



ЖЕМЧУЖИНЫ РАЗРАБОТКИ

Чему мы научились за 50 лет создания ПО



КАРЛ ВИГЕРС

Предисловие Стив Макконнелл





Software Development Pearls

Lessons from Fifty Years of Software Experience

Karl Wiegers

◆ Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo



Жемчужины разработки

Чему мы научились
за 50 лет создания ПО

Карл Вигерс



Санкт-Петербург • Москва • Минск

2024

ББК 32.973.2-018
УДК 004.4
B41

Вигерс Карл

- B41 Жемчужины разработки. Чему мы научились за 50 лет создания ПО. — СПб.: Питер, 2024. — 368 с.: ил. — (Серия «Для профессионалов»).
ISBN 978-5-4461-1986-8

Совершенное программное обеспечение невозможно создать без изучения накопленного опыта.

Опыт — главный учитель, но медленный и нередко болезненный. Но зачем же нам повторять ошибки? Книга «Жемчужины разработки» поможет совершенствоваться быстрее и избежать многих проблем, обучаясь на опыте других людей, которые уже поднялись по кривой обучения. Карл Вигерс сформулировал 60 кратких практических уроков, которые подойдут для любых проектов, независимо от роли, отрасли, технологии или методологии.

Идеи и конкретные рекомендации охватывают шесть важнейших элементов успеха: требования, дизайн, управление проектами, культуру и командную работу, качество и совершенствование процессов. Для каждого из направлений Вигерс предлагает «первые шаги», позволяющие осмыслить собственный опыт, уроки с основными идеями, реальными примерами и действенными решениями и «следующие шаги» для внедрения опыта в вашем проекте, команде или организации. Эти знания нельзя получить в университете!

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018
УДК 004.4

Права на издание получены по соглашению с Pearson Education Inc. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-0137487776 англ.
ISBN 978-5-4461-1986-8

© 2022 Karl E. Wiegert
© Перевод на русский язык ООО «Прогресс книга», 2024
© Издание на русском языке, оформление ООО «Прогресс книга», 2024
© Серия «Для профессионалов», 2024

Краткое содержание

Отзывы о книге	18
Предисловие	21
Благодарности	23
Об авторе	25
От издательства	26
Глава 1. Обучение на собственном опыте	27
Глава 2. Требования	33
Глава 3. Проектирование	112
Глава 4. Управление проектами	150
Глава 5. Культура и командная работа	209
Глава 6. Качество	252
Глава 7. Совершенствование процессов	297
Глава 8. Что дальше?	341
Приложение. Краткий перечень уроков	349
Список использованных источников	353

Оглавление

Отзывы о книге	18
Предисловие.....	21
Благодарности	23
Об авторе	25
От издательства.....	26
Глава 1. Обучение на собственном опыте.....	27
Моя точка зрения.....	27
О книге.....	29
Примечание о терминологии	31
Ваши возможности.....	31
Глава 2. Требования.....	33
Введение.....	33
Множество типов требований	33
Подобласти разработки требований.....	35
Роль бизнес-аналитика.....	37
Требования — это фундамент.....	38
УРОК 1. Если вы неверно определили требования, то неважно, насколько хорошо вы выполните остальную часть работы	39
Правильные требования, но когда?	41
Правильные требования, но как?	41
УРОК 2. Основной результат разработки требований — общее введение и понимание.....	43

УРОК 3. Интересы всех заинтересованных сторон нигде не пересекаются так явственно, как в требованиях	46
Анализ заинтересованных сторон.....	47
Кто звонит?.....	50
Мы все на одной стороне.....	51
УРОК 4. В требованиях в первую очередь важны особенности использования, а затем — функциональность	51
Зачем нужна лишняя функциональность?	52
Ставим особенности использования на первое место.....	53
Проблема пользовательских историй	54
Правила использования	56
УРОК 5. Разработка требований — итеративный процесс	57
Постепенное уточнение деталей.....	57
Возникающие функциональные требования.....	58
Возникающие нефункциональные требования.....	60
УРОК 6. Agile-требования не отличаются от других	60
Роли и обязанности	61
Терминология.....	62
Детальность документации	62
Выбор времени для выполнения действий.....	63
Готовые формы.....	64
Когда расставлять приоритеты.....	65
Есть ли разница?.....	65
УРОК 7. Запись знаний обходится дешевле, чем повторное их обретение	66
Боязнь рутинны	67
Преимущества письменного общения	68
Разумный баланс	70
УРОК 8. Главное требование к разработке — наложенное и эффективное общение	71
Разные аудитории, разные потребности	73
Выбор методов представления	74
Мы можем поговорить?.....	76

УРОК 9. Качественность требований каждый определяет по-своему	77
Многообразие получателей требований	77
Контрольный список для оценки качественности требований	79
УРОК 10. Требования должны быть достаточно хорошими, чтобы разработка могла продолжаться	
с приемлемым уровнем риска	81
Уровень детализации	81
Достаточно — это сколько?	83
УРОК 11. Люди не просто так собирают требования	83
Сбор и выявление	83
Когда выявлять требования	85
Контекст выявления	86
Методы выявления	86
Закладка фундамента	89
УРОК 12. Выявление требований должно помочь разработчикам услышать голос клиента	89
Способы взаимодействий	90
Менеджер по продвижению продукта	91
Другие пути передачи требований	92
Преодоление разрыва	93
УРОК 13. Две распространенные практики выявления требований — телепатия и ясновидение. Но они не работают	94
Угадай это требование!	94
Старайтесь выражаться ясно	95
Телепатия не срабатывает	96
УРОК 14. Большая группа людей не способна организованно покинуть горящую комнату, не говоря уже о том, чтобы сформулировать какое-то требование	97
Концентрируйте внимание!	98
Координатор во спасение	99
Фокус, фокус, фокус	100
Не ограничивайтесь обсуждениями в группах	101
УРОК 15. Когда принимаете решение о добавлении функций, избегайте расстановки приоритетов по децибелам	102
Методы определения приоритетов	103

Критерии выбора приоритетов	104
Анализ превыше громкости	105
УРОК 16. Не задокументировав и не согласовав содержимое проекта, нельзя узнать, увеличивается ли его объем.....	106
Призрак разбухания проекта.....	106
Как документировать объем.....	107
Это входит в рамки проекта?.....	108
Расплывчатые требования = расплывчатый объем проекта	109
Глава 3. Проектирование.....	112
Введение.....	112
Различные аспекты проектирования	113
У вас хороший проект?.....	115
УРОК 17. Проектирование — итеративный процесс.....	118
Сила прототипов	119
Проверка концепции.....	120
Макеты	121
УРОК 18. Чем выше уровень абстракции, тем проще выполнять итерации	122
Отступая от деталей.....	124
Быстрые визуальные итерации.....	125
Итерации стали проще	127
УРОК 19. Разрабатывайте продукты так, чтобы их легко было использовать правильно и трудно — неправильно	128
Не давайте пользователю возможности ошибиться	130
Затрудните пользователю возможность ошибиться	131
Упростите исправление допущенной ошибки	131
Просто позвольте ошибкам случиться.....	131
УРОК 20. Невозможно оптимизировать все желаемые атрибуты качества	132
Измерения качества	133
Определение атрибутов качества	136
Проектирование для качества.....	136
Архитектура и атрибуты качества	138

УРОК 21. Проблемы легче предупредить, чем исправить	138
Технический долг и рефакторинг	139
Архитектурные недостатки	141
УРОК 22. Проблемы многих систем скрываются в интерфейсах	142
Технические проблемы с интерфейсами	143
Проверка входных данных	146
Проблемы с пользовательским интерфейсом	147
Сложности с интерфейсами	148
Глава 4. Управление проектами	150
Введение.....	150
Управление персоналом.....	151
Управление требованиями	151
Управление ожиданиями	151
Управление задачами	152
Управление обязательствами	152
Управление рисками	153
Управление коммуникациями	153
Управление изменениями.....	153
Управление ресурсами	154
Управление зависимостями	154
Управление контрактами.....	154
Управление поставщиками.....	155
Устранение препятствий.....	155
УРОК 23. При планировании работ нужно учитывать разногласия.....	157
Переключение между задачами и состояние потока	157
Эффективные часы.....	160
Другие источники разногласия в проекте	161
Последствия планирования	162
УРОК 24. Не давайте оценок наугад	163
Поспешные прогнозы	163
Страх неопределенности	165
УРОК 25. Айсберги всегда больше, чем кажутся	166
Резерв времени.....	167

Рискованные оценки.....	169
Контракты на айсбергах.....	171
Преимущества резервов времени.....	172
УРОК 26. Ваши переговорные позиции будут сильнее при наличии обосновывающих данных	172
Откуда вы взяли эту цифру?.....	173
Принципиальные переговоры	174
УРОК 27. Не записывая оценки и не сверяя их с тем, что произошло на самом деле, вы всегда будете строить догадки, а не оценивать.....	175
Несколько источников данных за прошедший период.....	176
Характеристики программного обеспечения	178
УРОК 28. Не меняйте оценку в зависимости от того, что хочет услышать получатель	179
Цели и оценки.....	180
Когда корректировать оценку	181
УРОК 29. Держитесь подальше от критического пути.....	182
Определение критического пути	182
Держитесь в стороне	183
УРОК 30. Задание либо полностью выполнено, либо не выполнено: частичное выполнение не засчитывается	185
Что означает «готово»?.....	185
Не бывает частичной готовности.....	187
Отслеживание по статусу требований	189
Готовность ведет к ценности	189
УРОК 31. Команде проекта нужна гибкость в отношении хотя бы одного из пяти измерений: масштаба, плана, бюджета, персонала и качества.....	190
Пять измерений проекта	190
Соглашения о приоритетах	192
Диаграмма гибкости	193
Практическое применение анализа пяти измерений.....	194
УРОК 32. Если вы не контролируете риски своего проекта, то они будут контролировать вас.....	195
Что такое управление рисками?	196

Выявление рисков использования программного обеспечения.....	196
Действия по управлению рисками	198
Всегда есть о чем беспокоиться	201
УРОК 33. Клиент не всегда прав	201
Быть «неправым»	202
Уважение точки зрения	205
УРОК 34. Мы слишком часто принимаем желаемое за действительное	205
Жизнь в стране фантазий	205
Иррациональное преувеличение	206
Игры, в которые играют люди.....	207
Глава 5. Культура и командная работа	209
Введение.....	209
Сохраняя веру	210
Культурная конгруэнтность	211
Кристаллизация культуры	212
Увеличение команды.....	214
УРОК 35. Передача знаний не ведет к проигрышу	216
Пожиратель знаний.....	217
Исправление невежества	217
Расширение масштабов передачи знаний.....	218
Здоровая информационная культура.....	220
УРОК 36. Как бы сильно на вас ни давили, не берите на себя обязательства, которые не сможете выполнить.....	221
Обещания, обещания.....	223
В жизни случается всякое.....	224
УРОК 37. Не ждите, что без обучения и освоения передовых практик продуктивность повысится как по волшебству.....	225
В чем проблема?	226
Некоторые возможные решения	226
Инструменты и обучение.....	228
Индивидуальные особенности разработчиков	229

УРОК 38. Люди много говорят о своих правах, но права подразумеваются ответственность	231
Некоторые права и обязанности клиентов	232
Некоторые права и обязанности разработчиков	232
Некоторые права и обязанности руководителя или спонсора проекта.....	233
Некоторые права и обязанности члена автономной команды	233
Опасения перед кризисом	234
УРОК 39. Даже небольшие физические расстояния препятствуют общению и совместной работе	234
Барьеры пространства и времени.....	235
Виртуальные команды: максимальное разделение	237
Дверь, дверь, королевство за дверь!.....	237
УРОК 40. Неформальные подходы, используемые небольшими сплоченными командами, плохо масштабируются.....	239
Процессы и инструменты.....	240
Необходимость специализации	241
Коммуникационные конфликты	242
УРОК 41. Не стоит недооценивать сложность изменения культуры организации по мере перехода к новым методам работы.....	243
Ценности, модели поведения и практика	243
Agile-разработка и изменение культуры	245
Интернализация	246
УРОК 42. Никакие инженерные или управленческие приемы не дадут эффекта, если вы имеете дело с неразумными людьми	247
Попробуйте поделиться знаниями	248
Кто здесь вне очереди?	249
В пользу гибкости	250
Глава 6. Качество.....	252
Введение.....	252
Определения качества.....	252
Планирование качества.....	254
Несколько взглядов на качество	255
Последовательное обеспечение качества.....	256

УРОК 43. Решая вопрос о качестве программного обеспечения, вы можете выбирать: платить немало сейчас или позже, но еще больше.....	259
Кривая роста стоимости исправлений.....	259
Сложнее найти	262
Ранние действия по улучшению качества	263
УРОК 44. Высокое качество естественным образом ведет к повышению продуктивности.....	265
История двух проектов.....	266
Бич переделок	268
Стоимость качества.....	269
УРОК 45. У организаций никогда нет времени, чтобы правильно создать программное обеспечение, но они находят ресурсы, чтобы исправить его позже	271
Почему не сразу?	272
Синдром 100 миллионов долларов	273
Достижение баланса.....	274
УРОК 46. Остерегайтесь малозаметных разрывов между плохим и хорошим	274
Иллюстрация малозаметного разрыва между плохим и хорошим.....	275
Малозаметные разрывы между плохим и хорошим в программном обеспечении	276
УРОК 47. Никогда не поддавайтесь уговорам руководителя или клиента сделать работу наспех	277
Умение противостоять силе	278
Спешка в программировании	278
Нехватка знаний	279
Негласная этика	280
В обход процессов.....	280
УРОК 48. Стремитесь к тому, чтобы дефект нашли коллеги, а не покупатели	281
Преимущества ревью	282
Разновидности ревью программного обеспечения	284
Положительная сторона: культурные последствия ревью.....	285

УРОК 49. Разработчики программного обеспечения любят инструменты, но дурак с инструментами — это вооруженный дурак	287
Инструмент должен добавлять ценность	288
Инструменты должны использоваться разумно	289
Инструмент — это не процесс	290
УРОК 50. Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения	291
Технический долг и профилактическая поддержка.....	292
Осознанный технический долг.....	293
Качественное проектирование — сейчас или позже	294
Глава 7. Совершенствование процессов.....	297
Введение.....	297
Совершенствование процесса разработки: что и зачем	297
Не бойтесь процессов	298
Как взвеси SPI в привычку	299
УРОК 51. Остерегайтесь «менеджмента по Businessweek».....	301
Сначала проблема, потом решение	303
Пример основной причины	304
Постановка диагноза ведет к излечению	306
УРОК 52. Не спрашивайте: «Что это даст мне?» Спрашивайте: «Что это даст нам?»	307
Выгода для команды	308
Личная выгода	309
Вносите свой вклад в общее дело.....	310
УРОК 53. Боль — лучшая мотивация для изменения методов работы	310
Боль причиняет неудобства!	312
Незаметная боль	313
УРОК 54. Внедряя новые методы работы, делайте это мягко, но непрерывно	314
Руководство.....	314
Управление вышестоящими руководителями	317

УРОК 55. У вас нет времени, чтобы совершить все ошибки, сделанные до вас	318
Кривая обучения	319
Хорошие практики	321
УРОК 56. Здравый смысл и опыт иногда важнее определенного процесса	322
Процессы и ритмы	323
Не будьте категоричны	325
УРОК 57. Адаптируйте готовые шаблоны документов	326
УРОК 58. Если не тратить время на учебу и совершенствование, то не стоит ждать, что следующий проект будет реализован лучше предыдущего	331
Оглядываясь в прошлое	332
Структура ретроспективы	334
После ретроспективы	335
УРОК 59. Самая удручающая закономерность в индустрии программного обеспечения — повторение одних и тех же неэффективных действий снова и снова	337
Преимущества обучения	338
Преимущества рассуждения	339
Глава 8. Что дальше?	341
УРОК 60. Невозможно изменить все сразу	342
Приоритизация изменений	344
Проверка реальности	345
Планирование действий	346
Ваши собственные уроки	348
Приложение. Краткий перечень уроков	349
Список использованных источников	353

Как всегда, посвящается Крис

Отзывы о книге

«Это сборник уроков, которые Карл усвоил за свою долгую жизнь и, могу честно сказать, выдающуюся карьеру. Это ретроспектива всего хорошего (и кое-чего плохого), что встретилось ему по пути. Однако это не воспоминание а-ля “в мое время было так”, а ценные советы, которые могут принести пользу любому, кто, пусть даже косвенно, связан с разработкой программного обеспечения. Данная книга удивительна. Это не просто список жемчужин мудрости. В каждом уроке Карл тщательно аргументирует и объясняет, почему то или иное важно для вас, и, что особенно ценно, показывает, как реализовать теорию на практике».

