хотел бы завершить эту главу предостережением. Какие бы данные вы ни собирали, в организации обязательно найдется хотя бы один человек, который привяжется к ним сверх всякой разумной меры и станет придавать им слишком большое значение. Эти данные будут затем использоваться для “подстегивания” команд. Я далек от того, чтобы утверждать, будто показатели ни в коем случае нельзя использовать в качестве кнута, но показатели, речь о которых шла в этой главе, имеют важную особенность: они должны использоваться для того, чтобы сфокусировать усилия команды и способствовать лучшему пониманию ситуации, а вовсе не для того, чтобы выдвигать против людей обвинения в чем-либо или принуждать их к чему-либо.

Дополнительная литература

Gilb, Tom. 2005. *Competitive Engineering: A handbook for systems engineering, requirements engineering, and software engineering using planguage*. Butterworth-Heinemann.

Гилб, изобретатель Evo-процесса, использует строгий подход к измерениям. В его книге приводятся многочисленные примеры квантификации и измерения того, что, на первый взгляд, вообще не поддается измерению. Подход Гилба особенно полезен, когда используется для определения опережающих и запаздывающих индикаторов сбалансированной системы показателей.

Hubbard, Douglas W. 2007. *How to measure anything: Finding the value of “intangibles” in business*. Wiley.

Замечательная книга, которая полностью соответствует своему названию.² В ней излагаются способы измерения тех или иных величин. Предлагаемый для этого математический аппарат изложен автором во вполне доступной форме, что позволяет его использовать даже неискушенному читателю. Если вы не знаете, как измерить что-либо, прочтите эту книгу — и получите массу идей для решения своей проблемы.

Kaplan, Robert S., and David P. Norton. 1992. The balanced scorecard: Measures that drive performance. *Harvard Business Review*, January–February, 71–79.

Эта статья дала начало движению за применение сбалансированной системы показателей. После ее опубликования вышла в свет книга тех же авторов, однако эта короткая статья остается наилучшим кратким введением в данную тему.

Krebs, William, and Per Kroll, 2008. Using evaluation frameworks for quick reflections. *Agile Journal*, February 9. <http://www.agilejournal.com/articles/columns/column-articles/750-using-evaluation-frameworks-for-quick-reflections>.

Эта статья является самым лучшим введением в подход Agile:EF, используемый для оценки прогресса команды.

Leffingwell, Dean. 2007. *Scaling software agility: Best practices for large enterprises*. Addison-Wesley Professional.

В главу 22 этой книги включена дополнительная информация об использовании сбалансированной системы показателей. Кроме того, в ней содержится командно-ориентированный опрос, касающийся оценки степени освоения командой гибкой методологии разработки и подобный опросам, речь о которых шла в данной главе. С этой оценкой можно ознакомиться по адресу <http://scalingsoftwareagility.files.wordpress.com/2008/01/team-agility-assessment-in-pdf>.

Mair, Steven. 2002. A balanced scorecard for a small software group. *IEEE Software*, November/December, 21–27.

² Название книги Хаббарда в переводе звучит так: «*Как измерить что угодно: определение значений «нематериальных величин» в бизнесе*». — Примеч. пер.

В этой статье излагается предыстория метода сбалансированной системы показателей. В ней показано также, как создать сбалансированную систему показателей для небольшого подразделения, основываясь на четырех исходных точках зрения, предложенных Капланом и Нортоном.

Williams, Laurie, Lucas Layman, and William Krebs. 2004. Extreme programming evaluation framework for object-oriented languages, version 1.4. North Carolina State University Department of Computer Science, TR-2004-18.

В этом техническом отчете содержится описание схемы оценки степени освоения командой экстремального программирования (Extreme Programming: Evaluation Framework, или XP:EF), которая, основываясь на опросе на соответствие уровню “шодан”, включает контекст проекта и показатели достигнутых результатов. Дополнительные данные, собранные авторами отчета, могут представлять интерес для исследователей или даже для отделов управления проектами, заинтересованных в получении более строгих оценок. Однако команда, заинтересованная лишь в собственном совершенствовании, может ограничиться опросом “шодан”, который положен в основу XP:EF и полностью описан в приложении.

Глава 22

Впереди еще много работы

Мы с вами прошли долгий путь. Надеюсь, вы приняли на вооружение если не все, то по крайней мере большую часть рекомендованных мною методов и выполнили действия, описанные в рубрике “Попробуйте прямо сейчас”. Если мои надежды не обманывают меня, то вы, наверное, уже добились значительного прогресса. Вы наверняка создали в своей организации сообщество в поддержку перехода к Scrum (ETC), которое будет способствовать внедрению Scrum в вашей организации. ETC, в свою очередь, сформировало среду, способствующую созданию и развитию сообществ в поддержку усовершенствования организации. Одни из этих сообществ в поддержку усовершенствования организации — после того как было достигнуто все, что они намеревались усовершенствовать — уже расформированы, а другие, напротив, либо расширили круг своих интересов, либо по-прежнему боятся над реализацией возможностей по усовершенствованию, осуществить которые оказалось не так-то просто.

К настоящему времени методология Scrum стала стандартным способом выполнения проектов по крайней мере для некоторых команд в вашей организации. Внедрение Scrum началось с того, что сотрудники этих команд осознали необходимость изменения используемого ими способа разработки программного обеспечения. По мере дальнейшего углубления этого понимания оно постепенно превратилось в желание разрабатывать программное обеспечение по-другому. В ответ у людей появилась способность разрабатывать программное обеспечение посредством гибкой методологии разработки. Это привело к первым успехам в деле применения Scrum. Информация об этих успехах была доведена до сведения других команд, чтобы они начали собственные циклы осознания, возникновения желания и способности. Наконец результаты применения методологии Scrum были перенесены и на другие части вашей организации, чтобы организационная гравитация не привела каждого из сотрудников и каждой из команд обратно к тому, с чего они начинали.

В ходе внедрения Scrum вы изменили не только должностные инструкции членов команды, но и само видение ими своих ролей в команде. Команды теперь структурированы на основе реализации функциональных возможностей, а не на основе технологий или уровней архитектуры. Несмотря на предоставленную командам

возможность самоорганизации, лидеры оказывают на них некоторое (хотя и менее значительное) влияние; впрочем, сами лидеры уже научились работать с самоорганизующимися командами. Члены команды освоили новые технические приемы и методы, которые помогают им писать более высококачественный код. Команда осознала важность своего журнала запросов на выполнение работ, научилась эффективно работать в рамках Scrum-спринтов, жестко ограниченных временными рамками, и выполнять планирование в условиях недостатка информации. Наверняка повысилось качество — вам удалось не только включить тестеров в процесс разработки, но и добиться, чтобы программисты оказывали помощь в создании автоматизированных тестов.

К настоящему времени вам, наверное, уже удалось применить методологию Scrum к более крупным проектам или проектам, охватывающим несколько городов или даже континентов. Вероятно, вы научились справляться со специфическими проблемами, возникающими при реализации таких проектов. Сейчас методология Scrum более глубоко интегрирована в вашу организацию и даже может сосуществовать и конкурировать с другими корпоративными нормами, такими как соответствие CMMI или ISO 9001. Сотрудники отдела кадров, группы технического обеспечения и отдела управления проектами стали теперь вашими союзниками в деле обеспечения устойчивых и необходимых перемен. Иными словами, ваша организация добилась огромного прогресса.

Однако это вовсе не повод для того, чтобы остановиться!

Не важно, в какой мере вы освоили гибкую методологию разработки и Scrum. Как бы вы ни были сильны в Scrum сегодня, если вы не выйдете на более высокий уровень ее использования в следующем месяце, считайте, что вы уже утратили гибкость в разработке программного обеспечения. Совершенствоваться нужно *всегда*.