Джеймс Робертсон (James Robertson), автор книги *Mastering the Requirements Process*

«Было бы здорово получать жизненный опыт в самом начале карьеры, когда он особенно необходим, не расплачиваясь за неизбежные ошибки, возникающие в ходе обретения собственного опыта. Более полувека Карл Вигерс (Karl Wiegers) проработал консультантом в сфере разработки и управления программным обеспечением, и его часто приглашали исправить неудачи других людей. В своей книге “Жемчужины разработки” Карл описывает наиболее распространенные и вопиющие ошибки, с которыми он сталкивался. Любому разработчику полезно знать, где таятся самые серьезные проблемы и какие ошибки люди продолжают повторять снова и снова.

Карл не просто специалист по чрезвычайным ситуациям — он хорошо разбирается в передовых методах бизнес-анализа, разработки программного обеспечения и управления проектами. Благодаря его опыту и знаниям вы получите краткую, но важную информацию о том, как оправиться от неудач и, что особенно важно, избежать их.

Сорок шесть лет назад мне посчастливилось наткнуться на классическую книгу Фреда Брукса (Fred Brooks) *The Mythical Man-Month*¹, раскрывшую мне глаза на мою начинающуюся карьеру. Книга Карла во многом подобна ей, но имеет более широкий охват и более актуальна для современного мира. Мой собственный полувековой опыт подтверждает, что он точно угадал с уроками, которые были выбраны для “Жемчужин разработки”.

*Мейлир Пейдж-Джонс (Meilir Page-Jones),
старший бизнес-аналитик, Wayland Systems Inc.*

«Карл написал еще одну замечательную книгу, полную всесторонних советов разработчикам программного обеспечения. Его уроки будут актуальны для профессионалов и учащихся, молодых и старых, начинающих и умудренных опытом. Я занимаюсь разработкой программного обеспечения уже много лет, но эта книга вовремя напомнила мне, что еще можно улучшить в моей команде. Теперь я жду не дождусь, когда все мои коллеги прочитают эту книгу.

“Жемчужины разработки” основана на реальном опыте развития множества проектов и анализе достигнутого, подкрепляющем уроки. Как и во всех книгах Карла, эти уроки получаются простыми и увлекательными. Они наполнены интересными историями и забавными комментариями. Вы можете прочитать книгу от начала до конца или просто углубиться в конкретные разделы, относящиеся к областям, которые вы хотите улучшить сегодня. Увлекательное повествование плюс практические советы — вы не ошибетесь, если выберете эту книгу!»

Джой Битти (Joy Beatty), вице-президент Seilevel

«Книга Карла “Жемчужины разработки” преследует сложную цель — обозначить и объяснить многие особенности, которые вы вряд ли откроете для себя в процессе обучения и которые многие специалисты познают на горьком опыте. Все эти идеи имеют решающее значение для разработки отличного программного обеспечения.

Книга построена так, что вам придется подключить весь свой опыт и определить, как можно изменить свое поведение, и это прекрасно.

¹ Брукс Ф. Мифический человеко-месяц, или Как создаются программные системы. — СПб.: Питер, 2021.

В ней вы найдете 60 уроков, посвященных экосистеме разработки программного обеспечения. Подробный анализ, уже проведенный автором, поможет вам сэкономить время, повысить эффективность совместной работы и иначе взглянуть на распространенные заблуждения, а также даст возможность создавать более совершенные системы. Книга написана простым и доступным языком, и к тому же материал подкреплен большим количеством ссылок на других экспертов, которые в своей практической работе открыли для себя те же идеи.

Эти уроки — настоящие жемчужины: ценные крупицы мудрости, которые помогут вам разрабатывать совершенное программное обеспечение независимо от вашей роли. Подумайте о том, чтобы приобрести два экземпляра: один для себя и один для коллег — оставьте его там, где другие участники команды смогут взять его и обнаружить собственные жемчужины».

Джим Броско (Jim Brosseau), Clarrus

«Это отличная книга для всех, кто занимается разработкой программного обеспечения. Один из потрясающих (и необычных) плюсов книги — ее организация в виде отдельных уроков. Они легко запоминаются при чтении и быстро приходят на ум, когда в них возникает потребность. Такое случилось со мной недавно, когда я обсуждал со старшим руководителем требования к компетенциям в проектах, использующих методы гибкой разработки, и сразу же вспомнил об уроке 8 “Главное требование к разработке — наложенное и эффективное общение”.

Из личного опыта могу подтвердить ценность таких уроков, как № 22 “Проблемы многих систем скрываются в интерфейсах”, поскольку сильно обожгся, не уделив интерфейсам должного внимания. Любой, кто занимается разработкой программного обеспечения, в конечном счете накапливает подобный опыт, который в будущем подсказывает, что надо и чего не надо делать. Эта книга поможет вам обрести его безболезненно. Как говорит Карл в уроке 7, “запись знаний обходится дешевле, чем повторное их обретение”. Эти слова не только являются хорошим советом для практиков, но и четко объясняют, почему вам следует купить данную книгу».

*Говард Подесва (Howard Podeswa),
автор The Agile Guide to Business Analysis
and Planning: From Strategic Plan to Continuous Value Delivery*

Предисловие

Получив степень доктора в области органической химии, Карл Вигерс устроился научным сотрудником в компанию Kodak в Рочестере, штат Нью-Йорк. Перед собеседованием он думал, что понимает особенности будущей работы, и полагал, что будет проводить исследования, связанные с фотопленкой, процессами проявления фотографий и всем подобным.

Когда Карл в первый раз пришел на работу в Kodak, его провели через световой шлюз в лабораторию. Этот шлюз похож на воздушный шлюз на подводной лодке, за исключением того, что предотвращает проникновение света в комнату, которая должна оставаться полностью темной. Карлу потребовалось несколько минут, чтобы глаза привыкли к слабому освещению. Никто его не предупредил, что исследования предстоит проводить в фотолаборатории.

Карл быстро осознал, что не хотел бы тратить годы на работу в прямом смысле в темноте и поэтому вскоре перешел на должность разработчика программного обеспечения, затем администратора ПО и, наконец, руководителя процесса разработки и совершенствования ПО. Позднее он основал собственную компанию Process Impact.

С помощью этой книги Карл пытается вывести других разработчиков ПО из темноты на свет. Как и в других его книгах, здесь больше практики, чем теории. Карл концентрируется на областях, в которых непосредственно имеет опыт работы: особенно это касается требований, совершенствования процессов, качества, культуры и командной работы.

Карл не объясняет, почему для своей книги он выбрал название «Жемчужины разработки». Жемчужина начинает расти, когда в раковину устрицы попадает раздражитель, например песчинка. Защищая себя, устрица начинает выделять перламутр. Процесс длится многие годы,

но в итоге песчинка-раздражитель превращается в ценную жемчужину.

Карл — один из самых вдумчивых людей, которых я знаю. Он тщательно анализировал проблемы, с которыми сталкивался в процессе разработки программного обеспечения, и собрал в этой книге 60 самых ценных советов.

*Стив Макконнелл (Steve McConnell),
Construx Software и автор Code Complete¹*

¹ Макконнелл С. Совершенный код. — СПб.: Питер, 2005.

Благодарности

За пятьдесят с лишним лет я научился разрабатывать программное обеспечение, управлять проектами и совершенствовать процессы. Я прочитал бесчисленное количество книг и статей, посетил множество курсов профессиональной подготовки и прослушал немало докладов на конференциях. Я благодарен моим учителям, которые помогли мне пройти путь от обретения полезных знаний до совершенно нового понимания той или иной части нашей дисциплины. Особенно я хочу отметить Стива Боденхаймера (Steve Bodenheimer) и доктора Джойса Стаца (Dr. Joyce Statz). Сколько имен учителей останется в вашей памяти спустя десятилетия?

Огромное количество литературы по разработке программного обеспечения является практически неисчерпаемым источником просветления. Вот авторы, чьи работы я нашел особенно яркими: Майк Кон (Mike Cohn), Ларри Константин (Larry Constantine), Аллан Дэвис (Alan Davis), Том ДеМарко (Tom DeMarco), Том Гилб (Tom Gilb), Роберт Гласс (Robert Glass), Эллен Готтесдинер (Ellen Gottesdiener), Каперс Джонс (Capers Jones), Норм Керт (Norm Kerth), Тим Листер (Tim Lister), Стив Макконнелл, Роксанна Миллер (Roxanne Miller), Джеймс Робертсон, Сюзанна Робертсон (Suzanne Robertson), Джоанна Ротман (Johanna Rothman) и Эд Юрдон (Ed Yourdon). Если вы не читали их книги и статьи, то обязательно прочтайте. Мне выпала редкая удача за свою карьеру подружиться со многими мудрыми авторами и консультантами.

Мне посчастливилось работать с несколькими талантливыми инженерами-программистами. Наблюдая за работой других, можно многому научиться и пополнить свою копилку опыта. Будучи главным консультантом в моей компании Process Impact, я предоставил свои услуги примерно 150 компаниям и госорганам. Я благодарен всем клиентам и слушателям моих учебных курсов, которые поделились со мной своими историями успеха и неудач. Эти истории помогли мне узнать, какие методы работают или нет в тех или иных ситуациях. Все, что

я узнал из этих многочисленных источников, я постарался изложить в виде уроков в данной книге.

При подготовке книги я имел возможность получить бесценные отзывы Джима Броссо, Тани Чарбери (Tanya Charbury), Майка Кона, Дэвида Хикерсона (David Hickerson), Тони Хиггинаса (Tony Higgins), Норма Керта, Рэмзи Миллера (Ramsay Miller), Говарда Подесвы, Холли Ли Сефтона (Holly Lee Sefton) и особенно Мейлира Пейджа-Джонса, Кена Пью (Ken Pugh) и Кэти Рейнольдс (Kathy Reynolds). Я искренне благодарен им за терпение и ответы на мои вопросы, а также за их опыт, которым они щедро поделились со мной. И спасибо всем, кто прислал мне содержательные отзывы, подкрепившие мои личные наблюдения.

Я признателен Джой Битти, Джиму Броссо, Майку Кону (Mike Cohn), Гари К. Эвансу (Gary K. Evans), Лонни Фрэнксу (Lonnie Franks), Дэвиду Хикерсону (David Hickerson), Кэти Иберле (Kathy Iberle), Норму Керту, Дэррилу Логсдону (Darryl Logsdon), Жаннин Макконнелл (Jeannine McConnell), Марко Негри (Marco Negri), Мейлиру Пейджу-Джонсу, Нилю Поттеру (Neil Potter), Кену Пью, Джине Шмидт (Gina Schmidt), Джейму Шилдсу (James Shields), Джону Зигристу (John Siegrist), Дженейлу Стивену (Jeneil Stephen), Тому Томасовичу (Tom Tomasovic) и Себастьяну Ватцингеру (Sebastian Watzinger) за ценный вклад в рецензирование рукописи. Обзоры комментариев, которые выполнили Тания Чарбери (Tanya Charbury), Кэти Рейнольдс (Kathy Reynolds), Мод Шлих (Maud Schlich) и Холли Ли Сефтон (Holly Lee Sefton), были особенно ценными. Спасибо также Гари К. Эвансу за разрешение изменить удачный рисунок со схемой организации интерфейсов.

Я благодарен Хэзи Гумберт (Haze Humbert), Менке Мехта (Menka Mehta), а также всему редакционному и производственному отделу издательства Pearson за прекрасную работу над моей рукописью.

Как всегда, я в особом долгу перед своей женой Крис за то, что она была терпелива, пока я занимался еще одной книгой.

Об авторе



С 1997 года Карл Вигерс (Karl Wiegers) занимает пост главного консультанта в Process Impact, консалтинговой и обучающей компании в сфере разработки программного обеспечения, находящейся в Хэппи-Вэлли, штат Орегон. До этого Карл 18 лет работал в Kodak, где занимал должности ученого-исследователя фотографического дела, разработчика программного обеспечения, администратора программного обеспечения и, наконец, руководителя процесса разработки и совершенствования программного обеспечения. Карл получил степень доктора органической химии в Университете штата Иллинойс.

Карл — автор 12 книг, в том числе *The Thoughtless Design of Everyday Things*, *Software Requirements*¹, *More About Software Requirements*, *Practical Project Initiation*, *Peer Reviews in Software*, *Successful Business Analysis Consulting*, и детективного романа под названием *The Reconstruction*. Он написал множество статей по разработке программного обеспечения, менеджменту, проектированию, консалтингу, химии и военной истории. Несколько книг Карла были отмечены наградами, одна из последних — премия Общества технических коммуникаций за книгу *Software Requirements*, написанную в соавторстве с Джой Битти. Карл работал в редакции журнала *IEEE Software* и писал статьи для журнала *Software Development*.

Когда он не сидит за компьютером, то любит дегустировать вина, работать волонтером в публичной библиотеке, играть на гитаре, сочинять и записывать песни, читать книги о военной истории и путешествиях. Вы можете связаться с ним через www.processimpact.com или www.karlwiegers.com.

¹ Вигерс К. И. Разработка требований к программному обеспечению.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу:
comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Глава 1

Обучение на собственном опыте

Я не знаю никого, кто мог бы, положа руку на сердце, сказать: «Сегодня я пишу программное обеспечение лучше, чем когда-либо». Всем, кто не может этого сказать, было бы полезно обучиться более эффективным методам работы. Моя книга дает возможность ускорить прохождение этого квеста.

Обретение опыта — это форма обучения, которая лучше всего способствует закреплению знаний и умений. Но она же и самая болезненная. В попытках попробовать новые подходы мы часто терпим неудачу. Мы должны преодолевать трудности и прикладывать усилия, чтобы освоить новые методы и понять, когда и как их использовать.

К счастью, есть другой путь. Кривую обучения можно сделать более пологой, опираясь на уроки, советы и приемы людей, которые уже получили знания и опыт. В книге собраны полезные идеи в области разработки ПО и управления проектами. Эти жемчужины мудрости я обрел на личном опыте и наблюдал в работе других. Ваш собственный опыт и уроки могут быть другими, и вы можете соглашаться не со всем, о чем я рассказываю. Это нормально — опыт каждого уникален. Тем не менее все описанное было весьма ценным для меня и пригодилось в моей карьере программиста.

МОЯ ТОЧКА ЗРЕНИЯ

Позвольте начать с короткого рассказа о моем прошлом, чтобы объяснить, как я усвоил эти уроки. Свой первый курс по программированию я прослушал в 1970-м в колледже. В основе курса лежал, конечно

же, язык FORTRAN. Следующим летом я приступил к выполнению своего первого задания. Нужно было автоматизировать некоторые операции в отделе бухгалтерии в моем колледже, причем сделать это я должен был в одиночку. У меня за плечами было уже два курса по программированию, так что я чувствовал себя полноценным инженером-программистом. Я на удивление легко справился с заданием, если учесть мой небольшой опыт. Несмотря на еще два года учебы, все остальное, что я узнал о программировании, я почерпнул сам из сторонних источников и от своих коллег. В прошлом такое неофициальное начало карьеры не было чем-то необычным, поскольку разработка программного обеспечения привлекала многих людей из разных слоев общества, которые не всегда имели формальное образование в области информатики.