Вы можете выдвинуть последнее возражение: моя цель вовсе не в том, чтобы до бесконечности совершенствоваться в освоении гибкой методологии разработки. Моя цель — создавать высококачественные продукты. Если я достиг определенного, достаточно высокого, уровня в освоении гибкой методологии разработки, почему я не могу остановиться? На это возражение я могу ответить так: разумеется, ваша цель вовсе не в том, чтобы до бесконечности совершенствоваться в деле освоения гибкой методологии разработки. Ваша главная цель действительно состоит в том, чтобы создавать высококачественные продукты, делая это как можно быстрее и с как можно меньшими затратами. Однако для этого, как мне кажется, вам нужна гибкость при разработке программного обеспечения. А чтобы проявлять такую гибкость, необходимо постоянно совершенствоваться.

Непрерывное совершенствование — не такое уж сложное дело, как может показаться на первый взгляд. Вы уже посеяли семена, необходимые для такого совершенствования. Ваши сообщества в поддержку усовершенствования организации будут играть ключевую роль в культуре, толерантной к риску, способствующей генерируанию новых идей и обучению, — культуре, которую формирует и поддерживает сообщество в поддержку перехода к Scrum. Путем экспериментирования, методом проб и ошибок эти сообщества поведут вашу организацию к освоению все более и более эффективных способов работы. Главное, однако, в том, что эти сообщества пробудят в ваших работниках заинтересованность и энтузиазм. Это, в свою очередь, позволит вашей организации перейти от анализа проблем к анализу благоприятных возможностей.

При условии непрерывного совершенствования гибкая методология разработки обязательно принесет вам успех!

Список литературы

- Adler, Paul S., Avi Mandelbaum, Vien Nguyen, and Elizabeth Schwerer. 1996. Getting the most of your product development process. *Harvard Business Review*, March-April, 13–151.
- Adzic, Gojko. 2009. *Bridging the communication gap: Specification by example and agile acceptance testing*. Neuri Limited.
- Allen-Meyer, Glenn. 2000a. *Nameless organizational change: No-hype, low-resistance corporate transformation*. Syracuse University Press.
- Allen-Meyer, Glenn. 2000b. *Overview: Nameless organizational change: No-hype, low-resistance corporate transformation*. Работа размещена на сайте <http://www.nameless.org>.
- Allen-Meyer, Glenn. 2000c. 21st century schizoid change; *OD Practitioner* 32 (3): 22–26.
- Ambler, Scott. 2008a. Agile adoption rate survey, February. <http://www.ambyssoft.com/surveys/agileFebruary2008.html>.
- Ambler, Scott. 2008b. Scott Ambler on agile's present and future. Interview by Floyd Marinescu. InfoQ website, December 1. <http://www.infoq.com/interviews/Agile-Scott-Ambler>.
- Ambler, Scott. Без указания даты. Agile Data Home Page. <http://www.agiledata.org>.
- Ambler, Scott W., and Pramod J. Sadalage. 2006. *Refactoring databases: Evolutionary database design*. Addison-Wesley.
- Anderson, Philip. 1999. Seven levers for guiding the evolving enterprise. В книге *The biology of business: Decoding the natural laws of enterprise*, ed. John Henry Clippinger III, 113–152. Jossey-Bass.
- Appelo, Jurgen. 2008. We increment to adapt, we iterate to improve. *Methods & Tools*, Summer, 9–22.
- Armour, Phillip G. 2006. Software: Hard data. *Communications of the ACM*, September, 15–17.
- Avery, Christopher M. 2005. Responsible change. *Cutter Consortium Agile Project Management Executive Report* 6 (10): 1–28.
- Avery, Christopher M., Meri Aaron Walker, and Erin O'Toole. 2001. *Teamwork is an individual skill: Getting your work done when sharing responsibility*. Berrett-Koehler Publishers.
- Babinet, Eric, and Rajani Ramanathan. 2008. Dependency management in a large agile environment. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 401–406. IEEE Computer Society.
- Bain, Scott L. 2008. *Emergent design: The evolutionary nature of professional software development*. Addison-Wesley Professional.

- Barnett, Liz. 2005. Metrics for agile development projects: Emphasize value and customer satisfaction. With Carey Schwaber and Lindsay Hogan. Forrester. <http://www.forrester.com/Research/Document/Eccerpt/0,7211,37380,00.html>.
- Barnett, Liz. 2008. Incremental agile adoption. *Agile Journal*, February 11. <http://agilejournal.com/articles/columns/from-the-editor-mainmenu-45/755-incremental-agile-adoption>.
- Beavers, Paul A. 2007. Managing a large “agile” software engineering organization. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 296–303. IEEE Computer Society.
- Beck, Kent. 2002. *Test-driven development: By example*. Addison-Wesley Professional.
- Beck, Kent, and Cynthia Andres. 2004. *Extreme programming explained*. 2nd ed. Addison-Wesley Professional.
- Beck, Kent, and Cynthia Andres. 2005. Getting started with XP: Toe dipping, racing dives, and cannonballs. Доступно в виде PDF-файла на сайте Three Rivers Institute по адресу www.threeriversinstitute.org/Toe%20Dipping.pdf.
- Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Manifesto for agile software development. <http://www.agilemanifesto.org/>.
- Benefield, Gabrielle. 2008. Rolling out agile in a large enterprise. В сборнике *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, 461–470. IEEE Computer Society.
- Boehm, Barry W. 1981. *Software engineering economics*. Prentice Hall.
- Boehm, Barry, and Richard Turner. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, September/October, 30–39.
- Bos, Eric, and Christ Vriens. 2004. An agile CMM. В сборнике *Extreme Programming and Agile Methods: XP/Agile Universe 2004*, ed. C. Zannier, H. Erdogmus, and L. Lindstrom, 129–138. Springer.
- Bradner, E., G. Mark, and T.D. Hertel. 2003. Effects of team size on participation, awareness, and technology choice in geographically distributed teams. В сборнике *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 271a. IEEE Computer Society.
- Bridges, William. 2003. *Managing transitions: Making the most of change*. 2nd ed. Da Capo Press.
- Brodwall, Johannes. 2008. An informative workplace. *Thinking inside a bigger box*, November 23. <http://brodwall.com/johannes/blog/2008/11/23/an-informative-workplace/>.
- Brooks, Frederick P. 1995. *The mythical man-month: Essays on software engineering*. 2nd ed. Addison-Wesley Professional. (Orig. pub. 1975.)
- Campbell, Donald T. 1965. Variation and selective retention in socio-cultural evolution. В сборнике *Social change in developing areas: A reinterpretation of evolutionary theory*, ed. Herbert R. Barringer, George I. Blanksten, and Raymond W. Mack, 19–49. Schenkman.
- Cao, Lan, and Balasubramaniam Ramesh. 2008. Agile requirements engineering practices: An empirical study. *IEEE Software*, January/February, 60–67.

- Carmel, Erran. 1998. *Global software teams: Collaborating across borders and time zones*. Prentice Hall.
- Carr, David K., Kelvin J. Hard, and William J. Trahant. 1996. *Managing the change process: A field book for change agents, team leaders, and reengineering managers*. McGraw-Hill.
- Catmull, Ed. 2008. How Pixar fosters collective creativity. *Harvard Business Review*, September, 65–72.
- Cichelli, Sharon. 2008. Globally distributed Scrum. *Girl Writes Code* blog entry, May 9. <http://www.invisible-city.com/sharon/2008/05/globally-distributed-scrum.html>.
- Cirillo, Francesco. 2007. The pomodoro technique. Доступно в виде PDF-файла по адресу http://www.pomodorotechnique.com/resources/cirillo/ThePomodoroTechnique_v1-3.pdf.
- Clark, Kim B., and Steven C. Wheelwright. 1992. *Managing new product and process development: Text and cases*. The Free Press.
- Cockburn, Alistair. 2000. Balancing lightness with sufficiency. *Cutter IT Journal*, November.
- Cockburn, Alistair. 2006. *Agile software development: The cooperative game*. 2nd ed. Addison-Wesley Professional.
- Cockburn, Alistair. 2008. Using both incremental and iterative development. *Crosstalk*, May, 27–30.
- Cohn, Mike. 2004. *User stories applied: For agile software development*. Addison-Wesley Professional.
- Cohn, Mike. 2005. *Agile estimating and planning*. Addison-Wesley Professional.
- Conner, Daryl R. 1993. *Managing at the speed of change: How resilient managers succeed and prosper where others fail*. Random House.
- Conway, Melvin E. 1968. How do committees invent? Первоначально опубликовано в журнале *Datamation*, April 1968. В настоящее время располагается на сайте автора по адресу <http://www.melconway.com/research/committees.html>.
- Cooper, Robert G. 2001. *Winning at new products: Accelerating the process from idea to launch*. 3rd ed. Basic Books.
- Coyne, Kevin P., Patricia Gorman Clifford, and Renée Dye. 2007. Breakthrough thinking from inside the box. *Harvard Business Review*, December, 71–78.
- Creasey, Tim, and Jeff Hiatt, eds. 2007. *Best practices in change management*. Prosci.
- Crispin, Lisa, and Janet Gregory. 2009. *Agile Testing: A practical guide for testers and agile teams*. Addison-Wesley Professional.
- Crosby, Philip. 1979. *Quality is free: The art of making quality certain*. McGraw-Hill.
- Cunningham, Ward. 1992. The WyCash portfolio management system. В сборнике *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, 29–30. ACM. Имеется также на сайте по адресу <http://c2.com/doc/oops1a92.html>.
- Davies, Rachel, and Liz Sedley. 2009. *Agile coaching*. The Pragmatic Bookshelf.
- Deemer, Pete, Gabrielle Benefield, Craig Larman, and Bas Vodde. 2008. *The Scrum primer*. Scrum Training Institute.
- DeGrace, Peter, and Leslie Hulet Stahl. 1990. *Wicked problems, righteous solutions: April catalogue of modern software engineering paradigms*. Prentice Hall.

- DeMarco, Tom, Peter Hruschka, Tim Lister, Suzanne Robertson, James Roberts, and Steve McMenamin. 2008. *Adrenaline junkies and template zombies: Understanding patterns of project behavior*. Dorset House.
- DeMarco, Tom, and Timothy Lister. 1999. *Peopleware: Productive projects and teams*. 2nd ed. Dorset House.
- Deming, W. Edwards. 2000. *Out of the crisis*. MIT Press.
- de Pillis, Emmeline, and Kimberly Furumo. 2007. Counting the cost of virtual teams. *Communications of the ACM*, December, 93–95.
- Derby, Esther. 2006. A manager's guide to supporting organizational change. *Crosstalk*, January, 17–19.
- Derby, Esther, and Diana Larsen. 2006. *Agile retrospectives: Making good teams great*. Pragmatic Bookshelf.
- Deutschman, Alan. 2007. Inside the mind of Jeff Bezos. *Fast Company*, December 19. http://www.fastcompany.com/magazine/85/bezos_1.html.
- Dinwiddie, George. 2007. Common areas at the heart. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 207–211. IEEE Computer Society.
- Drummond, Brian Scott, and John Francis "JF" Unson. 2008. Yahoo! Distributed agile: Notes from the world over. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 315–321. IEEE Computer Society.
- Duarte, Deborah L., and Nancy Tennant Snyder. 2006. *Mastering virtual teams: Strategies, tools, and techniques that succeed*. 3rd ed. Jossey-Bass.
- Duck, Jeanie Daniel. 1993. Managing change: The art of balancing. *Harvard Business Review*, November-December, 109–119.
- Duvall, Paul, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: Improving software quality and reducing risk*. Addison-Wesley Professional.
- Dybå, Tore, Erik Arisholm, Dag I. K. Sjøberg, Jo Erskine Hannay, and Forrest Shull. 2007. Are two heads better than one? On the effectiveness of pair programming. *IEEE Software*, June, 12–15.
- Edmondson, Amy, Richard Bohmer, and Gary Pisano. 2001. Speeding up team learning. *Harvard Business Review*, October, 125–132.
- Elssamadisy, Amr. 2007. *Patterns of agile practice adoption: The technical cluster*. C4Media.
- Emery, Dale H. 2001. Resistance as a resource. *Cutter IT Journal*, October.
- Eoyang, Glenda Holladay. 2001. Conditions for self-organizing in human systems. PhD diss., The Union Institute and University.
- Feathers, Michael. 2004. *Working effectively with legacy code*. Prentice Hall PTR.
- Fecarotta, Joseph. 2008. MyBoeingFleet and agile software development. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 135–139. IEEE Computer Society.
- Feynman, Richard P. 1997. *Surely you're joking, Mr. Feynman! Adventures of a curious character*. W. W. Norton & Co.
- Fisher, Kimball. 1999. *Leading self-directed work teams*. McGraw-Hill.
- Florida, Richard, and James Goodnight. 2005. Managing for creativity. *Harvard Business Review*, July, 125–131.

- Fowler, Martin. 1999. *Refactoring: Improving the design of existing code*. При участии Kent Beck, John Brant, William Opdyke, и Don Roberts. Addison-Wesley Professional.
- Fowler, Martin. 2006. Using an agile software process with offshore development. Персональный сайт Мартина Фаулера, 18 июля. <http://martinfowler.com/articles/agileOffshore.html>.
- Fry, Chris, and Steve Greene. 2007. Large-scale agile transformation in an on-demand world. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 136–142. IEEE Computer Society.
- Gabardini, Juan. 2008. E-mail to Scrum Development mailing list, February 23. <http://groups.yahoo.com/group/scrumdevelopment/message/25071>.
- Gates, Bill. 1995. E-mail to Microsoft executive staff and his direct reports, May 26. Downloaded from the U.S. Department of Justice online case files. <http://www.usdoj.gov/atr/cases/exhibits/20.pdf>.
- George, Boby, and Laurie Williams. 2003. An initial investigation of test-driven development in industry. В сборнике *SAC'03: Proceedings of the 2003 ACM symposium on applied computing*, 1135–1139. ACM.
- Gilb, Tom. 1988. *Principles of software engineering management*. Addison-Wesley Professional.
- Gilb, Tom. 2005. *Competitive Engineering: A handbook for systems engineering, requirements engineering, and software engineering using planguage*. Butterworth-Heinemann.
- Gladwell, Malcolm. 2002. *The tipping point: How little things can make a big difference*. Back Bay Books.
- Glazer, Hillel, Jeff Dalton, David Anderson, Mike Konrad, and Sandy Shrum. 2008. *CMMI or agile: Why not embrace both!* Software Engineering Institute at Carnegie Mellon, November. <http://www.sei.cmu.edu/pub/documents/08.reports/08tn003.pdf>.
- Goldberg, Adele, and Kenneth S. Rubin. 1995. *Succeeding with objects: Decision frameworks for project management*. Addison-Wesley Professional.
- Goldstein, Jeffrey. 1994. *The unshackled organization: Facing the challenge of unpredictability through spontaneous reorganization*. Productivity Press.
- Gonzales, Victor M., and Gloria Mark. 2004. Constant, constant, multi-tasking craziness: Managing multiple working spheres. В сборнике *Proceedings of the CHI 2004 Connect Conference*, 113–120. ACM.
- Gratton, Lynda. 2007. *Hot spots: Why some teams, workplaces, and organizations buzz with energy — and others don't*. Berrett-Koehler Publishers.
- Gratton, Lynda, Andreas Voigt, and Tamara J. Erickson. 2007. Bridging faultlines in diverse teams. *MIT Sloan Management Review*, Summer, 22–29.
- Greene, Steve. 2007. Wall posting on the Facebook page of Adaptive Development Methodology (ADM), October 27. <http://www.facebook.com/wall.php?id=4791857957>.
- Greene, Steve. 2008. Unleashing the fossa: Scaling agile in an ambitious culture. Session presented at Agile Leadership Summit, Orlando. <http://www.slideshare.net/sgreen/unleashing-the-fossa-scaling-agile-in-an-ambitious-culture-presentation>.
- Greene, Steve, and Chris Fry. 2008. Year of living dangerously: How Salesforce.com delivered extraordinary results through a “big bang” enterprise agile revolution. Session presented at Scrum Gathering, Stockholm. <http://www.slideshare.net/sgreen/scrum-gathering-2008-stockholm-salesforcecom-presentation>.