С самого начала я много работал с программным обеспечением: разрабатывал требования, архитектуру и дизайн пользовательского интерфейса, писал код, тестировал его, занимался управлением проектами и писал документацию, наконец, пытался совершенствовать качество и процесс разработки. Попутно я защитил докторскую диссертацию по органической химии. Уже тогда третья моей диссертации состояла из программного обеспечения, которое анализировало экспериментальные данные и моделировало химические реакции.

В начале карьеры в компании Eastman Kodak, превратившейся в огромную и успешную корпорацию, я использовал компьютеры, чтобы заниматься проектированием и анализом экспериментов. Вскоре я перешел на должность программиста и занялся созданием приложений для исследовательских лабораторий Kodak, а затем несколько лет руководил небольшой группой разработчиков. Я обнаружил, что моя научная подготовка позволяет мне использовать более систематический подход к разработке программного обеспечения.

В 1983 году я написал первую статью о программном обеспечении. С тех пор из-под моего пера вышло много статей и восемь книг на темы, связанные с программированием. Как независимый консультант и преподаватель, начиная с 1997 года я оказал услуги почти 150 компаниям и государственным учреждениям их многих сфер. Эта деятельность позволила мне увидеть, какие методы эффективно работают в программных проектах, а какие — нет.

Ко многим своим идеям, связанным с разработкой программного обеспечения и управлением проектами, я пришел на личном опыте. Одни

из них были полезными, другие — ошибочными. Некоторые идеи я почерпнул из опыта моих клиентов, в основном из проектов, потерпевших крах. Никто не вызывает консультанта, когда все идет хорошо. Я написал эту книгу, чтобы вам не приходилось медленно и мучительно проходить все те же уроки на личном опыте. Один опытный инженер-программист, прочитавший этот список уроков, сказал: «С каждым из этих уроков связан шрам (или несколько), полученный при его усвоении».

О КНИГЕ

Эта книга содержит 60 уроков, связанных с разработкой и управлением программным обеспечением. Уроки разбиты на шесть групп, каждая из которых описывается в отдельной главе.

- Глава 2. Требования.
- Глава 3. Проектирование.
- Глава 4. Управление проектами.
- Глава 5. Культура и командная работа.
- Глава 6. Качество.
- Глава 7. Совершенствование процессов.

Глава 8 содержит заключительный общий урок, о котором следует помнить всегда. Для удобства все 60 уроков перечислены в приложении.

Я не пытался составить исчерпывающие списки уроков в названных областях. В каждой накоплено так много знаний, что никто не сможет представить более или менее полный список. Кроме того, я не рассматриваю другие важные и наиболее очевидные аспекты разработки ПО: программирование, тестирование и управление конфигурацией. Обсуждение этих процессов вы найдете в трудах других авторов, например, в таких книгах, как:

- *Programming Pearls*¹ Джона Бентли (Jon Bentley) (Bentley, 2000);
- *Lessons Learned in Software Testing* Джема Канера (Cem Kaner), Джеймса Баха (James Bach) и Брета Петтикорда (Bret Pettichord) (Kaner, Bach, Pettichord, 2002);

¹ Бентли Дж. Жемчужины программирования. 2-е изд. — СПб.: Питер, 2002.

- *Code Complete* Стива Макконнелла (McConnell, 2004);
- *Software Engineering at Google*¹ Титуса Уинтерса (Titus Winters), Тома Мэншрека (Tom Manshreck) и Хайрама Райта (Hyrum Wright) (Winters, Manshreck, Wright, 2020).

Темы и уроки в этой книге в значительной степени независимы, поэтому вы можете знакомиться с ними в любой последовательности. Каждая глава начинается с краткого обзора соответствующей области разработки программного обеспечения. Затем несколько врезок под названием «Первые шаги» предлагают вам поразмышлять о предыдущем опыте работы, прежде чем погрузиться в уроки главы. Эти врезки предлагают задуматься о проблемах, с которыми сталкиваются команды в данной области, влиянии этих проблем и возможных глубинных причинах.

Каждый урок начинается с краткого изложения основной мысли, далее следует обсуждение и приводятся рекомендации, которые команды могут применять в своей практике. Читая каждую главу, думайте о том, как описанные приемы можно использовать в вашей ситуации.



Значок с изображением книги на полях, как здесь, указывает на реальную историю из личного опыта или опыта моих клиентов или коллег. Все истории реальны, хотя имена участников изменены, чтобы сохранить конфиденциальность.



Ключ на полях указывает на основные моменты в описании каждого урока.



Некоторые уроки содержат ссылки на другие уроки. В таких случаях приводится этот значок со стрелкой.

Врезка «Следующие шаги» в конце каждой главы подскажет вам, как применить полученные сведения в вашем проекте, команде или организации. Независимо от вида проекта, над которым вы работаете, его жизненного цикла или создаваемого продукта, ищите идею, которую несет каждый урок, и думайте, как вы можете адаптировать ее, чтобы помочь проекту стать более успешным.

Попробуйте проработать материал всех врезок вместе с коллегами. В начале всех обучающих курсов, которые я вел, участники объединялись в небольшие группы и обсуждали задачи, стоявшие перед их

¹ Винтерс Т., Маншрек Т., Хайрам Р. Делай как в Google. Разработка программного обеспечения. — СПб.: Питер, 2021.

командами («Первые шаги»). В конце те же группы изучали варианты решения этих задач, проводили мозговой штурм, посвященный тому, как решить их с помощью материалов курса («Следующие шаги»). Мои студенты увидели, что в эти дискуссионные группы полезно включать представителей разных заинтересованных сторон, так как они высказывали свою точку зрения на функционал проекта и его продвижение. Благодаря всестороннему обсуждению можно достичь глубокого понимания текущей деятельности и творчески подойти к выбору решений.

Я надеюсь, что многие из моих уроков найдут отклик и мотивируют вас попробовать что-то новое в своих проектах. Однако вы вряд ли сможете изменить все сразу. Разные люди, команды и организации воспринимают новый опыт с разной скоростью. Заключительная глава «Что дальше?» поможет вам наметить путь, как реализовать уроки на практике. В ней предлагается установить приоритеты изменений, которые вы хотели бы внести, и разработать план действий, который поможет перейти из точки, в которой вы находитесь сегодня, в точку, куда хотели бы попасть.

ПРИМЕЧАНИЕ О ТЕРМИНОЛОГИИ

В этой книге я использую термины «система», «продукт», «решение» и «приложение» более или менее взаимозаменяемо. В каждом случае я имею в виду просто конечный результат вашего проекта, поэтому, пожалуйста, не ищите особого смысла в этих терминах. Независимо от того, работаете ли вы с корпоративной или государственной информацией, сайтами, коммерческими программными продуктами или аппаратными устройствами со встроенным ПО, уроки и соответствующие практики найдут свое применение.

ВАШИ ВОЗМОЖНОСТИ

Если вы действительно не относитесь к той редкой категории практиков, которые уже создают программное обеспечение лучше, чем кто-либо другой, то у вас всегда найдется что улучшить. Мы все — и отдельные разработчики, и группы, и организации — должны постоянно расширять свои возможности. Мы все хотим иметь как можно меньше шрамов.

Младший разработчик по имени Захари Минотт (Zachary Minott, 2020) поделился своей методикой, которая помогла ему превзойти более опытных разработчиков. Минотт описал этический подход, согласно которому он признает отсутствие у себя тех или иных знаний и обязуется изучать и применять новые знания на практике. Он сказал: «Если у меня и есть какая-то сверхспособность, то это способность быстро учиться и сразу же применять новые знания в своей работе». Минотт открыл важнейший механизм, позволяющий ему постоянно совершенствоваться в своей сфере.

Нам всем нужно постоянно совершенствоваться,
мы все хотим иметь как можно меньше шрамов.

Возможно, вы решите пройти курс обучения, чтобы освоить новый навык или усовершенствовать привычные методы работы. Пока вы учитесь, невыполненная работа продолжает накапливаться. Из-за спешки велик соблазн проигнорировать новые знания и продолжать работать по старинке. Вы можете чувствовать себя вполне комфортно, поскольку текущих знаний и умений пока достаточно. Но это не путь к совершенствованию.

В каждом проекте я выделил две области, в которых нужно совершенствоваться. Я тратил какую-то часть времени, чтобы изучить новые темы и попытаться применить новые знания. Удача не всегда сопутствовала мне, но мой подход позволил постепенно накопить навыки, сослужившие мне хорошую службу.

Я призываю вас поступить так же. Не просто читайте книгу — делайте следующий шаг. Решите, как вы и ваши коллеги сможете применять описанные здесь методы и что, по вашему мнению, они дадут вам. Составьте список практик, о которых хотели бы узнать больше, а затем применяйте их. Так постепенно вы достигнете прогресса.

Глава 2

Требования

ВВЕДЕНИЕ

У каждого проекта есть свои задачи — цели или результаты, на достижение которых направлена работа. Каждый проект также имеет требования, определяющие, что необходимо для удовлетворения запросов бизнеса или вывода продукта на рынок. В большинстве проектов с самого начала в требованиях не хватает многих деталей. Эти детали начинают обретать все более ясные черты по мере того, как клиенты осваивают программный продукт и присылают команде разработчиков свои замечания и предложения. Требования могут быть точно задокументированы или существовать только в головах заинтересованных сторон. В любом случае при отсутствии четкого понимания требований весьма маловероятно, что команде удастся достичь поставленных целей.

В конечном итоге команда (или, по крайней мере, большинство ее членов) выявит все требования заказчика. И чем раньше это произойдет — хотя бы до того, как команда решит, что разработка завершена, — тем проще и дешевле будет их воплотить.

Множество типов требований

Изучать требования гораздо сложнее, чем просто спросить пользователей, чего они хотят. (См. урок 11 «Люди не просто так собирают требования».) Первая сложность заключается в том, что все по-разному представляют, что такое *требование*. В литературе по программному



обеспечению можно найти множество определений. Вот одно из них (Wiegers, Beatty, 2013):

Требование — это заявление о потребности или цели клиента либо об условиях или возможностях, которыми должен обладать продукт, чтобы удовлетворить такую потребность или цель. Свойство, которым должен обладать продукт, чтобы представлять ценность для клиента.



Термин «требования» охватывает множество категорий; некоторые из них определены в табл. 2.1 (Wiegers, Beatty, 2013; ПВА, 2015). У программистов нет единого мнения о том, как называть каждую категорию. Но название не так важно, как признание необходимости исследовать, записывать и передавать эти разнообразные категории требований людям, которые будут работать над проектом (см. табл. 2.1).

Таблица 2.1. Несколько категорий информации о требованиях

Категория требований	Краткое определение
Бизнес	Бизнес-цели или задачи, для достижения или решения которых был запущен проект. Могут быть записаны в документе о видении и масштабе, уставе проекта или экономическом обосновании
Пользовательские	Описание действий, задач или целей, которые пользователи должны иметь возможность выполнять или достигать с помощью продукта. Обычно представлены в виде вариантов применения или пользовательских историй. Иногда под пользовательскими требованиями понимаются требования заинтересованных сторон, охватывающие более широкий круг запросов, выходящих за рамки использования продукта
К решению	Описание возможностей и качеств решения, отвечающего требованиям заинтересованных сторон
Функциональные	Описание поведения, которое продукт должен демонстрировать при определенных условиях. Основная часть требований к решению — функциональные требования. Разработчики пишут код, удовлетворяющий функциональным требованиям, которые соответствуют конкретным запросам пользователя

Категория требований	Краткое определение
Нефункциональные	Детали требований к решениям, которые обычно относятся к качеству и эксплуатационным характеристикам продукта, также называемым <i>атрибутами качества</i>
К внешнему интерфейсу	Описание связей продукта с внешним миром, включая пользователей, другие программные системы, аппаратные устройства и механизмы взаимодействий
К переходу	Описание условий, которым должен соответствовать продукт, или действий, которые необходимо выполнить, чтобы обеспечить успешную миграцию из текущего состояния в обновленное

Проще говоря, бизнес-требования описывают, *почему* организация берется за проект. Пользовательские требования описывают, *что* пользователи смогут делать с помощью продукта. Функциональные требования сообщают разработчикам, *что* они должны создать. Согласование бизнес-, пользовательских и функциональных требований — важный компонент успешного планирования.

Основу составляет набор требований к продукту или решению, которые описывают возможности продукта, функциональное поведение и характеристики. В проектах часто есть дополнительные требования к переходу, описывающие, что еще должно быть сделано в рамках проекта, помимо создания самого продукта (ПВА, 2015). Примерами таких требований к переходу могут служить: создание и передача учебных материалов; создание документации для сертификации продукта; создание вспомогательной документации; перенос данных и другие действия, необходимые для того, чтобы помочь пользователям перейти от текущего состояния к обновленному.

Подобласти разработки требований

Обширную область разработки требований можно разделить на подобласти разработки и управления требованиями. Они охватывают пять основных видов деятельности, перечисленных в табл. 2.2 (Wiegers, Beatty, 2013). Команды разработчиков программного обеспечения не всегда строго последовательно выполняют различные действия по разработке требований, а делают это, то забегая вперед, то возвращаясь назад.



Таблица 2.2. Подобласти разработки требований

Подобласть	Деятельность	Описание
Разработка требований	Выявление	Действия по выявлению и изучению запросов клиентов и определению требований к решению, удовлетворяющему эти запросы
	Анализ	Действия для достижения четкого и полного понимания требований, их уточнения до соответствующего уровня детализации, определения приоритетов и выявления взаимосвязей между ними
	Детализация	Действия, направленные на формирование знаний о требованиях, их хранение и передачу заинтересованным сторонам
	Проверка	Действия, помогающие подтвердить, что решение, удовлетворяющее указанным требованиям, также будет удовлетворять запросы клиентов
Управление требованиями		Действия, направленные на отслеживание состояния требований во время разработки, обеспечение реакции на их изменение и отслеживание требований для последующих продуктов

Самое главное в процессе разработки требований к программному обеспечению — убедиться, что команда понимает и решает реальную задачу, которая может отличаться от задачи, первоначально озвученной заказчиком. Клиенты часто высказывают свои идеи вместо потребностей. Эти идеи не всегда отражают реальную задачу и могут приводить к реализации решения, не соответствующего действительности.



В отличие от большей части работы по созданию программного продукта, работа с требованиями связана не столько с программированием, сколько с общением между людьми. Требования к разработке сложны, поэтому не следует ожидать, что каждый член команды будет владеть ими в полной мере. Для работы с требованиями многие

организации формируют штат людей, обладающих высокой квалификацией: подготовленных и опытных бизнес-аналитиков, продакт-менеджеров или (в проектах, использующих подходы Agile-разработки) владельцев продуктов. Сегодня для обозначения людей, которые аккумулируют итоговые требования в программных проектах, все чаще используется термин «бизнес-аналитик» и почти не употребляются другие термины, такие как «инженер по требованиям», «аналитик требований», «системный аналитик» и просто «аналитик». Если различие между ролями несущественно, то для обозначения тех, кто занимается формированием требований, я буду использовать термин «бизнес-аналитик» независимо от должностей или других обязанностей этих людей.

В отличие от большей части работы по созданию программного продукта, работа с требованиями связана не столько с программированием, сколько с общением между людьми.

Роль бизнес-аналитика

В последние годы важность бизнес-анализа в проекте была признана благодаря созданию профессиональных организаций, таких как Международный институт бизнес-анализа (International Institute of Business Analysis, IIBA; www.iiba.org). Эти организации разработали своды знаний и программы сертификации (IIBA, 2015). Даже если в команде нет отдельной должности бизнес-аналитика, его роль играют другие члены команды, работающие с заинтересованными сторонами, стремясь понять требования и определить решения.

Квалифицированные бизнес-аналитики выявляют реальные потребности заинтересованных сторон и пишут документацию для дизайнеров, разработчиков, тестировщиков и других специалистов. Специально назначенный бизнес-аналитик может оценить требования в широком бизнес-контексте, так как имеет представление о них на уровне системы или предприятия. Когда клиенты сообщают о своих потребностях непосредственно разработчикам, обе стороны имеют лишь ограниченное представление о системе со своих точек зрения. Бизнес-аналитик же предлагает представление более высокого уровня.

Различные организации поручают своим бизнес-аналитикам выполнять разные функции в проектах. Обычно бизнес-аналитики руководят деятельностью по разработке требований и управлению проектом. Они организуют обсуждения с представителями пользователей и с помощью различных методик стараются выявить все требования. Получая информацию от компетентных заинтересованных сторон, бизнес-аналитики структурируют ее, записывают и распространяют.

Требования — это фундамент

Требования служат основой для проектов. Не существует единственного «правильного» способа обработки требований. Для проектов разработки программного обеспечения можно выбрать нужное из множества жизненных циклов и моделей разработки, которые подразумевают различные способы представления требований. Но при этом важно, чтобы все разработчики обладали одной и той же информацией для создания правильного программного обеспечения независимо от подхода к разработке, принятого в команде. (См. урок 6 «Agile-требования не отличаются от других».)



Не все команды пишут конкретные технические задания. Тем не менее они аккумулируют всю информацию о разного рода требованиях и хранят ее в некоем документе, который можно назвать спецификацией требований.

Некоторые мои клиенты спрашивали меня: «Как компании, которые действительно хорошо собирают и формулируют требования, делают это?» На что я всегда отвечал: «Не знаю, они мне не звонят». Трудно узнать, что делают организации, освоившие процесс разработки требований, если они не делятся своим опытом через публикации или презентации. У меня также было несколько клиентов, которые говорили мне: «Вы здесь, потому что мы столкнулись с непреодолимыми трудностями». Чаще всего главной причиной этих трудностей были недоработки в требованиях.

Все команды должны серьезно относиться к требованиям, принимать и адаптировать выбранные методы разработки требований, учитывая характер своего проекта и командную культуру. Команды разработчиков программного обеспечения, которые пренебрегают требованиями, увеличивают риск провала проекта. Примерно с 1985 года я много размышлял о том, как помочь командам разработчиков ПО и систем разобраться с требованиями. В этой главе описываются 16 ценных уроков, которые я усвоил за это время.

ПЕРВЫЕ ШАГИ: ТРЕБОВАНИЯ



Прежде чем вы перейдете к изучению уроков, связанных с требованиями, предлагаю вам потратить несколько минут на следующие действия. По мере чтения подумайте, в какой степени каждый из этих пунктов применим к вашей организации или команде.

1. Перечислите методы работы с требованиями, в которых особенно преуспела ваша организация. Задокументирована ли информация об этих методах? Доступна ли она другим членам команды, чтобы они могли ознакомиться с этими методами и применять их на практике?
2. Определите любые проблемы (болевые точки), которые можно отнести к недостаткам, мешающим командам разобраться с требованиями по проекту.
3. Опишите, как каждая проблема влияет на вашу способность успешно завершать проекты. Как они мешают достижению успеха в бизнесе и разработчикам, и их клиентам? Проблемы могут привести к материальным и нематериальным затратам из-за незапланированных доработок, задержек, поддержки и сопровождения продукта, негативных отзывов и неудовлетворенности клиентов.
4. Для каждой проблемы, выявленной на шаге 2, определите основные причины, провоцирующие или усугубляющие ее. Одни первопричины могут скрываться внутри команды или организации; другие спровоцированы внешними условиями и неподконтрольны вам. Проблемы, последствия и первопричины могут сливаться, поэтому постарайтесь разделить их и увидеть, как они связаны. Вы можете найти несколько основных причин, способствующих появлению одной и той же проблемы, или несколько проблем, обусловленных одной общей причиной.
5. Читая эту главу, отметьте любые практики, которые могут быть полезны вашей команде.

Урок 1

Если вы неверно определили требования, то неважно, насколько хорошо вы выполните остальную часть работы



Бизнес-аналитик одного из моих клиентов рассказал о неудаче, постигшей проект. Их IT-отдел взялся за создание новой информационной системы для своей компании. Команда разработчиков считала, что прекрасно понимает требования системы и без дополнительных

отзывов пользователей. Участники команды не были высокомерными, просто оказались чересчур уверенными в себе. Однако, когда разработчики представили готовую систему, реакция пользователей была такой: «А если серьезно, где наше приложение?» Они категорически забраковали систему.

Разработчики были шокированы; они думали, что работают на совесть и создали правильный продукт. Однако отказ от взаимодействия с пользователями, которое помогает убедиться в правильном понимании требований, был серьезным упущением.

Когда вы с гордостью представляете миру свое новое детище, то едва ли хотите услышать: «Ваше детище уродливо». В описанном случае произошло именно это. Итак, что же сделала компания? Разработчики переделали систему, на этот раз внимательно прислушиваясь к пользователям. (См. урок 45 «У организаций никогда нет времени, чтобы правильно создать программное обеспечение, но они находят ресурсы, чтобы исправить его позже».) Это был дорогой урок о важности участия клиента в составлении требований.

Независимо от того, создаете вы новый продукт или улучшаете существующий, требования — это фундамент для всей последующей работы над проектом. Проектирование, конструирование, тестирование, документирование, обучение и миграция из одной системы или операционной среды в другую зависят от наличия правильных требований. Многочисленные исследования показали, что эффективная разработка и своевременное информирование о требованиях являются решающими факторами успеха любого проекта. И наоборот, неадекватное видение проекта, неполные и неточные требования, а также меняющиеся требования и цели проекта — частые причины неудачи (PMI, 2017). Правильное определение требований помогает гарантировать, что решение будет соответствовать видению продукта и бизнес-стратегии организации (Stretton, 2018). Не сформулировав правильно требования, вы потерпите неудачу.



При отсутствии качественной проработки требований заинтересованные стороны могут удивиться тому, что предлагает команда разработчиков. Такие программные сюрпризы часто оказываются неприятными.

Правильные требования, но когда?

Я не говорю, что нужно иметь полный набор требований перед началом реализации. Это нереально для любых продуктов, кроме самых маленьких и стабильных. Всегда будут появляться новые идеи, изменения и исправления, которые вы должны учитывать в своих планах развития. Но для любой части системы, которую вы создаете, будь то отдельная фаза разработки, конкретный релиз или полный продукт, вам необходимо иметь как можно более полные и правильные требования. В противном случае планируйте доработку после окончания реализации. В проектах, практикующих Agile-разработку, предусмотрен дополнительный этап проверки требований. Чем дальше первоначальные требования от того, что действительно нужно клиентам, тем больше доработок понадобится.

Некоторые утверждают, что невозможно сразу правильно выявить все требования, что клиенты всегда предлагают добавить что-то еще — и поэтому среда постоянно развивается. Возможно, они правы, но я скажу так: «В таком случае вы никогда не сможете закончить проект». Учитывая, что всегда можно что-то добавить, вы никогда не сможете идеально выполнить требования. Но оговоренный объем разработки требует, чтобы вы сделали все правильно, иначе успеха вам не видать.

Дело обстоит немного иначе, если вы создаете инновационный продукт. Если никто и никогда не делал ничего подобного, то у вас вряд ли все получится с первой попытки. Ваша первая попытка — это, по сути, этап проверки гипотез и определения требований экспериментальным путем. Однако в конечном счете ваши эксперименты приведут к тому, что вы разберетесь в возможностях и характеристиках нового продукта — его требованиях.

Правильные требования, но как?

Ничто не заменит постоянного взаимодействия с клиентами, при котором можно определить набор точных, четких и своевременных требований. (См. урок 12 «Выявление требований должно помочь разработчикам услышать голос клиента».) Нельзя просто провести встречу в самом начале, а затем сказать клиентам: «Мы позвоним вам, когда закончим». В идеале команда должна быть на связи с представителями клиентов на протяжении всего времени разработки проекта. У разработчиков будет много вопросов и моментов, требующих уточнения.





Участники команды должны будут прописать общие требования на ранних этапах и постепенно уточнять их в дальнейшем. Команде нужна надежная частая обратная связь с пользователями и другими заинтересованными сторонами, чтобы подтвердить правильность понимания требований и предлагаемых решений.

Не всегда получается уговорить клиентов на такое тесное взаимодействие. В конце концов, у них есть своя работа; руководители могут воспротивиться тому, чтобы некоторые из их лучших сотрудников тратили много времени на проект. «Вы можете посетить одну-две встречи, — скажет руководитель, — но я не хочу, чтобы эти программисты постоянно отрывали вас от дела своими вопросами».

Один из способов убедить клиента в необходимости постоянного взаимодействия — привести примеры проблем, с которыми столкнулась организация из-за недостаточного участия клиентов в разработке продукта. Еще лучше рассказать об опыте, когда взаимодействие с клиентами окупилось. Другой метод убеждения — предложить четкий план взаимодействий для уточнения требований, а не оставлять этот вопрос полностью открытым. Подобный план может предусматривать ряд неформальных обсуждений, встреч, обзоров требований и утверждения эскизов, прототипов и последовательности релизов.

Клиенты с большей вероятностью будут в восторге от проекта и с готовностью согласятся внести свой вклад, если увидят ощутимый прогресс, например в виде периодического выпуска новых версий работающего программного обеспечения. Они также воодушевятся, если увидят, что их вклад действительно влияет на продвижение проекта. Иногда бывает трудно убедить заказчиков принять обновленную программную систему. Представители заказчиков, работавшие с командой разработчиков, знающие суть нововведений и понимающие их необходимость, могут помочь облегчить переход.

Я работал с несколькими представителями клиентов, которые оказали огромное влияние на успех проекта. Помимо информации о требованиях, некоторые из них также предоставили эскизы пользовательского интерфейса и тесты, помогающие убедиться в правильной реализации разных частей программного обеспечения. Трудно переоценить вклад таких преданных клиентов, помогающих команде разработчиков точно определить требования и выработать правильное решение.

При отсутствии качественной проработки требований заинтересованные стороны могут удивиться тому, что предлагает команда разработчиков. Такие программные сюрпризы часто оказываются неприятными. Я хочу, чтобы, увидев продукт, мои клиенты реагировали так: «Ого! Карл, получилось лучше, чем я мог себе представить. Спасибо!»



Урок 2

Основной результат разработки требований — общее видение и понимание

Материальным результатом разработки требований является документ в некой хранимой форме. Обычно это письменный документ, часто называемый спецификацией требований к программному обеспечению, бизнес- или рыночными требованиями. Как вариант, требования можно оформить в виде каталожных карточек, стикеров на доске, диаграмм, приемочных тестов или их комбинаций.



Однако наиболее важными результатами разработки требований являются общее понимание и согласие заинтересованных сторон в отношении решения, которое будет создано командой проекта. Добившись этого понимания, можно проверить, соответствуют ли предлагаемый объем и бюджет проекта необходимым возможностям и характеристикам решения.

Управление ожиданиями — важная часть управления проектами. Разработка требований направлена на формирование общих ожиданий (общего видения) у представителей заинтересованных сторон. Спецификации требований содержат информацию об особенностях соглашения. Благодаря общему видению согласовывается вся деятельность, связанная с проектом (Davis, 2005):

- работа, которую финансирует спонсор проекта;
- решение, которое, как ожидают клиенты, позволит им достичь бизнес-целей;
- программное обеспечение, которое проверяют тестировщики;
- продукт, который отделы маркетинга и продаж предлагают миру;
- планы и списки задач, создаваемые руководителями и командами разработчиков проекта.

Часто трудно определить, одинаково ли понимают несколько человек нечто столь сложное, как проект по разработке программного обеспечения. Я бывал на встречах, где участники достигали определенного согласия, а позже выяснялось, что некоторые детали соглашения (и, следовательно, результат) они понимают по-разному. Эти различия могут привести к тому, что участники будут стремиться к противоположным целям.

Заявление о видении определяет общую стратегическую цель, к достижению которой должны стремиться все участники проекта.



Заявление о видении помогает достичь общего понимания и непротиворечивых ожиданий. Я использовал следующий шаблон заявления, чтобы помочь заинтересованным сторонам сформулировать свои мысли (Wieggers, Beatty, 2013; Moore, 2014):

Для	[целевые клиенты]
Которые	[изложение бизнес-потребностей или возможностей]
Предполагает- ся создать	[название продукта или проекта]
Являющийся	[тип продукта или проекта]
Позволяющий	[основные возможности продукта; основные преимущества, которые он обеспечит; веская причина покупки продукта или реализации проекта]
В отличие от	[текущей реальности или альтернативных продуктов]
Наш продукт	[краткое изложение основных преимуществ этого продукта по сравнению с текущей реальностью или товарами конкурентов]

В качестве простого примера ниже приводится заявление о видении, которое я написал для сайта, созданного мной в целях поддержки книги.

Несмотря на то что этот крошечный проект я целиком хранил в своей памяти, написание заявления о видении в самом начале помогло внести ясность в то, чего я надеялся достичь с помощью сайта.

Для читателей, которые интересуются книгой «Жемчуг из песчинок», предполагается создать сайт PearlsFromSand.com, позволяющий посетителям получить информацию о книге и ее авторе, приобрести копии в различных форматах и создать сообщество людей, заинтересованных в обмене своим жизненным опытом. В отличие от сайтов, которые просто описывают и рекламируют книгу, наш продукт — PearlsFromSand.com — позволит посетителям делиться своим жизненным опытом, а также читать и комментировать сообщения друг друга.

Если у вашего проекта нет заявления о видении, то никогда не поздно его написать. На своих учебных занятиях при обсуждении требований к программному обеспечению я прошу студентов написать заявление о видении своего текущего проекта с использованием этого шаблона. Меня всегда впечатляют краткие описания, которые студенты создают всего за пять минут. Из их заявлений о видении я могу быстро понять, с какой целью создаются проекты.

Когда занятия посещают несколько человек из одной команды, бывает так, что их заявления о видении существенно различаются. Я часто прошу, чтобы представители разных заинтересованных сторон, имеющие разные точки зрения, писали заявления о видении по отдельности. В дальнейшем, сравнив эти заявления, можно увидеть, пришли ли заинтересованные стороны к общему пониманию того, куда движется проект. Наличие расхождений подсказывает, что члены команды должны еще поработать над согласованием своих ожиданий.

У моей подруги-консультанта был точно такой же опыт работы с клиентским проектом. Она сказала: «Я попросила представителей четырех основных заинтересованных сторон изложить на бумаге собственное видение, пока мы все были в одной комнате. В результате получились очень разные и кое в чем несовместимые заявления. О таких вещах лучше узнавать заранее».

Заявление о видении определяет общую стратегическую цель, к достижению которой должны стремиться все участники проекта. Если в ходе работы над проектом видение изменится, то заказчик должен сообщить об этих изменениях всем, кого они касаются, чтобы у людей по-прежнему было единое понимание требований. Заявление о видении



не заменяет спецификации требований, но задает отправную точку, позволяющую гарантировать, что требования, озвученные команде, соответствуют этому видению и, следовательно, успех будет достигнут.