- Griskevicius, V., R. B. Cialdini, and N. J. Goldstein. 2008. Applying (and resisting) peer influence. *MIT Sloan Management Review*, Winter, 84–88.
- Grossman, Lev. 2005. How Apple does it. *Time*, October 24, 66–70.
- Hackman, J. Richard. 2002. *Leading Teams: Setting the stage for great performances*. Harvard Business School Press.
- Hackman, J. Richard, and Diane Coutu. 2009. Why teams don't work. *Harvard Business Review*, May, 98–105.
- Hiatt, Jeffrey. 2006. *ADKAR: A model for change in business, government and our community*. Prosci Research.
- Highsmith, Jim. 2002. *Agile software development ecosystems*. Addison-Wesley.
- Highsmith, Jim. 2005. Managing change: Three readiness tests. *E-Mail Advisor*, July 14. Cutter Consortium.
- Highsmith, Jim. 2009. *Agile project management: Creating innovative products*. 2nd ed. Addison-Wesley Professional.
- Hodgetts, Paul. 2004. Refactoring the development process: Experiences with the incremental adoption of agile practices. В сборнике *Proceedings of the Agile Development Conference*, 106–113. IEEE Computer Society.
- Hofstede, Geert, and Gert-Jan Hofstede. 2005. *Cultures and organizations: Software of the mind*. 2nd ed. McGraw-Hill.
- Hogan, Ben. 2006. Lessons learned from an extremely distributed project. В сборнике *Proceedings of the Agile 2006 conference*, ed. Joseph Chao, Mike Cohn, Frank Maurer, Helen Sharp, and James Shore, 321–326. IEEE Computer Society.
- Honious, Jeff, and Jonathan Clark. 2006. Something to believe in. В сборнике *Proceedings of the Agile 2006 Conference*, ed. Joseph Chao, Mike Cohn, Frank Maurer, Helen Sharp, and James Shore, 203–212. IEEE Computer Society.
- Hubbard, Douglas W. 2007. *How to measure anything: Finding the value of "intangibles" in business*. Wiley.
- Iacovou, Charalambos L., and Robbie Nakatsu. 2008. A risk profile of offshore-outsourced development projects. *Communications of the ACM*, June, 89–94.
- James, Michael. 2007. A ScrumMaster's checklist, August 13. Блог Майкла Джеймса находится на веб-сайте Danube по адресу http://danube.com/blog/michaeljames/a_scrummasters_checklist.
- Jeffries, Ron. 2004a. Big visible charts. *XP*, October 20. <http://www.xprogramming.com/xpmag/BigVisibleCharts.htm>.
- Jeffries, Ron. 2004b. *Extreme programming adventures in C#*. Microsoft Press.
- Johnston, Andrew. 2009. The role of agile architect, June 20. Материал с сайта Agile Architect <http://www.agilearchitect.org/agile/role.htm>.
- Jones, Do-While. 1990. The breakfast food cooker. http://www.ridgecrest.ca.us/~do_while/toaster.htm.
- Kaplan, Robert S., and David P. Norton. 1992. The balanced scorecard: Measures that drive performance. *Harvard Business Review*, January–February, 71–79.
- Karten, Naomi. 1994. *Managing expectations*. Dorset House.
- Katzenbach, Jon. R. 1997. *Real change leaders: How you can create growth and high performance at your company*. Three Rivers Press.

- Katzenbach, Jon R., and Douglas K. Smith. 1993. *The wisdom of teams: Creating the high-performance organization*. Collins Business.
- Keith, Clinton. 2006. Agile methodology in game development: Year 3. Session presented at Game Developers Conference, San Jose.
- Kelly, James, and Scott Nadler. 2007. Leading from below. *MIT Sloan Management Review*, March 3. <http://sloanreview.mit.edu/business-insight/articles/2007/1/4917/leading-from-below>.
- Kerievsky, Joshua. 2005. Industrial XP: Making XP work in large organizations. *Cutter Consortium Agile Project Management Executive Report* 6 (2).
- Koskela, Lasse. 2007. *Test driven: TDD and acceptance TDD for Java developers*. Manning.
- Kotter, John P. 1995. Leading change: Why transformation efforts fail. *Harvard Business Review*, March-April, 59–67.
- Kotter, John P. 1996. *Leading change*. Harvard Business School Press.
- Krebs, William, and Per Kroll. 2008. Using evaluation frameworks for quick reflections. *Agile Journal*, February 9. <http://www.agilejournal.com/articles/columns/column-articles/750-using-evaluation-frameworks-for-quick-reflections>.
- Krug, Steve. 2005. *Don't make me think: A common sense approach to web usability*. 2nd ed. New Riders Press.
- LaFasto, Frank M.J., and Carl E. Larson. 2001. *When teams work best: 6,000 team members and leaders tell what it takes to succeed*. Sage Publications, Inc.
- Larman, Craig, and Victor R. Basili. 2003. Iterative and incremental development: A brief history. *IEEE Computer*, June, 47–56.
- Larman, Craig, and Bas Vodde. 2009. *Scaling lean & agile development: Thinking and organizational tools for large-scale Scrum*. Addison-Wesley Professional.
- Larson, Carl E., and Frank M. J. LaFasto. 1989. *Teamwork: What must go right/what can go wrong*. SAGE Publications.
- Lawrence, Paul R. 1969. How to deal with resistance to change. *Harvard Business Review*, January–February, 4–11.
- Leffingwell, Dean. 2007. *Scaling software agility: Best practices for large enterprises*. Addison-Wesley Professional.
- Liker, Jeffrey K. 2003. *The Toyota way*. MaGraw-Hill.
- Little, Todd. 2005. Context-adaptive agility: Managing complexity and uncertainty. *IEEE Software*, May–June, 28–35.
- Luecke, Richard. 2003. *Managing change and transition*. Harvard Business School Press.
- MacDonald, John D. 1968. *The girl in the plain brown wrapper*. Fawcett.
- Machiavelli, Nicollò. 2005. *The prince*. trans. Peter Bondanella. Oxford University Press.
- Mah, Michael. 2008. How agile projects measure up, and what this means to you. *Cutter Consortium Agile Product & Project Management Executive Report* 9 (9).
- Mair, Steven. 2002. A balanced scorecard for a small software group. *IEEE Software*, November/December, 21–27.
- Mangurian, Glenn, and Keith Lockhart. 2006. Responsibility junkie: Conductor Keith Lockhart on tradition and leadership. *Harvard Business Review*, October.

- Mann, Chris, and Frank Maurer. 2005. A case study on the impact of Scrum on overtime and customer satisfaction. В сборнике *Proceedings of the Agile Development Conference*, 70–79. IEEE Computer Society.
- Manns, Mary Lynn, and Linda Rising. 2004. *Fearless change: Patterns for introducing new ideas*. Addison-Wesley.
- Marick, Brian. 2007. *Everyday scripting with Ruby: For teams, testers, and you*. Pragmatic Bookshelf.
- Marsh, Stephen, and Stelios Pantazopoulos. 2008. Automated functional testing on the TransCanada Alberta gas accounting replacement project. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 239–244. IEEE Computer Society.
- Martin, Angela, Robert Biddle, and James Noble. 2004. The XP customer role in practice: Three studies. В сборнике *Proceedings of the Agile Development Conference*, 42–54. IEEE Computer Society.
- Martin, Robert C. 2008. *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.
- McCarthy, Jim. 2004. Twenty-one rules of thumb for shipping great software on time. Опубликовано как часть блога Дэвида Гриствуда по адресу http://blogs.msdn.com/David_Gristwood/archive/2004/06/24/164849.aspx.
- McCarthy, Jim, and Michele McCarthy. 2006. *Dynamics of software development*. Microsoft Press.
- McFarland, Keith R. 2008. Should you build strategy like you build software? *MIT Sloan Management Review*, Spring, 69–74.
- McKinsey & Company. 2008. Creating organizational transformations: McKinsey global survey results. *McKinsey Quarterly*, August. http://www.mckinseyquarterly.com/Creating_organizational_transformations_McKinsey_Global_Survey_results_2195.
- McMichael, Bill, and Marc Lombardi. 2007. ISO 9001 and agile development. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 262–265. IEEE Computer Society.
- Mediratta, Bharat. 2007. The Google way: Give engineers room. As told to Julie Bick. *The New York Times*, October 21. <http://www.nytimes.com/2007/10/21/jobs/21pre.html>.
- Mello, Antonio S., and Martin E. Ruckes. 2006. Team composition. *The Journal of Business* 79 (3): 1019–1039.
- Meszaros, Gerard. 2007. *xUnit test patterns: Refactoring test code*. Addison-Wesley.
- Miller, Ade. 2008. *Distributed agile development at Microsoft patterns & practices*. Microsoft. Доступно на сайте издательства по адресу <http://www.pnpguidance.net/Post/Distributed-AgileDevelopmentMicrosoftPatternsPractices.aspx>.
- Miller, Lynn. 2005. Case study of customer input for a successful product. В сборнике *Proceedings of the Agile Development Conference*, 225–234. IEEE Computer Society.
- Mintzberg, Henry. 2009. Rebuilding companies as communities. *Harvard Business Review*, July-August, 140–143.
- Moløkken-Østvold, Kjetil, and Magne Jørgensen. 2005. A comparison of software project overruns: Flexible versus sequential development methods. *IEEE Transactions on Software Engineering*, September, 754–766.
- Moore, Pete. 2005. *E=mc²: The great ideas that shaped our world*. Friedman.

- Moore, Richard, Kelly Reff, James Graham, and Brian Hackerson. 2007. Scrum at a Fortune 500 manufacturing company. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 175–180. IEEE Computer Society.
- Mugridge, Rick, and Ward Cunningham. 2005. *Fit for developing software: Framework for integrated tests*. Prentice Hall.
- Nicholson, Nigel. 2003. How to motivate your problem people. *Harvard Business Review*, January, 56–65.
- Nickols, Fred. 1997. Don't redesign your company's performance appraisal system, scrap it! *Corporate University Review*, May-June.
- Nielsen, Jakob. 2008. Agile development projects and usability. Alertbox, the author's online column, November 17. <http://www.useit.com/alertbox/agile-methods.html>.
- Nonaka, Ikujiro, and Hirotaka Takeuchi. 1995. *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. Oxford University Press.
- Ohno, Taiichi. 1982. *Workplace management*. trans. Jon Miller. Gemba Press. Цитируется у Poppendieck 2007.
- Olson, Edwin E., and Glenda H. Eoyang. 2001. *Facilitating organization change: Lessons from complexity science*. Pfeiffer.
- Paulk, Mark. 2001. Extreme programming from a CMM perspective. *IEEE Software*, November, 19–26.
- Pichler, Roman. Forthcoming. *Agile product management with Scrum: Creating products that customers love*. Addison-Wesley Professional.
- Poppendieck, Mary. 2007. E-mail to Lean Development mailing list, October 6. <http://tech.groups.yahoo.com/group/leandevlopment/message/2111>.
- Poppendieck, Mary, and Tom Poppendieck. 2006. *Implementing lean software development: From concept to cash*. Addison-Wesley Professional.
- Porter, Joshua. 2006. The freedom of fast iterations: How Netflix designs a winning web site. *User Interface Engineering*, November 14. http://www.uie.com/articles/fast_iterations/.
- Putnam, Doug. Team size can be the key to a successful project. Статья в QSM's Process Improvement Series. http://www.qsm.com/process_01.html.
- Ramasubbu, Narayan, and Rajesh Krishna Balan. 2007. Globally distributed software development project performance: An empirical analysis. В сборнике *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 125–134. ACM.
- Ramingwong, Sakgasit, and A. S. M. Sajeev. 2007. Offshore outsourcing: The risk of keeping mum. *Communications of the ACM*, August, 101–103.
- Rayhan, Syed H., and Nimat Haque. 2008. Incremental adoption of Scrum for successful delivery of an IT project in a remote setup. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 351–355. IEEE Computer Society.
- Reale, Richard C. 2005. *Making change stick: Twelve principles for transforming organizations*. Positive Impact Associates, Inc.
- Rico, David F. 2008. What is the ROI of agile vs. traditional methods? An analysis of extreme programming, test-driven development, pair programming, and Scrum (using real

options). Электронная таблица, которую можно загрузить с персонального сайта Дэвида Рико по адресу <http://davidfrico.com/agile-benefits.xls>.

Robarts, Jane M. 2008. Practical considerations for distributed agile projects. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 327–332. IEEE Computer Society.

Robbins, Stephen P. 2005. *Essentials of organizational behaviour*. Prentice Hall.

Rossi, Ernest Lawrence. 2002. The 20-minute ultradian healing response: An interview with Ernest Lawrence Rossi. Размещено на сайте автора по адресу <http://ernestrossi.com/interviews/ultradia.htm>.

Sanchez, Julio Cesar, Laurie Williams, and E. Michael Maximilien. 2007. On the sustained use of a test-driven development practice at IBM. 2007. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 5–14. IEEE Computer Society.

Schatz, Bob, and Ibrahim Abdelshafi. 2005. Primavera gets agile: A successful transition to agile development. *IEEE Software*, May/June, 36–42.

Schatz, Bob. 2006. The agile marathon. В сборнике *Proceedings of the Agile 2006 Conference*, ed. Joseph Chao, Mike Cohn, Frank Maurer, Helen Sharp, and James Shore, 139–146. IEEE Computer Society.

Schubring, Lori. 2006. Through the looking glass: Our long day's journey into agile. *Agile Development*, Spring, 26–28. http://www.agilealliance.org/agile_magazine.

Schwaber, Ken. 2004. *Agile project management with Scrum*. Microsoft Press.

Schwaber, Ken. 2006. The canary in the coal mine. Recorded video of session at Agile 2006 Conference, 1 hour, 9 min., 14 sec., embedded on InfoQ website, November 13. <http://www.infoq.com/presentations/agile-quality-canary-coalmine>.

Schwaber, Ken. 2007. *The enterprise and Scrum*. Microsoft Press.

Schwaber, Ken. 2009. *Scrum guide*, March. Размещено в виде PDF-файла на сайте Scrum Alliance по адресу <http://www.scrumalliance.org/resources/598>.

Schwaber, Ken, and Mike Beedle. 2001. *Agile software development with Scrum*. Prentice Hall.

Schwartz, Tont, and Catherine McCarthy. 2007. Manage your energy, not your time. *Harvard Business Review*, October, 63–73.

Seffernick, Thomas R. 2007. Enabling agile in a large organization: Our journey down the yellow brick road. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 200–206. IEEE Computer Society.

Shaw, D. M. 1960. Size of share in task and motivation in work groups. *Sociometry* 23: 203–208.

Sliger, Michele. 2006. Bridging the gap: Agile projects in the waterfall enterprise. *Better Software*, July/August, 26–31.

Sliger, Michele, and Stacia Broderick. 2008. *The software project manager's bridge to agility*. Addison-Wesley Professional.

Sosa, Manuel E., Steven D. Eppinger, and Craig M. Rowles. 2007. Are your engineers talking to one another when they should? *Harvard Business Review*, January, 133–142.

Spann, David. 2006. Agile manager behaviors: What to Look for and develop. *Cutter Consortium Executive Report*, September.