Урок 3

Интересы всех сторон нигде не пересекаются так явственно, как в требованиях



Консультант и писатель Тим Листер (Tim Lister) трактует успех проекта как «удовлетворение набора всех требований и соблюдение всех ограничений, определяющих ожидания ключевых заинтересованных сторон». Эта формулировка подразумевает, что команда разработчиков проекта должна определить заинтересованные стороны и порядок взаимодействия с ними, чтобы понять эти требования и ограничения.

Заинтересованная сторона — это любое лицо или группа людей, активно участвующих в проекте, затрагиваемых им или способных влиять на его продвижение. Связи между заинтересованными сторонами и проектом могут быть разными. Одним заинтересованным сторонам просто навязывают результат разработки проекта; другие тщательно прорабатывают требования. И среди них всегда будет тот, кто сможет изменить направление проекта или даже остановить его.

Заинтересованная сторона — это любое лицо или группа людей, активно участвующих в проекте, затрагиваемых им или способных влиять на его продвижение.

Заинтересованные стороны могут быть внутренними по отношению к команде проекта или организации, занимающейся разработкой, либо внешними. На рис. 2.1 перечислены типичные заинтересованные стороны, мнение которых необходимо учитывать при разработке большинства программных проектов. В зависимости от категории продукта могут быть и другие заинтересованные стороны: корпоративная информационная система, коммерческое программное приложение, государственная система или материальный продукт, содержащий встроенное ПО.



Рис. 2.1. Заинтересованные стороны, определяющие требования к проекту, которым он должен удовлетворять, и ограничения, которые он должен соблюдать

Анализ заинтересованных сторон

Команда проекта должна заранее определить потенциальные группы заинтересованных сторон. Не удивляйтесь, если список получится пугающе длинным. Потребуется приложить некоторые усилия, чтобы определить ваши заинтересованные стороны. Однако это намного лучше, чем игнорировать критически настроенное сообщество и вносить корректировки на поздних стадиях проекта.

Конечные пользователи и клиенты, приобретающие продукты, которыми будут пользоваться другие, выступают основными источниками требований. Заказчик, который выбирает продукт или платит за него, не всегда использует его и может иметь неправильное представление о том, что нужно пользователям для выполнения их работы. Многие продукты имеют широкий круг конечных пользователей.



Чтобы упростить исследование требований, разделите своих *пользователей на группы* с разными наборами потребностей (Wiegers, Beatty, 2013). Пользователи могут быть даже не людьми, а аппаратными устройствами или другими программными системами, взаимодействующими с вашими продуктами. Вам нужно будет определить людей, которые могут предоставить требования от имени этих компонентов.



Обычно мы представляем себе *прямых пользователей*, которые будут непосредственно взаимодействовать с продуктом, но у вас могут быть и *непрямые пользователи*. Они могут предоставлять данные, поступающие в информационную систему, или получать выходные данные системы, даже притом что сами не будут генерировать выходные данные. Однажды я участвовал в создании корпоративной системы, объединявшей данные из десятков проектов и составлявшей ежемесячные отчеты, которые затем рассылались многим руководителям. Эти руководители были непрямыми пользователями — они не работали с самой системой. Тем не менее, будучи получателями системных отчетов, они были основными заинтересованными сторонами.



Один мой коллега лаконично описал непрямого пользователя: «Ваш клиент и после удаления из списка остается вашим клиентом». Чтобы идентифицировать непрямых пользователей, нужно подняться на один или два уровня выше непосредственного контекста приложения и посмотреть, какие группы людей и какие другие системы должны быть представлены. Необходимо также определить классы нежелательных пользователей, которые *не должны* иметь возможности пользоваться системой. Например, хакеры не являются заинтересованными сторонами (они не будут определять требования или ограничения), но вам нужно предвидеть и пресекать их злые намерения.

Попробуйте ответить на следующие вопросы о каждой выявленной вами группе заинтересованных сторон.

Кто они? Опишите каждую группу, чтобы все участники проекта понимали, кто это. Описания заинтересованных сторон могут повторно использоваться в нескольких проектах организации.

Насколько они заинтересованы? Подумайте о том, как сильно результат проекта повлияет на группу и насколько велико их желание участвовать в проекте. Вам нужно узнать об ожиданиях, инте-

ресурсах, опасениях и ограничениях каждой группы заинтересованных сторон.

Какое влияние на проект они имеют? Определите, какие решения каждая заинтересованная сторона может и не может принимать. Какие группы обладают наибольшим влиянием на проект? Каковы их взгляды и приоритеты? Вы должны уделить особое внимание группам, которые проявляют наибольший интерес к проекту и оказывают наибольшее влияние на него (Lucidchart, 2021).

С кем лучше всего говорить? Определите подходящих представителей в каждом сообществе, с которыми нужно работать. Они должны быть авторитетными источниками информации.

Где они? Сбор информации — итеративный процесс, требующий множества встреч. Пуще всего получить информацию от группы заинтересованных сторон, если у вас есть прямой доступ к отдельным ее представителям. В противном случае продумайте механизмы и протоколы удаленного взаимодействия.

Что мне нужно от них? Определите информацию, решения и данные, которые понадобятся получить от каждой группы. Это поможет вам выбрать наилучшие способы получения информации в нужное время. Для каждой группы пользователей вы должны понять их требования (что продукт должен позволять им делать) и их ожидания в отношении качества. Некоторые группы заинтересованных сторон будут накладывать ограничения, которые должна соблюдать команда проекта. Ограничения делятся на несколько категорий, среди которых:

- финансовые, временные и ресурсные ограничения;
- применимые политики, нормы и стандарты (бизнес-правила);
- совместимость с другими продуктами, системами или интерфейсами;
- юридические или договорные ограничения;
- требования к сертификации;
- ограничения возможностей продукта (то есть какие функции *не должны* включаться в продукт).

Что им нужно от меня? Одни заинтересованные стороны нужно проинформировать о существенных проблемах, которые могут их затронуть, поэтому вам необходимо знать, какая информация



о проекте актуальна для каждой группы. Другим может понадобиться пересмотреть требования, чтобы убедиться, что они не противоречат существующим правилам и ограничениям. Взаимодействуйте с заинтересованными сторонами, чтобы понять, чего они ожидают от вас, и сообщайте им о своих ожиданиях. Успех сотрудничества во многом зависит от создания и поддержания взаимного доверия с помощью эффективного общения.

Как и когда мне с ними взаимодействовать? Как только вы очертите круг заинтересованных сторон, подумайте о том, какими способами лучше всего обмениваться с ними необходимой вам всем информацией. Если у вас нет прямого доступа к реальным представителям определенной группы пользователей, то подумайте о возможности создания *персон*, воображаемых людей, которые будут замещать реальных (Cooper et al., 2014).

Какие заинтересованные стороны наиболее важны при разрешении конфликтов? При разрешении противоречивых требований и принятии важных решений оцените, какой результат больше всего соответствует бизнес-целям проекта. Одни группы пользователей могут быть предпочтительнее других; удовлетворение их потребностей способствует успеху в бизнесе в большей степени, чем удовлетворение требований других групп пользователей. Продублируйте влияние и интересы заинтересованных сторон, чтобы выявить эти приоритеты, не ждите, пока назреет первый конфликт.

Кто звонит?

Важно сразу определить лиц, принимающих решения. В некоторых случаях таким лицом может быть один человек, например спонсор проекта или владелец продукта. Взаимодействие с ним наиболее эффективно при условии, что у него есть вся нужная информация, чтобы принять соответствующее решение, и он сможет быстро сделать это при необходимости. Однако чаще вам придется определять правильные группы людей для принятия решений из разных областей. На принятие групповых решений потребуется больше времени, зато они лучше отражают все интересы, соответствующие целям проекта.



Лица, ответственные за принятие решений, касающихся нескольких заинтересованных сторон, должны основывать свои решения на бизнес-целях проекта. Цели, заявление о видении, ограничения проекта

и другие бизнес-требования обычно оформляются в виде документа о видении и масштабе проекта или в виде устава проекта (Wiegers, 2007; Wiegers, Beatty, 2013). В проекте, не имеющем четких бизнес-требований, мало оснований для принятия и аргументирования важных решений.

Мы все на одной стороне

Не всегда получается поразить все заинтересованные стороны результатами проекта. Напряженность между сторонами может расти, если люди преследуют противоположные цели, стараясь защитить свои интересы. Выстраивание отношений в духе сотрудничества с ключевыми заинтересованными сторонами имеет большое значение для достижения успеха проекта. Поскольку в будущем вам, возможно, придется работать с теми же людьми, стоит с самого начала строить общение на основе взаимоуважения.

Урок 4

В требованиях в первую очередь важны особенности использования, а затем — функциональность

Один из наших внутренних корпоративных пользователей попросил мою команду добавить новую возможность в приложение, которое использовала его группа. Он настаивал, что эта возможность необходима, поэтому мы удовлетворили его просьбу. Однако, насколько нам известно, никто так и не воспользовался добавленным функционалом. В следующий раз я бы скептически отнесся к просьбе данного клиента.



В индустрии программного обеспечения существует убеждение, согласно которому (в зависимости от источника, который вы читаете) от 50 до 80 % возможностей ПО используются редко или не используются никогда (The Standish Group, 2014). У меня нет точных цифр, но я могу смело заявить, что значительная доля функционала поставляемого программного обеспечения не представляет большой ценности для конечных пользователей. Используют ли ваши сотрудники возможности каждого приложения в полной мере? Мои — нет. Я написал множество книг и статей, пользуясь Microsoft Word, но в Word есть куча возможностей, которыми я никогда не пользовался и не буду. То же верно и для других приложений, которыми я пользуюсь. К со-

жалению, индустрия ПО прилагает значительные усилия для реализации возможностей, которые практически не используются.

Зачем нужна лишняя функциональность?

Если разработчики стремятся удовлетворить требования к возможностям продукта, растет риск появления невостребованных функций. Если заказчик хочет, чтобы функции можно было постоянно расширять, их количество будет увеличиваться. Ориентированность на богатство функциональных возможностей может привести к выпуску продукта, который, казалось бы, имеет нужные функции, но не позволяет пользователям решать их задачи.



Я рекомендую обсуждать не только возможности самого продукта, но и то, что пользователи будут с ним делать. Мы должны смещать акцент с функциональности на особенности использования, с решения на потребность. Это быстрее поможет бизнес-аналитику и команде разработчиков понять контекст и цели пользователя. Имея такую информацию, бизнес-аналитик сумеет точнее определить, какими возможностями должно обладать решение, для кого оно, как и когда будет использоваться.

Сочетая оба подхода, то есть учитывая и технические возможности, и особенности использования, можно определить функциональные требования, которые будут озвучены разработчикам. Однако фокус на особенностях использования позволит гарантировать, что в продукт будут добавлены те функции, которые действительно необходимы пользователям для выполнения их задач. Это снижает риск создания избыточной функциональности, которая вроде и кажется хорошей идеей, но на деле не помогает пользователям достичь конкретных целей. Ориентируясь на особенности использования, мы повышаем удобство работы с продуктом, поскольку разработчики могут взвешенно подходить к реализации каждого элемента функциональности с учетом задач или целей пользователя (Constantine, Lockwood, 1999).

Я рекомендую смещать обсуждение требований с самого продукта на то, что пользователи будут с ним делать.

Ставим особенности использования на первое место

Когда при определении требований мы ориентируемся на особенности использования, немного меняется суть вопросов, которые бизнес-аналитик может задавать во время сбора информации. Вместо «Что вы хотите?» или «Что, по вашему мнению, должна делать система?» бизнес-аналитик может спросить: «Как предполагается использовать систему?» В процессе обсуждения определяются задачи, которые пользователи должны решать с помощью системы.



Обсуждение вариантов использования — хороший способ понять эти задачи (Kulak, Guiney, 2004). Редкие пользователи запускают приложение ради определенной функции; подавляющее большинство пользуются им, чтобы достичь конкретной цели. Каждый раз, когда я запускаю свое приложение финансового учета, я преследую одну или несколько целей, например хочу сверить состояние счета кредитной карты, перевести средства на личный банковский счет, оплатить что-либо или открыть депозит. Каждая из этих целей — вариант, или случай, использования. Я открываю приложение, имея определенное намерение, и выполняю последовательность шагов, вызывая функции, необходимые для решения задачи. Если все идет хорошо, то я успешно решаю свою задачу и закрываю приложение — желаемый результат достигнут.

Есть несколько причин, объясняющих привлекательность вариантов использования. Прежде всего, они помогают представителям пользователей задуматься о своих потребностях. Пользователям сложно правильно сформулировать перечень функций, которые должны быть реализованы в продукте, но они с легкостью расскажут о сценариях использования из своей повседневной жизни. Эти сценарии помогают структурировать и организовать описания связанных функций. В шаблон варианта использования можно включить дополнительное описание, настолько подробное, насколько сочтет нужным ваша команда (Wiegers, Beatty, 2013). Связанная функциональность включает описание наиболее типичной последовательности взаимодействий или действий по умолчанию для задачи (обычный сценарий) и любые варианты этой типичной последовательности (альтернативные сценарии). По этим описаниям бизнес-аналитик или разработчики смогут определить, какие функции должно предоставить приложение, чтобы пользователи могли решать поставленные задачи. В надлежащем описании варианта использования также учтено, какие ошибки могут

возникнуть и как система должна их обрабатывать (нештатные ситуации).

Анализ, ориентированный на особенности использования, помогает расставить приоритеты. К функциональным требованиям с наивысшим приоритетом относятся те, которые позволяют пользователям решать самые важные задачи. Одни варианты использования будут более важными и востребованными, чем другие, поэтому должны быть реализованы в первую очередь. В пределах одного варианта использования наивысший приоритет имеет нормальный сценарий вместе с возможными непривычными ситуациями. Альтернативные сценарии имеют более низкие приоритеты, и часто их допускается реализовать позже или даже не реализовать никогда. Изучение особенностей применения поможет вам определить, какие варианты использования встречаются чаще и должны иметь наивысший приоритет. (Подробнее об особенностях применения см. урок 15 «Когда принимаете решение о добавлении функций, избегайте расстановки приоритетов по децибелам».)



Поставив себя на место пользователя, вы сможете продуктивнее взаимодействовать с ним и получить более полное представление об ограничениях реализации, которое труднее получить в случае подхода, ориентированного на функциональность продукта. Если пользователи с помощью продукта не смогут решать свои задачи или он им не понравится, то добавление дополнительных функций не повысит их лояльность.

Проблема пользовательских историй

Многие Agile-команды записывают требования в виде пользовательских историй. По словам эксперта по гибкой разработке Майка Кона, «пользовательская история описывает функциональность, которая будет полезна как пользователю, так и покупателю системы или программного обеспечения» (Cohn, 2004). Пользовательские истории обычно записываются с использованием простого шаблона:

Как <тип пользователя>, я хочу <описание решаемой задачи>, чтобы я мог <достижение некой цели>.

или

Как <тип пользователя>, я хочу достичь <некая цель>, чтобы <некий результат>.

Пользовательские истории напоминают членам команды о необходимости обсудить перед реализацией недостающие детали.

Одна из проблем, связанных с пользовательскими историями, заключается в том, что они не имеют внутренней организационной схемы. Простой сбор множества пользовательских историй, даже если они записываются по приведенному шаблону, мало чем отличается от старого метода опроса пользователей: «Чего вы хотите?» Вы получаете много битов важной информации, случайно перемешанной с посторонними сведениями.

В одном большом проекте было собрано несколько тысяч пользовательских историй, записанных на желтых стикерах. Некоторые истории были непонятными; многие противоречили друг другу. Одни казались повторяющимися, а другие — соблазнительно многообещающими, но неполными. Проект располагал большим объемом неорганизованной информации самого разного вида, каждый фрагмент которой был помечен как пользовательская история. По этому объему сведений было сложно определить, какие истории были связаны с задачами пользователей и соответствовали бизнес-целям проекта, а какие являлись просто мыслями, высказанными вслух.