- Stangor, Charles. 2004. *Social groups in action and interaction*. Psychology Press.
- Steiner, I. D. 1972. *Group process and productivity*. Academic Press, Inc.
- Striebeck, Mark. 2006. Ssh! We are adding a process... В сборнике *Proceedings of the Agile 2006 Conference*, ed. Joseph Chao, Mike Cohn, Frank Maurer, Helen Sharp, and James Shore, 185–193. IEEE Computer Society.
- Striebeck, Mark. 2007. Agile adoption at Google: Potential and challenges of a true bottom-up organization. Session presented at Agile 2007 conference, Washington, DC.
- Subramaniam, Venkat, and Andy Hunt. 2006. *Practices of an agile developer: Working in the real world*. Pragmatic Bookshelf.
- Summers, Mark. 2008. Insights into an agile adventure with offshore partners. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 333–339. IEEE Computer Society.
- Sutherland, Jeff, Carsten Ruseng Jakobsen, and Kent Johnson. 2007. Scrum and CMMI level 5: The magic potion for code warriors. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 272–278. IEEE Computer Society.
- Sutherland, Jeff, Guido Schoonheim, Eelco Rustenburg, and Mauritz Rijk. 2008. Fully distributed Scrum: The secret sauce for hyperproductive offshore development teams. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 339–344. IEEE Computer Society.
- Sutherland, Jeff, Anton Viktorov, and Jack Blount. 2006. Adaptive engineering of large software projects with distributed/outsourced teams. В сборнике *Proceedings of the Sixth International Conference on Complex Systems*, ed. Ali Minai, Dan Braha, and Yaneer Bar-Yam. New England Complex Systems Institute.
- Sutherland, Jeff, Anton Viktorov, Jack Blount, and Nikolai Puntikov. 2007. Distributed Scrum: Agile project management with outsourced development teams. В сборнике *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, 274a. IEEE Computer Society.
- Sy, Desirée. 2007. Adapting usability investigations for agile user-centered design. *Journal of Usability Studies* 2 (3): 112–132.
- Tabaka, Jean. 2006. *Collaboration explained: Facilitation skills for software project leaders*. Addison-Wesley Professional.
- Tabaka, Jean. 2007. Twelve ways agile adoptions fail. *Better Software*, November, 7.
- Takeuchi, Hirotaka, and Ikujiro Nonaka. 1986. The new new product development game. *Harvard Business Review*, January, 137–146.
- Tengshe, Ash, and Scott Noble. 2007. Establishing the agile PMO: Managing variability across projects and portfolios. В сборнике *Proceedings of the Agile 2007 Conference*, ed. Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, 188–193. IEEE Computer Society.
- Thaler, Richard H., and Cass R. Sunstein. 2009. *Nudge: Improving decisions about health, wealth, and happiness*. Updated ed. Penguin.
- Therrien, Elaine. 2008. Overcoming the challenges of building a distributed agile organization. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 368–372. IEEE Computer Society.

- Thomas, Dave. 2005. Agile programming: Design to accommodate change. *IEEE Software*, May/June, 14–16.
- Toffler, Alvin. 1970. *Future shock*. Random House.
- Tubbs, Stewart L. 2004. *A systems approach to small group interaction*. 8th ed. McGraw-Hill.
- Turner, Richard, and Apurva Jain. 2002. Agile meets CMMI: Culture clash or common cause? В сборнике *Extreme Programming and Agile Methods: XP/Agile Universe 2002*, ed. D. Wells and L. A. Williams, 153–165. Springer.
- Unson, J. F. 2008. E-mail to Scrum Development mailing list, May 26. <http://groups.yahoo.com/group/scrumdevelopment/message/29481>.
- Vax, Michael, and Stephen Michaud. 2008. Distributed agile: Growing a practice together. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 310–314. IEEE Computer Society.
- Venners, Bill. 2003. Tracer bullets and prototypes: A conversation with Andy Hunt and Dave Thomas, part VIII. *Artima Developer*, April 21. <http://www.artima.com/intv/tracer.html>.
- VersionOne. 2008. The state of agile development: Third annual survey. Доступно в виде PDF-файла в библиотеке Library of White Papers на сайте VersionOne по адресу http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf.
- Wake, William C. 2003. *Refactoring workbook*. Addison-Wesley Professional.
- Ward, Allen C. 2007. *Lean product and process development*. Lean Enterprise Institute.
- Wenger, Etienne, Rechard McDermott, and William M. Snyder. 2002. *Cultivating communities of practice*. Harvard Business School Press.
- Williams, Laurie, Lucas Layman, and William Krebs. 2004. Extreme programming evaluation framework for object-oriented languages, version 1.4. North Carolina State University Department of Computer Science, TR-2004-18.
- Williams, Laurie, Anuja Shukla, and Annie I. Anton. 2004. An initial exploration of the relationship between pair programming and Brook's law. В сборнике *Proceedings of the Agile Development Conference*, 11–20. IEEE Computer Society.
- Williams, Wes, and Mike Stout. 2008. Colossal, scattered, and chaotic: Planning with a large distributed team. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 356–361. IEEE Computer Society.
- Woodward, E. V., R. Bowers, V. Thio, K. Johnson, M. Srihari, and C. J. Bracht. Forthcoming. Agile methods for software practice transformation. *IBM Journal of Research and Development* 54 (2).
- Wright, Graham. 2003. Achieving ISO 9001 certification for an XP company. В сборнике *Extreme Programming and Agile Methods: XP/Agile Universe 2003*, ed. F. Maurer and D. Wells, 43–50. Springer.
- Yegge, Steve. 2006. Good agile, bad agile. *Stevey's Blog Rants*, September 27. http://steve-yegge.blogspot.com/2006/09/good-agile-bad-agile_27.html.
- Young, Cynick, and Hiroki Terashima. 2008. How did we adapt agile processes to our distributed development? Overcoming the challenges of building a distributed agile organization. В сборнике *Proceedings of the Agile 2008 Conference*, ed. Grigori Melnik, Philippe Kruchten, and Mary Poppendieck, 304–309. IEEE Computer Society.

Предметный указатель

A

ADAPT, 58
Agile\EF, 535
Agile Manifesto, 251; 288; 309; 443
ATDD, 401

C

CMMI, 497

D

DEEP, 326

E

ETC, 107

I

IC, 114
ISO 9001, 494

P

PMO, 504; 521

S

Scrum-мастер, 167; 195
SEPG, 122

T

TDD, 213

U

UED, 206; 347

A

Автоматизация тестирования, 204
Автоматизация тестов, 398
Администратор базы данных, 202
Аналитик, 190
Архитектор, 196
Аттестация, 506

B

Визиты знакомства, 459
Владелец продукта, 176
команда, 181
Внутреннее наставничество, 94; 95
Выбор времени, 131

Г

Группа по интересам, 114
Группа технического обеспечения, 503; 512
Групплет, 116

Д

Документирование, 306

Ж

Журнал запросов на выполнение работ, 305;
416
DEEP, 326
Журнал
проблем, 432
совершенствования, 105

З

Заблуждение, 149
Запаздывающий индикатор, 543

И

Измерение, 531
История пользователя, 191; 309
эпическая, 314

К

Кадры, 252
Карьера, 510
Качество, 52
Коллектив, 32
Коллективная ответственность, 266
Команда, 32; 235
компонентная, 244
обучение, 274
размер, 238
структура, 237
функциональная, 244
Консерваторы, 147
Контейнер, 289
Культурные различия, 450

Л

Лидер мнения, 68

М

Маркетинговый отдел, 78
Метод
"разделить и засеять", 92
"нарастить и разделить", 93
Многозадачность, 254
корпоративная, 258
Модель ADAPT, 58
желание, 63
осознание, 59