Одни из этих пользовательских историй были ориентированы на особенности использования, другие — нет. Истории охватывали широкий спектр деталей разной важности. Они варьировались от «Как пользователь, я хочу, чтобы экранный шрифт был без засечек, чтобы я мог легко его прочитать» до «Как начальник отдела труда и заработной платы, я хочу, чтобы система рассчитывала налог в фонд пособий по безработице для каждого штата, в котором у нас есть сотрудники, чтобы мы могли правильно платить этот налог». Подобные истории касаются не пользователей и особенностей использования ими продукта, а скорее функций и свойств системы.

Для накопления обширной коллекции историй, содержащих отдельные фрагменты функциональности, требуется, чтобы кто-то отсортировал их и выявил темы, связанные с пользовательскими задачами. Упорядочение любого объема информации похоже на собирание пазла: вы берете фрагменты по одному и размышляете: «Интересно, куда он подходит». Мой мозг работает лучше, когда я использую подход сверху

вниз. Я предпочитаю начинать с общих мазков, например, определяя пользовательские задачи, а затем постепенно уточнять и детализировать их. Так у меня меньше шансов упустить что-то важное; к тому же мне требуется меньше усилий, чем если бы я собирал весь пазл, выкладывая фрагменты по одному.

На первый взгляд, простой шаблон пользовательской истории кажется удобным вариантом записи требований пользователей. Полезно знать, какая группа пользователей запрашивает те или иные функциональные возможности, чтобы потом, в нужное время, обговорить с ними конкретные детали. Такое откладывание изучения деталей до момента, когда информация действительно понадобится, — эффективный способ распределить ограниченное время. Требования, написанные в такой форме, могут быть ориентированы на особенности использования, описывать задачи и содержать цели, которых хотят достичь пользователи. Однако автор Радж Нагаппан (Raj Nagappan 2020a) указывает на возможные проблемы из-за неправильного применения шаблона пользовательской истории, например, когда истории в большей степени сосредоточены на решениях, а не на задачах, или когда отсутствуют необходимые детали. Альтернативой является шаблон *истории работы* (*job story*), который емко и структурированно описывает потребность (Klement, 2013):

Когда <ситуация>, я хочу <выполнить некую задачу>, чтобы я мог <достичь результата>.

Правила использования

Подход, ориентированный на особенности использования, я начал применять к разработке требований в 1994 году и быстро понял, насколько он целенаправленнее и эффективнее по сравнению с моим предыдущим подходом. Правильно оформленные с точки зрения задач, целей и вариантов использования истории работы и пользовательские истории побуждают участников разработки требований сосредоточиться на особенностях использования продукта людьми, а не только на его функционале.

Даже выбрав pragматичный подход и ограничившись подмножеством вариантов использования, вы все равно придетете к решениям, удовлетворяющим потребности ваших клиентов, если будете ориентироваться на особенности использования.

Урок 5**Разработка требований — итеративный процесс**

На заре моей карьеры программиста я часто начинал писать код, имея лишь смутное представление о том, что должна делать программа. Иногда я работал практически вхолостую: писал и переписывал код, внося множество изменений, но почти не продвигаясь вперед. Я начинал паниковать, понимая, что боксую. В конце концов я осознал проблему: недостаточная продуманность требований к программе. Мои фальстарты были результатом бессистемного подхода к написанию кода — я не прорабатывал алгоритмы в своей голове. Осознав это, я начал тратить больше времени на тщательное изучение требований и только потом открывал редактор исходного кода. После этого я никогда больше не паниковал.

В то же время, приступая к разработке программного обеспечения для других людей, я старался завершать обсуждение с клиентами, как только чувствовал, что понял сказанное ими и что у меня есть вся необходимая информация. Но в процессе работы с первоначальными требованиями у меня часто возникали вопросы и обнаруживались пробелы в знаниях. Мне приходилось возвращаться к клиентам, чтобы прояснить задачи, освежить информацию в памяти и закрыть пробелы. Клиенты не всегда были рады видеть меня снова, но мы понимали, что разработка требований — итеративный и поэтапный процесс.

Постепенное уточнение деталей

Осознав, насколько важно понять направление движения, прежде чем приступить к написанию кода, я понял, что никогда не смогу заранее узнать все требования даже для небольшого приложения, а также не смогу с самого начала продумать все детали каждого требования. Но потом я пришел к выводу, что в этом нет ничего страшного. Мне не нужны все подробности сразу. Мне просто нужно достаточно информации, чтобы начать думать.

Вы должны получить довольно точную информацию о требованиях, прежде чем создавать какую-либо часть продукта, иначе вам придется создавать ее заново.



Эффективная разработка требований подразумевает постепенное уточнение набора требований и их деталей. Вы не сможете правильно определить все требования при первом обсуждении. Однако вы должны получить достаточно точную информацию о требованиях, прежде чем создавать какую-либо часть продукта, иначе вам придется создавать ее заново. Ниже схематично описан процесс, который, по моему мнению, успешно завершится после того, как будут определены бизнес-требования к проекту.

- Шаг 1. Разработайте предварительный список пользовательских требований (сценариев использования или пользовательских историй). Узнайте достаточное количество подробностей о каждом из них, чтобы понять их объем и относительную важность.
- Шаг 2. Распределите пользовательские требования по предстоящим циклам разработки в зависимости от их приоритета. Одни должны быть реализованы первыми; другие могут подождать.
- Шаг 3. Продолжите выявлять и уточнять детали тех требований, реализация которых запланирована в ближайшем цикле разработки, отделяя функциональные требования от пользовательских.
- Шаг 4. Перераспределите приоритеты, добавляя любые новые требования, которые привлекли ваше внимание, а затем перемещайтесь вниз по списку приоритетов по мере продолжения разработки.
- Шаг 5. Вернитесь к шагу 2 и повторите.

Постоянная расстановка приоритетов играет важную роль, поскольку нет смысла углубляться в детали требований, реализация которых не горит. По мере развития проекта некоторые потребности могут отойти на второй план или стать неактуальными. Майк Кон отмечает: «Перед реализацией требований команды должны убедиться, что хорошо понимают их и что реализовать их нужно именно сейчас, а не потом...» (Cohn, 2010).

Возникающие функциональные требования

Когда люди начинают использовать программное приложение, у них появляются новые идеи: «Было бы неплохо, если бы...» или «А что,

если бы я мог...» Возможно, пользователь думает о более простом способе выполнения какого-либо действия или обнаруживает, что, выполняя действие А, хотел бы ненадолго перейти к действию Б. Если эти идеи достаточно важны, то вам потребуется изменить систему, чтобы добавить их. Те элементы функциональности, которые нельзя определить заранее, относятся к *возникающим требованиям* (Cohn, 2010). Независимо от жизненного цикла разработки планы проектов должны учитывать это естественное и полезное расширение требований.



Это не означает, что необходимо построить систему целиком или хотя бы завершить полную итерацию, основанную на уже имеющихся знаниях, а затем добавлять все дополнительные функциональные возможности. Мы можем использовать разные методы для выявления некоторых из возникающих требований. Один из подходов заключается в том, чтобы создать несколько представлений требований. Вместо того чтобы просто записывать сценарии использования, функциональные требования или пользовательские истории, нарисуйте несколько картинок. Визуальные модели описывают требования на более высоком уровне абстракции, позволяют отвлечься от деталей и увидеть более широкую картину рабочего процесса и взаимосвязей.

Написание тестов дает возможность взглянуть на требования под другим углом. Тесты определяют приемы, с помощью которых можно узнать, работает ли система, как ожидается. При этом важно понимать, что написание тестов отличается от описания ожидаемого поведения системы в определенных условиях. Создавая тесты на ранней стадии, вы можете обнаружить неясности и ошибки в требованиях, а также отсутствующие требования, например необработанные исключения. Можно даже заметить, что некоторые требования просто не нужны, если нельзя придумать тесты, требующие их реализации. Идея опережающей разработки тестов легла в основу Agile-подхода к разработке через тестирование (Beck, 2003).

Прототипы — эффективное средство воплощения требований в жизнь. Благодаря им пользователи получают нечто более осозаемое, чем список функциональных требований или стопка карточек с историями. Прототипы могут быть простыми или сложными, концептуальными или конкретными, бумажными или программными (Wiegers, Beatty, 2013). Итеративное прототипирование ускоряет обсуждение требований и помогает пользователям находить ошибки и упущения в требованиях до того, как на создание продукта будет затрачено слишком много усилий.



Подробнее о прототипировании рассказывается в уроке 17 «Проектирование — итеративный процесс».

Возникающие нефункциональные требования

По аналогии с функциональными требованиями детали некоторых нефункциональных требований тоже трудно выяснить на ранней стадии. Насколько доступным, надежным или удобным должно быть приложение? Запланируйте несколько циклов исследования, чтобы определить измеримые, реально достижимые и экономически эффективные целевые значения для каждого важного атрибута качества.



Например, не ждите осмысленного ответа, когда впервые спросите пользователя: «Каковы ваши требования к удобству использования?» Начните с простого осознания важности данного удобства. Со временем вы расширите это осознание до понимания различных аспектов и в итоге определите целевые характеристики каждого из них. Однако в случае с атрибутами качества хитрость заключается в том, чтобы получить достаточную информацию как можно раньше, чтобы команда могла принять архитектурные решения, позволяющие достичь целевых показателей каждого атрибута. Добавлять новую функциональность гораздо проще, чем исправлять недостатки фундаментальной архитектуры. (См. урок 20 «Невозможно оптимизировать все желаемые атрибуты качества».)



Формирование полезного набора требований любого рода требует терпения для переосмыслиния, повторных попыток и накопления знаний, необходимых для создания правильного продукта. Я не знаю никаких коротких путей.

Урок 6

Agile-требования не отличаются от других

Многие компании, занимающиеся разработкой программного обеспечения, используют методы Agile-разработки, по крайней мере в некоторых своих проектах. Бизнес-аналитики и владельцы продуктов иногда применяют термин «*Agile-требования*» для описания своей работы (Leffingwell, 2011). Он подразумевает, что требования к проектам, где используются методики Agile-разработки, каким-то образом качественно отличаются от требований к проектам с использованием других

подходов. На мой взгляд, между ними нет никакой разницы (Wiegers, Beatty, 2016).

Важно понимать, что для правильной реализации требуемой функциональности разработчику нужна одна и та же информация независимо от методологии, используемой в процессе разработки. Проекты, где действуются гибкие и традиционные способы разработки, по-разному обрабатывают требования. Тем не менее большинство устоявшихся методов разработки и бизнес-анализа могут пригодиться и в Agile-проектах, если будут применяться обдуманно.

Agile-подходы подразумевают, что мы адаптируемся к неизбежным изменениям, а не воображаем, что все требования можно досконально понять на раннем этапе и они будут оставаться неизменными на протяжении всего проекта. Однако все проекты требуют выполнения одних и тех же основных шагов по работе с требованиями. Кто-то должен составить список заинтересованных сторон, собрать требования из различных источников и подтвердить, что решение, основанное на этих требованиях, достигнет бизнес-целей проекта. Основные различия в обработке требований в Agile- и традиционных проектах можно разделить на несколько категорий.



Для правильной реализации требуемой функциональности разработчику нужна одна и та же информация независимо от методологии, используемой в процессе разработки.

Роли и обязанности

В состав большинства традиционных проектных групп входит один или несколько бизнес-аналитиков, которые отвечают за работу по выявлению требований к проекту, анализу, спецификации, проверке и управлению или возглавляют ее. Во многих Agile-проектах отсутствует официальная должность бизнес-аналитика. В таких случаях за определение объема и границ проекта, создание и ведение списка пожеланий и подготовку пользовательских историй к реализации отвечает владелец продукта (Cohn, 2010; McGreal, Jocham, 2018). Разработка требований — совместный процесс, в котором



участвуют владелец продукта, представители пользователей и другие заинтересованные стороны. (См. урок 12 «Выявление требований должно помочь разработчикам услышать голос клиента».) Но за полноту информации в историях ответственность несут разработчики, а не бизнес-аналитики.



Терминология

Проекты, разрабатываемые с применением традиционных подходов, обычно основаны на вариантах использования и функциональных требованиях. Agile-команды, напротив, часто опираются на пользовательские истории, обобщенные описания функций или задач, разбитые на более мелкие пользовательские истории, приемочные тесты и желания клиентов (Cohn, 2004). Но это те же самые знания о требованиях, просто они по-другому называются. Независимо от представления или названия команда должна создавать и передавать эти знания, чтобы каждый мог эффективно выполнять свою работу.



Детальность документации

Agile-методы полагаются на доступность документации в нужный момент. Благодаря тесному сотрудничеству клиентов с разработчиками в Agile-проектах требования могут содержать меньше деталей, чем в традиционных проектах. Заинтересованные стороны могут сообщить все необходимые детали, когда это действительно потребуется, на встречах и в соответствующей документации. Некоторые пользовательские истории могут содержать мало подробностей, а сложные или важные функции могут прорабатываться более детально.



Однако не стоит чрезмерно полагаться только на вербальное общение. Воспоминания могут быть неполными, непоследовательными и отрывочными. Люди приходят в проектные команды и уходят. И, конечно же, система продолжает функционировать еще долгое время после завершения первоначальной разработки. Кто-то должен обновлять и совершенствовать ее, обеспечивать производственную поддержку (иногда среди ночи) и, в конце концов, вывести ее из эксплуатации. Каждой проектной группе необходимо создать достаточный объем документации, позволяющей решать эти задачи, но при этом не тратить время на запись информации, которую никто не будет использовать. (См. урок 7 «Запись знаний обходится дешевле, чем повторное их обретение».)

Выбор времени для выполнения действий

При любом подходе к разработке программного обеспечения можно составить первоначальное мнение, расставить приоритеты и определить последовательность действий, начав с изучения высокоуровневых требований. Традиционные подходы предполагают создание довольно полного списка требований на ранней стадии проекта. Это приводит к успеху в одних ситуациях и к провалу — в других.

Agile-команды, напротив, предпочитают детализировать требования непосредственно перед реализацией определенной части функциональности. Это снижает риск того, что информация устареет или требования станут неактуальны, когда разработчики и тестировщики начнут с ними работать. Как показано на рис. 2.2, заинтересованные стороны и владелец продукта собирают некоторые первоначальные данные и анализируют их. Затем владелец продукта распределяет пользовательские истории и пожелания по конкретным итерациям для реализации. Владелец продукта, разработчики и заказчики будут дополнительно уточнять детали каждой истории с помощью обычных действий по разработке требований, создавая письменную документацию, объем которой не больше необходимого. И команда будет продолжать принимать и реализовывать требования на протяжении всего проекта.

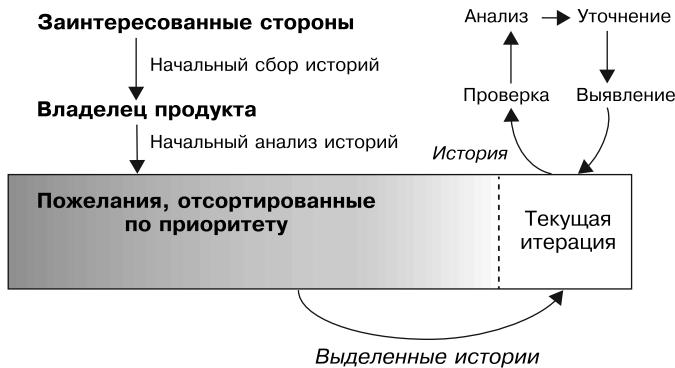


Рис. 2.2. Работа с требованиями выполняется в каждой итерации Agile-разработки

Детализация по мере необходимости снижает количество ненужных усилий, но затрудняет выявление зависимостей между требованиями и потенциальных архитектурных последствий, которые желательно

устранять как можно раньше, чтобы обеспечить стабильное развитие продукта. Чтобы снизить риск неправильной организации архитектуры, Agile-команды должны уже на ранних итерациях рассмотреть широкий спектр вопросов и подумать, какие важные архитектурные решения могут потребоваться. Точно так же команда должна как можно раньше начать изучение нефункциональных требований, чтобы проект достиг высокой производительности, доступности и других важных целей.