продвижение, 72

распространение, 76

способность, 68

Модель СДЕ, 292

Мягкий контроль, 252

Н

Недостаточная информированность, 150

Необходимость перемен, 60

Новаторы, 147

О

Обучение, 274

Одновременный переход, 86

Ожидания, 135

Окружение, 297

Опережающий индикатор, 543

Организационная гравитация, 76; 503; 547

источники, 77

Отдел

кадров, 77; 503; 504

управления проектами, 504; 521

Открытое пространство, 425

Открытый переход, 89

П

Парное программирование, 221

Пилотный проект, 62; 128

Повышение заинтересованности, 50

Повышение качества, 52

Последовательная разработка, 486

Постепенный переход, 85

Потенциально готовый продукт, 332

Пояз безопасности, 140

Прагматики, 147

Приемлемый темп, 367

Приемочный тест, 401

Программист, 200

Проектирование, 224

на основе тестирования, 212

пользовательского интерфейса, 347

Проектировщик пользовательского
интерфейса, 206

Производительность, 47; 241

Р

Различие, 290

Разработчик, 32

Рассредоточенные коллективы, 445

Рефакторинг, 215

Руководитель проекта, 192

С

Самоорганизация, 288

Сбалансированная система показателей, 542

Сверхурочная работа, 367

Синхронизация спринтов, 433

Скрытый переход, 90

Совещание, 468

ежедневное разработчиков, 432; 475

Сообщества практиков, 438

Сообщество в поддержку

перехода к Scrum, 107

усовершенствований, 32; 114

Сопротивление, 146

консерваторы, 160

последователи, 162

саботажники, 158

скептики, 155

Сопротивление переменам, 72; 146

Сотрудничество, 282

Спонсор перехода, 109

Спонтанные требования, 312

Спринт, 329

синхронизация, 433

Т

Тестер, 202

Тестирование, 390

Технический долг, 405

Трансформирующий обмен, 290

Трудности перехода, 39

У

Увольнение, 509

Условие удовлетворенности, 320

Ф

Физическое пространство, 512

Финансовый отдел, 78

Фобия, 149

Формулировка ожиданий, 135

Функциональный менеджер, 197

Ч

Число Данбара, 419

Ш

Шок от будущего, 44

Э

Эволюция, 296

Эффект умолчания, 467

ГИБКОЕ ТЕСТИРОВАНИЕ

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ДЛЯ ТЕСТИРОВЩИКОВ ПО И ГИБКИХ КОМАНД

**Лайза Криспин
Джанет Грегори**



В настоящей книге дается исчерпывающее определение гибкого тестирования и показана роль тестировщиков в реальных гибких командах. Подробно рассматривается использование квадрантов гибкого тестирования для идентификации потребностей в тестировании, анализируются требования к тестировщикам и предлагаются советы по построению набора инструментальных средств, помогающего проводить тестирование наиболее эффективно. В книге описана итерация гибкой разработки программного обеспечения с точки зрения тестировщика, а также объясняются семь ключевых факторов успеха гибкого тестирования. Читатели узнают, как вовлечь тестировщиков в гибкую разработку, каким образом произвести переход от традиционной циклической к гибкой разработке, как обеспечить полное выполнение всех действий по тестированию в течение коротких итераций, а также каким образом организовать управление процессом разработки с помощью тестов.

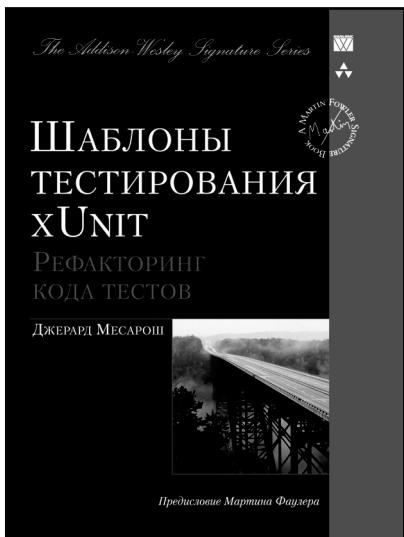
www.williamspublishing.com

ISBN 978-5-8459-1625-9

в продаже

ШАБЛОНЫ ТЕСТИРОВАНИЯ XUNIT РЕФАКТОРИНГ КОДА ТЕСТОВ

Джерард Месарош



www.williamspublishing.com

В данной книге показано, как применять принципы разработки программного обеспечения, в частности шаблоны проектирования, инкапсуляцию, исключение повторений и описательные имена, к написанию кода тестов. Книга состоит из трех частей. В первой части приводятся теоретические основы методов разработки тестов, описываются концепции шаблонов и "запахов" тестов (признаков существующей проблемы). Во второй и третьей частях книги приводится каталог шаблонов проектирования тестов, "запахов" и других средств обеспечения большей прозрачности кода тестов. Кроме этого, в третьей части книги сделана попытка обобщить и привести к единому знаменателю терминологию тестовых двойников и подставных объектов, а также рассмотрены некоторые принципы их применения при проектировании как тестов, так и самого программного обеспечения. Книга ориентирована на разработчиков программного обеспечения, практикующих гибкие процессы разработки.

ISBN 978-5-8459-1448-4

в продаже

РЕФАКТОРИНГ В C# И ASP.NET ДЛЯ ПРОФЕССИОНАЛОВ

Даниэль Арсеновски



www.dialektika.com

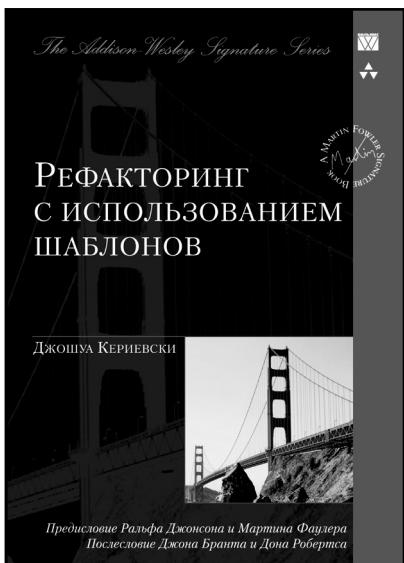
ISBN 978-5-8459-1618-1

В книге предлагается практический подход к проведению рефакторинга в C# и ASP.NET, который позволяет применять приемы рефакторинга при управлении и модификации существующего кода. Кроме того, вы научитесь строить прототип приложения с нуля и узнаете, как с помощью рефакторинга преобразовать прототип в хорошо спроектированное приложение уровня предприятия. Благодаря пошаговым инструкциям, вы легко разберетесь с разнообразными проблемами, связанными с кодом, и трансформациями рефакторинга. Многие из этих трансформаций разработаны на основе реальных сценариев и являются результатом ключевых бизнес-решений. Вдобавок вы найдете формальные определения приемов рефакторинга, на которые можно будет ссылаться во время работы. В этой книге описаны приемы рефакторинга, которые позволяют существенно повысить вашу производительность.

в продаже

РЕФАКТОРИНГ С ИСПОЛЬЗОВАНИЕМ ШАБЛОНОВ

Джошуа Кериевски



www.williamspublishing.com

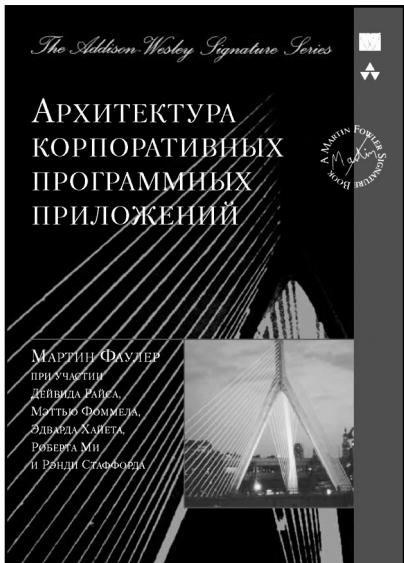
ISBN 5-8459-1087-0

Данная книга представляет собой результат многолетнего опыта профессионального программиста по применению шаблонов проектирования. Авторский подход к проектированию состоит в том, что следует избегать как недостаточного, так и избыточного проектирования, постоянно анализируя готовый работоспособный код и реорганизуя его только в том случае, когда это приведет к повышению его эффективности, упрощению его понимания и сопровождения. Автор на основании как собственного, так и чужого опыта детально рассматривает различные признаки кода, требующего рефакторинга, описывает, какой именно рефакторинг наилучшим образом подходит для той или иной ситуации, и описывает его механику, подробно разбирая ее на конкретных примерах из реальных задач. Книга может рассматриваться и как учебник по рефакторингу для программиста среднего уровня, и как справочное пособие для профессионала.