Готовые формы

В общем случае пользовательские истории похожи на сценарии использования. Разница лишь в том, насколько тщательно вы их детализируете и записываете ли информацию. В традиционном проекте бизнес-аналитик может разработать набор функциональных требований на основе сценариев использования. Многие Agile-команды конкретизируют каждую пользовательскую историю, определяя критерии приемки и тесты, которые покажут, правильно ли разработчики реализовали ее. Если в вашем проекте есть тесты, проверяющие специфические детали требований, то, читая другие уроки в этой главе, подумайте, как они применимы к вашим тестам.



На самом деле функциональные требования и соответствующие им тесты — альтернативные варианты представления одной и той же информации. Требования определяют, что создавать; тесты описывают, как выяснить, демонстрирует ли система ожидаемое поведение. Требования и тесты *вместе* представляют собой мощную комбинацию. Наилучшие результаты получаются, когда разные люди пишут требования и тесты на основе одного и того же источника информации, например сценария использования.



Каждый раз, создавая и сравнивая эти два представления требований, я находил пробелы, неясности и различия в интерпретации. Исправление этих недостатков во время исследования требований обходится гораздо дешевле, чем их поиск в реализованном программном обеспечении. Можно прибегнуть к альтернативной стратегии — записывать детали пользовательской истории в форме приемочных тестов, которые затем проверяет тестировщик. Записывая требования и тесты, вы подключаете двух человек для поиска проблем.

Когда создается несколько представлений требований, нестыковки между ними помогают выявить проблемы. Если вы создаете только

одно представление, то независимо от выбранного метода будете вынуждены доверять его точности.

Когда расставлять приоритеты

Расстановка приоритетов учитывает относительную ценность каждого требования для клиента по сравнению с усилиями, риском и стоимостью его реализации. Традиционные проекты могут определить приоритеты требований на раннем этапе и в дальнейшем почти не пересматривать их. В Agile-проекте задачи приоритизируются непрерывно. Вы должны постоянно выбирать, что добавить в ближайшие итерации, а что может быть вообще исключено. Команды всегда спрашивают: «Что из наиболее важного мы должны сделать дальше?» На самом деле все команды, а не только те, кто практикует методы гибкой разработки, должны управлять оставшимися задачами, чтобы как можно быстрее передать клиенту максимально ценный для него продукт.

Есть ли разница?

Большинству клиентов неинтересно, как создаются программные приложения. Они хотят лишь, чтобы продукты отвечали их потребностям, были эффективными, удобными и легко расширяемыми, а также удовлетворяли другие ожидания в отношении качества. Большинство методов разработки требований и управления ими, которые используются в традиционных проектах, в равной степени применимы и к Agile-проектам. Любая команда должна адаптировать методы разработки, чтобы они соответствовали их целям, культуре, среде и ограничениям.

В Agile-разработке всякое изменение добавляет независимую часть функциональности. Продукт — это система, которую вы итеративно совершенствуете, извлекая уроки из неудачных изменений и отыскивая лучшие способы реализации каждой его части. Выделение небольших требований, каждое из которых определяет ограниченное и функционирующее изменение, — совсем другой вид анализа, отличающийся от дробления большого интегрированного решения на фрагменты, вписывающиеся в итерации разработки.

Процесс бизнес-анализа в Agile-проектах тоже несколько иной. Несмотря на то что устаревшие методы бизнес-анализа все еще находят применение (анализ заинтересованных сторон и бизнес-правил, моделирование процессов и многое другое), их интеграция в поэтапный

гибкий процесс оказывается большой проблемой для многих бизнес-аналитиков (Podeswa, 2021). Эксперт по бизнес-анализу Говард Подесва отмечает:

Успешный переход требует развития нового мышления. Роль бизнес-аналитика в Agile-проекте заключается не столько в том, чтобы заранее определить, что будет сделано, сколько в постоянном просмотре условий в процессе разработки. Происходит постоянная оценка того, что должна или не должна делать команда, — компромиссов, необходимых для максимизации доставляемой ценности.



Однако в целом сведения о требованиях, используемые в Agile-проекте, качественно не отличаются от сведений в традиционном проекте. Разработка требований по-прежнему сводится к выявлению и передаче точной информации, чтобы участники проекта могли благополучно создать определенную часть продукта.

Урок 7

Запись знаний обходится дешевле, чем повторное ихобретение

Рассказывая о требованиях студентам, я спрашиваю, приходилось ли кому-нибудь из них исследовать существующие системы, чтобы выяснить, как их изменить или добавить новые функции. Почти все поднимают руку. Затем я спрашиваю, кто из них записывал все, что узнал, для дальнейшего использования. На этот раз руки поднимали единицы. Это означает, что если кому-то в будущем придется внести дополнительные изменения в ту же часть системы, то ему придется повторить процесс исследования.



Восстановление знаний путем исследования — утомительное занятие. Делать это снова и снова — неэффективно. Когда вы записываете все, что узнали, эта информация может пригодиться вам или кому-то еще, если придется вернуться к ней. Рекомендую записывать знания о плохо документированной системе, чтобы постепенно накапливать их в процессе работы над ней. На запись знаний почти всегда требуется меньше времени, чем на повторное ихобретение.

Я отказался бы от записи новых знаний, только если бы был уверен, что никому, в том числе и мне, не понадобится снова работать с этой частью системы. Но поскольку я не умею предсказывать будущее, то

предпочитаю хранить информацию в той форме, которой можно поделиться. Это лучше, чем хранить ее в памяти, где она со временем стирается.

Один из моих клиентов реконструировал полный набор сценариев использования своего плохо документированного флагманского продукта. Затем на основе этих сценариев команда разработала исчерпывающий набор тестовых примеров, что позволило проводить тщательное регрессионное тестирование по мере развития продукта. При этом они обнаружили, что время, потраченное на запись знаний, полученных в процессе исследования, стоило того.



Боязнь рутины

Некоторые отказываются тратить время на документирование требований. Но самое сложное не в том, чтобы записать требования, а в том, чтобы понять, что они собой представляют. Точно так же люди иногда неохотно пишут план проекта. Опять же, самое сложное в этом — продумать все действия, необходимые для завершения проекта: определить результаты, задачи, зависимости, необходимые ресурсы, планы и т. д. Написание плана — ручная работа, требующая времени. Тем не менее я уверен, что на это уйдет меньше времени, чем на попытку устно передать ту же информацию нескольким людям на протяжении всего времени существования проекта. Кроме того, при наличии письменной документации не приходится полагаться на чью-то память и способность точно помнить всю информацию.

Возможно, команды, неохотно документирующие свои требования, боятся оказаться в аналитическом параличе. Симптом этой ловушки — кажущийся бесконечным поток требований к разработке, а разработка не может начаться, пока не будут собраны все требования. Анализический паралич потенциально опасен, но здравый смысл позволит избежать этого риска. Боязнь аналитического паралича не может служить причиной для отказа документировать информацию об основных требованиях.

Однажды меня назначили ведущим бизнес-аналитиком в стороннюю команду, которой предстояло заняться важным корпоративным проектом. Две предыдущие попытки реализовать этот проект по какой-то причине, которую я так и не узнал, зашли в тупик. Когда мы с моими коллегами-аналитиками предложили ключевому клиенту поговорить о требованиях, он отказался. «Я передал свои требования вашим пред-



шественникам, — сказал он. — У меня больше нет времени говорить о требованиях. Создайте мне систему!» Последние три слова стали мантрой нашей группы бизнес-аналитиков.

К сожалению, предыдущие команды не задокументировали полученную информацию. Нам пришлось начинать с нуля. Наш клиент был недоволен, но согласился на встречу, когда мы пообещали использовать хорошо зарекомендовавшие себя приемы разработки требований и записать все, что узнаем. Эта попытка реализовать проект увенчалась успехом. Моя компания решила передать реализацию системы сторонней компании, специалисты которой убедились, что задокументированные нами требования обеспечивают прочный фундамент для разработки. Если бы первые две команды записали то, что узнали, то мы получили бы большое преимущество.

Преимущества письменного общения



В начале своей карьеры программиста я работал над проектом с двумя другими разработчиками. Двое из нас трудились в одном офисе, а третий — в здании за полкилометра от нас. У нас не было ни письменного плана проекта, ни перечня требований, но было общее понимание, куда мы движемся. Мы встречались раз в неделю, чтобы устно обсудить сделанное и планы на следующую неделю. И дважды случалось так, что один разработчик уходил с еженедельной встречи с неправильным пониманием принятых нами решений. Неделю он работал не в том направлении, а потом ему приходилось переделывать созданное. Нам было бы проще записывать планы и требования по ходу работы, чтобы исключить подобные ситуации. Я больше никогда не повторял эту ошибку.

Команды, практикующие методику гибкой разработки Scrum, проводят короткие *ежедневные встречи-планерки*, чтобы уточнить информацию о состоянии дел, выявить препятствия и согласовать свои действия на следующие 24 часа (Visual Paradigm, 2020). Это легче сделать, когда все находятся в одном месте или подключены к общей компьютерной сети. Однако после таких встреч не остается никаких документов, что не способствует хоть сколько-нибудь долгосрочному планированию. Если вы уверены, что никому и никогда не понадобится пересматривать решения или сведения, которыми участники обменивались на собрании, то нет причин записывать их. В противном случае время, необходимое для документирования этой полезной информации, не будет потрачено впустую.



Если бы каждый участник проекта имел право присутствовать на каждом обсуждении, интерпретировал информацию так же, как все остальные, и обладал идеальной памятью, то вам никогда не пришлось бы ничего записывать. Увы, в реальности все совсем не так. Документация играет роль постоянной групповой памяти, ресурса, к которому члены команды могут обращаться в любой момент (Rettig, 1990). В будущем записи помогут освежить воспоминания о намерениях и сравнить их с тем, что получилось на самом деле. Если впоследствии кому-то, кто не участвовал в разработке с самого начала, потребуется внести изменения в продукт, то хорошая документация позволит сэкономить время.

Чтобы не ждать, когда пользователи увидят работающее программное обеспечение и смогут предоставить полезные отзывы, можно передать документально оформленные требования приглашенным экспертам в предметной области и с их помощью выявить проблемы до того, как они будут реализованы в коде. Многие проекты используют сложную логику или бизнес-правила, которые лучше всего представлять в виде таблиц решений или математических формул. Сохраните эту информацию, чтобы иметь возможность проверить ее точность и полноту. Такая документация пригодится и для организации тестирования.



Если кто-то уже потрудился подготовить документацию для тех, кто работает с системой, то обязательно используйте эти сведения. Одна проектная группа разработала хороший набор требований, но сторонняя команда, привлеченная для реализации системы, проигнорировала их. Подрядчики решили сами поговорить с пользователями об их потребностях, тем самым раздражая их и теряя время.

Если бы каждый участник проекта имел право присутствовать на каждом обсуждении, интерпретировал информацию так же, как все остальные, и имел идеальную память, то вам никогда не пришлось бы ничего записывать.



Один из моих клиентов однажды нанял команду бизнес-аналитиков, которые создали несколько папок с требованиями для очень крупного проекта. Затем компания привлекла вторую команду для создания

продукта. Команда, нанятая для реализации, увидела папки и сказала: «У нас нет времени читать все эти требования. У нас есть задание — написать программное обеспечение!» Они создали систему такой, какой она должна была быть по их мнению. Затем им пришлось создавать ее заново, опираясь на фактические требования, собранные предыдущей командой бизнес-аналитиков. Мораль истории такова: насколько вы ни были ограничены во времени, изучить хороший набор требований будет быстрее, чем создавать продукт дважды.

Иногда необходимо задокументировать определенную информацию в интересах самого разработчика. Например, если команда пишет ПО для организации, не задокументировавшей свои бизнес-правила, то программный код, работающий на основе этих правил или обеспечивающий их соблюдение, становится окончательным источником знаний бизнес-уровня. Разработчик, реализовавший код на основе этих бизнес-правил, становится экспертом в предметной области, к которому бизнес должен обращаться при изменении политики. Люди не должны извлекать знания бизнес-уровня из кода конкретного приложения, тратя время на еще один процесс реинжиниринга.

Разумный баланс



У документации есть свои достоинства и недостатки. Даже лучшие требования не могут заменить человеческое общение, но они, безусловно, оказывают существенную помощь. Запись какой-либо информации не гарантирует ее точности, полноты или неизменности. Однако наличие письменных документов увеличивает вероятность, что люди, получившие доступ к информации, придут к тому же пониманию и впоследствии смогут освежить свои знания. Документация должна быть актуальной, точной и доступной для тех, кто в ней нуждается. Если читатели не смогут легко найти искомое, то не имеет значения, насколько хороша документация.

Некоторые неправильно понимают философию гибкой разработки и не хотят писать документацию. В Манифесте Agile для разработки программного обеспечения говорится: «Работающий продукт важнее исчерпывающей документации» (Beck et al., 2001). Но в нем не говорится: «Нам не нужна эта чертова документация». Эксперт по гибкой разработке Майк Кон указывает на недостатки письменных документов, но советует: «Не рискуйте, отказываясь писать документацию» (Cohn, 2010):



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... Цель Agile-разработки — найти правильный баланс между документированием и обсуждением. В прошлом очень часто наблюдался чрезмерный перевес в сторону документирования.

Кон советует бизнес-аналитикам, руководителям проектов, владельцам продуктов и разработчикам уделять письменной документации разумное внимание. Записывать информацию нужно, выдерживая соответствующий (не обязательно минимальный) уровень детализации. Когда детали известны и необходима точность, их обязательно нужно фиксировать в документации. Это более практичный подход, чем попытка высечь в камне изменчивую или предварительную информацию, которую кто-то должен хранить в течение долгого времени. Кроме того, он более безопасный, так как не вынуждает полностью положиться на человеческую память.

С опытом вы поймете, что мир далеко не черно-белый. Почти в любой ситуации глупо выбирать крайнюю позицию. Две крайности — описывать каждую часть проектной информации в мельчайших подробностях или вообще не иметь письменной документации — одинаково глупы. Помня, что запись знаний обходится дешевле, чем повторное их обретение, вы сохраняете свободу выбора в отношении того, какую информацию записать.

Урок 8

Главное требование к разработке — налаженное и эффективное общение

Разработка программного обеспечения отчасти связана с вычислениями и отчасти — с общением. А разработка требований полностью основана на общении. В целом мы лучше разбираемся в технической стороне разработки ПО, чем в человеческой. Те члены команды, которые отвечают за соблюдение требований (я называю их бизнес-аналитиками, как бы на самом деле ни назывались их должности), находятся в центре сети общения, как показано на рис. 2.3. Они координируют обмен знаниями о требованиях между всеми участниками проекта.