в продаже

АРХИТЕКТУРА КОРПОРАТИВНЫХ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Мартин Фаулер



www.williamspublishing.com

ISBN 978-5-8459-0579-6

Книга дает ответы на трудные вопросы, с которыми приходится сталкиваться всем разработчикам корпоративных систем. Автор, известный специалист в области объектно-ориентированного программирования, заметил, что с развитием технологий базовые принципы проектирования и решения общих проблем остаются неизменными, и выделил более 40 наиболее употребительных подходов, оформив их в виде типовых решений. Результат перед вами — незаменимое руководство по архитектуре программных систем для любой корпоративной платформы. Это своеобразное учебное пособие поможет вам не только усвоить информацию, но и передать полученные знания окружающим значительно быстрее и эффективнее, чем это удавалось автору. Книга предназначена для программистов, проектировщиков и архитекторов, которые занимаются созданием корпоративных приложений и стремятся повысить качество принимаемых стратегических решений.

в продаже

ИНФРАСТРУКТУРА ПРОГРАММНЫХ ПРОЕКТОВ

СОГЛАШЕНИЯ, ИДИОМЫ И ШАБЛОНЫ ДЛЯ МНОГОКРАТНО
ИСПОЛЬЗУЕМЫХ БИБЛИОТЕК .NET

ВТОРОЕ ИЗДАНИЕ

*Кржиштоф Цвалина
Брэд Абрамс*



www.williamspublishing.com

Эта книга позволяет разработчикам освоить лучшие приемы разработки многократно используемых библиотек для Microsoft .NET Framework. Расширенное и обновленное для .NET 3.5, это новое издание фокусирует внимание на разработке тех частей, которые непосредственно определяют применение в программах библиотеки классов, особенно общедоступных API.

Эта книга облегчит работу любого .NET-разработчика, который разрабатывает код, предназначенный для других разработчиков. Она содержит комментарии к правилам, принадлежащие тридцати пяти выдающимся архитекторам и программистам .NET Framework, оживленные обсуждения причин, по которым были введены данные правила, а также примеры, показывающие, когда можно нарушить обсуждаемые правила.

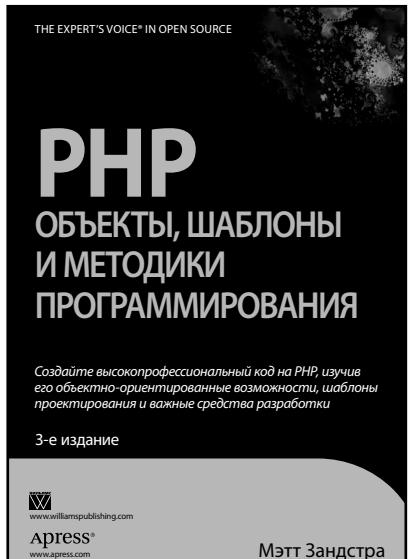
ISBN 978-5-8459-1692-1 в продаже

PHP

ОБЪЕКТЫ, ШАБЛОНЫ И МЕТОДИКИ ПРОГРАММИРОВАНИЯ

3-Е ИЗДАНИЕ

Мэтт Зандстра



www.williamspublishing.com

Создайте высокопрофессиональный код на PHP, изучив его объектно-ориентированные возможности, шаблоны проектирования и важные средства разработки

3-е издание

W
www.williamspublishing.com
Apress®
www.apress.com

Мэтт Зандстра

За последние десять лет PHP буквально охватила объектно-ориентированная революция, причем это относится как к самим средствам языка, так и к разработчикам, использующим эти средства, и к приложениям, которые они создают. Теперь основной акцент делается на объектах и объектно-ориентированном подходе к проектированию.

Книга начинается с обзора объектно-ориентированных возможностей PHP, в который включены важные темы, такие как определение класса, создание объектов, наследование, инкапсуляция методов и свойств. Вы изучите также и дополнительные темы: статические методы и свойства, абстрактные классы, обработка исключений, клонирование объектов, пространства имён, механизм замыканий и многое другое.

Следующая часть книги посвящена шаблонам проектирования, которые органически дополняют тему ООП и являются описанием элегантных решений распространенных проблем, возникающих при проектировании программного обеспечения. В ней описываются концепции шаблонов проектирования и показаны способы реализации нескольких важных шаблонов в приложениях на PHP.

ISBN 978-5-8459-1689-1 в продаже

ПРИМЕНЕНИЕ UML 2.0 И ШАБЛОНОВ ПРОЕКТИРОВАНИЯ, третье издание

Крэг Ларман



www.williamspublishing.com

В книге вы найдете новые сведения об итеративном и гибком моделировании, шаблонах проектирования GRASP и GoF, прецедентах, архитектурном анализе и многоуровневой архитектуре, а также рефакторинге, разработке на основе обязанностей и многих других вопросах. Весь материал рассматривается в контексте использования унифицированного процесса (UP) как легкого и гибкого подхода к разработке совместно с приемами из других итеративных методов, таких как XP и Scrum.

Данная книга будет прекрасным руководством для как для новичков, так и для специалистов, кто интересуется вопросами ООА/П, языком моделирования UML 2 и самыми современными эволюционными подходами к разработке программного обеспечения.

ISBN 978-5-8459-1185-8 в продаже

ШАБЛОНЫ РЕАЛИЗАЦИИ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ

Кент Бек



www.dialektika.com

Один из самых креативных и признанных лидеров в индустрии программного обеспечения Кент Бек собрал 77 шаблонов, предназначенных для выполнения ежедневных программистских задач и написания более читаемого кода. Эта новая коллекция шаблонов предназначена для реализации многих аспектов разработки, включая классы, состояние, поведение, методы, коллекции, инфраструктуры и т.д. Автор использует диаграммы, историю, примеры и эссе для того, чтобы увлечь читателя по ходу освещения шаблонов. Вы обнаружите проверенные решения для управления всем, от именования переменных до проверки исключений.

Эта книга предназначена для программистов всех уровней подготовки, особенно для тех, кто применяет в своей практике шаблоны проектирования и методы быстрой разработки. Книга также окажется неоценимым ресурсом для команд разработчиков, ищущих более эффективные методы совместной работы и построения более управляемого ПО.

ISBN 978-5-8459-1406-4 в продаже

ШАБЛОНЫ С++: СПРАВОЧНИК РАЗРАБОТЧИКА

**Дэвид Вандевурд,
Николай М. Джосаттис**



www.williamspublishing.com

в продаже

Несмотря на то, что шаблоны вошли в язык программирования C++ более десяти лет назад, они представляют собой активно развивающуюся часть C++, предоставляющую программисту новые возможности быстрой разработки эффективных и надежных программ и повторного использования кода. Будучи одной из наиболее мощных возможностей языка, шаблоны одновременно являются одной из самых запутанных, трудно понимаемых и зачастую неверно используемых частей C++. Данная книга, написанная в соавторстве теоретиком C++ и программистом-практиком с большим опытом, удачно сочетает строгость изложения и полноту освещения темы с вопросами практического использования шаблонов. В книге содержится масса разнообразного материала, относящегося к программированию с использованием шаблонов, включая такие вопросы, как новые идиомы программирования, связанные с шаблонами, метапрограммирование или возможные направления развития шаблонов в будущем. Кроме того, книга снабдит читателя материалом, который даст опытным программистам возможность преодолеть современные ограничения в этой области. Книга предполагает наличие у читателя достаточно глубоких знаний языка C++, поскольку в книгедается детальное описание конкретного средства языка программирования, но не основ самого языка. Тем не менее стиль изложения обеспечивает доступность материала как для квалифицированных специалистов, так и для программистов среднего уровня.