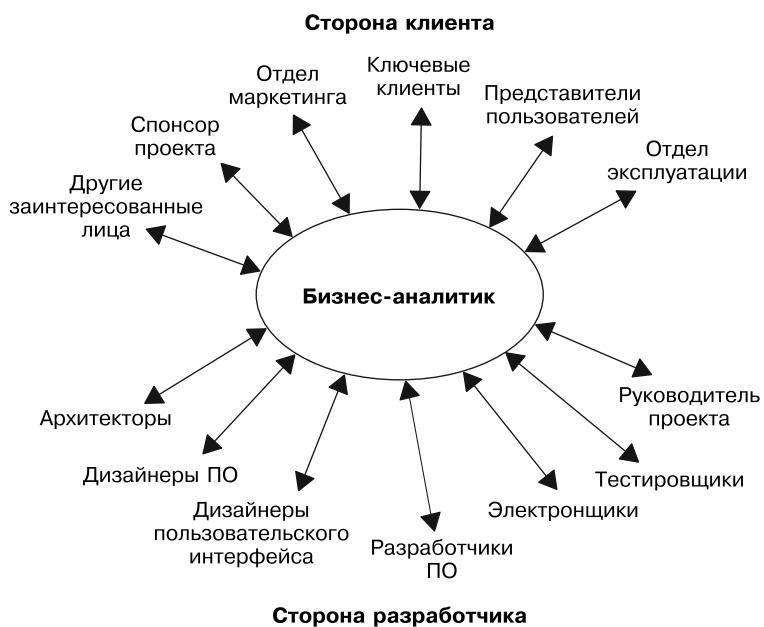


Рис. 2.3. Бизнес-аналитик координирует обмен информацией о требованиях между всеми участниками проекта

Все каналы общения на рис. 2.3 показаны двунаправленными стрелками. Участники, находящиеся на стороне клиента, только вносят требования в проект: спонсор проекта, отдел маркетинга, ключевые клиенты и представители пользователей. Участники, находящиеся на стороне разработчика: архитекторы, дизайнеры ПО и пользовательского интерфейса, разработчики и тестировщики, — пользуются результатами обработки требований. Если продукт содержит не только программные, но и аппаратные компоненты, то в проекте могут также участвовать инженеры-электронщики и инженеры-механики.

Бизнес-аналитик должен сообщать всем участникам информацию о требованиях, приоритетах, состоянии и изменениях. Каждый может участвовать в обсуждении требований, и такие разговоры очень полезны, поскольку участники видят проблемы с разных точек зрения. Кто-то неизбежно будет принимать множество решений, связанных с требованиями. И все участники должны вносить свои идеи и предложения, касающиеся требований, чтобы помочь команде достичь целостного понимания и поддерживать его.

Разные аудитории, разные потребности

У бизнес-аналитика непростая работа. Другие участники обсуждают добавляемые требования в основном устно. Бизнес-аналитик также может ссылаться на документы, например, описывающие бизнес-правила или содержащие информацию о похожих продуктах. Он должен оценить, классифицировать и записать всю эту информацию в соответствующей письменной форме. Я говорю «записать», поскольку эта информация должна быть передана стороне клиента для проверки и затем стороне разработки — для воплощения. Такая обширная аудитория включает специалистов с разным опытом и словарным запасом, и бизнес-аналитик должен тщательно продумать, как взаимодействовать с каждым из них. Люди, получающие знания о требованиях, различаются по некоторым параметрам:

- какую информацию они хотели бы получить;
- когда они хотели бы ее получить;
- насколько детальной должна быть информация;
- в каком виде они хотели бы ее получить;
- как она должна быть организована.

Любой, кто когда-либо пытался создать единое требование, которое бы сообщало всем участникам проекта все, что они должны знать, быстро понимал, что это невозможно. Как было показано в табл. 2.1 (см. выше), существует множество видов информации о требованиях. Бизнес-аналитик должен определить, как представить каждый вид на соответствующем уровне детализации и как организовать информацию для каждой аудитории.



Бизнес-аналитик должен узнать у различных участников, какая информация им нужна и в каком виде они хотят ее получить. Например, разработчикам и тестировщикам нужны подробности о каждом требовании, а спонсору проекта эти детали неинтересны. Те, кому достаточно общего описания, могут предпочесть получить информацию в графическом виде, чтобы не увязнуть в деталях. А для тех, кому нужны все подробности, оптимальным местом хранения может оказаться общий репозиторий.

Нелишним также будет использование стандартных шаблонов для определенных наборов информации, чтобы читатели знали, где найти то, что им нужно. Мне особенно пригодились шаблоны документов, описывающих видение и концепцию продукта с вариантами исполь-





зования и спецификациями требований к программному обеспечению (Wiegers, Beatty, 2013). Создатели документов должны общаться с их получателями и адаптировать стандартный шаблон под конкретные потребности. (См. урок 57 «Адаптируйте готовые шаблоны документов».)

Авторы документов сами выбирают словарный запас, уровень детализации и организационную схему. Но для некоторых читателей их выбор может оказаться не самым эффективным. Вот как один старший системный инженер описал проблему использования языка, отвечающего основным требованиям аудитории:

Требования в первую очередь адресованы разработчику, потому что именно его интерпретация определяет, каким будет продукт. Требование, написанное на языке бизнес-клиента, может быть насыщено терминами, незнакомыми обычному разработчику, из-за чего ему может быть трудно понять, что именно нужно клиенту. Требования часто пишутся на языке бизнес-клиентов, лишь бы удовлетворять букве контракта. Поэтому важно перевести их с языка бизнес-клиентов на язык, понятный разработчику.

Подумайте о создании словаря терминов, чтобы все, кто участвует в проекте, одинаково понимали соответствующие бизнес- и технические термины, аббревиатуры и сокращения. Словари можно повторно использовать в нескольких проектах в одной и той же предметной области и тем самым повышать согласованность.

Разработка качественного программного обеспечения невозможна без хорошо подобранных требований, оформленных в удобном для использования виде и доведенных до сведения всех, кто должен их знать.

Выбор методов представления

Самый очевидный способ представить информацию о требованиях — описать их на естественном языке. Однако любое такое представление (в виде документа, набора каталожных карточек или стикеров либо

файлов в репозитории) оказывается слишком громоздким для крупной системы. Читатели могут получить подробную информацию о требованиях, но им трудно составить общую картину и увидеть, как все эти детали сочетаются друг с другом. В естественном языке возможны двусмысленности и нечеткие формулировки, что оставляет слишком много пространства для воображения читателя. Тем не менее люди общаются с помощью естественного языка, поэтому такая форма представления требований вполне логична.

Требования можно записать с применением разных представлений. Одни команды оформляют их в виде сценариев использования и списков функциональных требований. Другие предпочитают пользовательские истории, описание функциональных возможностей и/или приемочные тесты. Но вообще выбор варианта оформления не имеет большого значения, если он обеспечивает точную и эффективную передачу информации.

Конечно, вы не ограничены использованием только естественного языка. Вы можете дополнять, а иногда и полностью заменять письменные требования различными представлениями: визуальными моделями, прототипами, макетами экранов, таблицами и математическими выражениями. Каждое представление характеризует часть того, что люди должны знать о требованиях, а путем объединения нескольких представлений можно достичь более глубокого понимания (Wiegers, 2006а). Существуют даже виды записи формальных требований, правильность которых можно доказать, но они используются довольно редко и в основном в критически важных системах.

Для тех, кому достаточно общего обзора, требования можно оформить в виде диаграмм. В свое время я прослушал замечательный курс по анализу и проектному моделированию, полностью изменивший мой подход к разработке программного обеспечения. Диаграммы позволяют иллюстрировать потоки процессов, отношения между данными, навигацию по пользовательскому интерфейсу, состояния системы и переходы между ними, логику принятия решений и многое другое. Я с удовольствием включил визуальное моделирование в свою практику разработки.

К моему разочарованию, я вскоре узнал, что ни одна диаграмма не в состоянии показать мне все, что я должен знать о программной системе (Davis, 1995). Вместо этого каждая модель отражает лишь часть знаний с определенной точки зрения. Бизнес-аналитикам нужно



выбирать подходящие модели с учетом того, какую информацию должна получать их аудитория.



Для ясности общения особенно важно иметь стандартные словари и обозначения. Мы не сможем работать вместе без общего понимания слов и символов. Однажды мой клиент попросил меня посмотреть модель, которую нарисовал один из его бизнес-аналитиков. В целом диаграмма была понятна, но меня смущало использование нестандартных обозначений стрелок. Я не знал, каково их значение (легенды не было) и что отличает их от стандартных стрелок, используемых в других местах.



Создатели методологий разработки программного обеспечения придумали множество стандартных обозначений для анализа диаграмм и моделей, в том числе:

- структурированный анализ (DeMarco, 1979);
- IDEF0 (Feldmann, 1998);
- унифицированный язык моделирования (Unified Modeling Language, UML) (Booch et al., 1999);
- язык моделирования требований (Requirements Modeling Language, RML) (Beatty, Chen, 2012).

Я настоятельно рекомендую использовать эти или другие стандартные модели. Не изобретайте собственных обозначений, если есть модели, позволяющие показать то же самое стандартными средствами. Иначе вам придется учить людей, просматривающих диаграммы, как читать ваши обозначения, и добавлять на диаграммы легенды, объясняющие назначение используемых символов. Страйтесь сохранить модели как можно более простыми — сосредоточьтесь на ясности передачи информации.

Мы можем поговорить?



Разработка качественного программного обеспечения невозможна без хорошо подобранных требований, полученных от нужных людей, оформленных в удобном для использования виде и доведенных до сведения всех, кто должен их знать. Эффективный бизнес-аналитик владеет многими формами общения: внимательно слушает, задает вопросы, переформулирует ответы, записывает, моделирует, представляет, помогает и считывает невербальные сигналы. Если вы

выступаете в роли бизнес-аналитика, то вам потребуются все эти навыки, а также опыт, позволяющий узнать, как и когда применять их и объединять всех участников проекта для достижения общей цели.

Урок 9

Качественность требований каждый определяет по-своему

Как известно, красота в глазах смотрящего. То же относится и к качественности. У требований к программному обеспечению есть своя аудитория: это разработчики, которые будут использовать их для выполнения своей работы, а также представители клиентов, которые получат или будут использовать продукт. Именно получатели, а не разработчики требований должны оценивать их качество.

Если кто-то обнаружит проблемы с моими требованиями, то это повод пересмотреть их. И неважно, насколько хорошими считал их я сам.

Я мог бы создать набор требований, кажущийся мне идеальным. В нем есть все, что нужно, и ничего лишнего, они логично структурированы, и все утверждения выглядят ясными и понятными. Но если кто-то обнаружит проблемы с моими требованиями, то это повод пересмотреть их, и неважно, насколько хорошими считал их я сам. Создатели (бизнес-аналитики) и получатели (архитекторы, дизайнеры, разработчики, тестировщики и др.) этих сводов знаний должны согласовать их содержание, формат, организацию, стиль и уровень детализации.



Многообразие получателей требований

Проблема бизнес-аналитика состоит в том, что существует довольно много людей, которым предназначена информация о требованиях, как описано в предыдущем уроке. Все эти люди имеют разные представления о качестве требований и о том, какие сведения те должны включать, поскольку используют информацию для разных целей. Они

имеют разное образование, различные взгляды, делают разные предположения и могут предпочитать разные средства коммуникации. Из-за такого разнообразия бизнес-аналитику сложно удовлетворить потребности каждого.

Лучший способ узнать, соответствуют ли ваши требования высокому качеству, — попросить людей, представляющих разные точки зрения, проанализировать их. (См. урок 48 «Стремитесь к тому, чтобы дефект нашли коллеги, а не покупатели».) Они помогут вам отыскать различные проблемы, если таковые имеются. В табл. 2.3 перечислен ряд проблем качества. Чтобы требования можно было считать качественными, люди должны ответить «да» на каждый вопрос. Такой способ формальной оценки, называемый *инспектированием*, особенно эффективен при выявлении определенных классов ошибок в требованиях (Wiegers, 2002а). В ходе проверки один участник описывает каждое требование своими словами. Другие участники могут сравнить эту интерпретацию со своим пониманием требования. Если интерпретации не совпадают, значит, имеет место двусмысленность.

Таблица 2.3. Те или иные проблемы качества, которые могут помочь отыскать разные читатели

Читатель требований	Некоторые проблемы качества
Спонсор проекта, отдел маркетинга, ключевые клиенты	<ul style="list-style-type: none"> Позволит ли решение, основанное на требованиях, достичь целей, поставленных ключевыми клиентами? Понятны ли риски и последствия для бизнеса, связанные с каждым требованием?
Представители пользователей	<ul style="list-style-type: none"> Понятно ли каждое требование? Достаточно ли точно каждое требование выражает потребность клиента? Будет ли решение, основанное на этом наборе требований, удовлетворять мои потребности? Все ли требования необходимы?
Руководитель проекта	<ul style="list-style-type: none"> Сможет ли команда разработать решение с учетом имеющихся ресурсов и в рамках существующих ограничений? Позволяет ли описание каждого требования оценить его сложность и влияние на проект?

Читатель требований	Некоторые проблемы качества
Бизнес-аналитик, владелец продукта, продукт-менеджер	<ul style="list-style-type: none"> • Каждое ли требование представляет ценность для клиента? • Являются ли требования четкими и однозначными? • Отсутствуют ли конфликты между требованиями?
Дизайнер, разработчик, инженер-электронщик	<ul style="list-style-type: none"> • Понятно ли каждое требование? • Содержат ли требования всю информацию, необходимую для разработки решения, и если нет, то указывают ли, где ее получить? • Возможно ли реализовать решение, основанное на требованиях, с технической точки зрения и с учетом имеющихся ресурсов и ограничений во времени?
Тестировщик	<ul style="list-style-type: none"> • Понятно ли каждое требование?
Другие заинтересованные стороны	<ul style="list-style-type: none"> • Соответствуют ли требования всем ожиданиям и ограничениям, накладываемым моей точкой зрения?

Контрольный список для оценки качественности требований

Ставя перед собой цель создать качественные требования, бизнес-аналитики должны стремиться к достижению следующих характеристик в своих результатах (Davis, 2005; Wiegers, Beatty, 2013).



- **Полнота.** Отсутствие упущений в требованиях. Каждое требование содержит всю информацию, необходимую читателю для выполнения его работы. Любые упущения отмечаются как подлежащие уточнению. Конечно, никто не сможет гарантировать, что выявил все требования. Но если вы намеренно пишете неполные требования, ожидая, что читатели запросят дополнительную информацию, когда она им понадобится, то убедитесь, что они знают об этом.
- **Непротиворечивость.** Решение, удовлетворяющее любому отдельно взятому требованию, должно быть совместимо с любыми другими требованиями. Уловить противоречия сложно. Порой очень трудно выявить несоответствия между требованиями разных типов,

например нарушение бизнес-правил со стороны функциональных требований или конфликт с требованиями пользователя.

- **Правильность.** В каждом требовании точно указывается потребность, озвученная пользователем или другим заинтересованным лицом. Эту характеристику могут оценить только соответствующие заинтересованные стороны.
- **Достижимость.** Разработчики смогут реализовать решение для удовлетворения данного требования в рамках известных технических, временных и ресурсных ограничений.
- **Необходимость.** Каждое требование описывает возможность, в которой действительно нуждаются те или иные заинтересованные стороны.
- **Приоритетность.** Требования классифицируются по их относительной важности и своевременности включения в продукт.
- **Отслеживаемость.** Каждому требованию присваивается уникальный идентификатор, чтобы его можно было связать с источником и далее с проектами, кодом, тестами и любыми другими элементами, созданными для удовлетворения этого требования. Знание источника дает дополнительный контекст и показывает, к кому можно обратиться за разъяснениями.
- **Однозначность.** Все читатели будут интерпретировать каждое требование совершенно одинаково. Если требование неоднозначно, то невозможно будет определить, является оно полным, правильным, выполнимым, необходимым или достоверным, поскольку вы не сможете точно сказать, что оно означает. Нельзя полностью избавиться от двусмыслинности естественного языка; тем не менее страйтесь избегать следующих слов и выражений: *лучший, и так далее, быстрый, гибкий, например, то есть, улучшенный, включая, максимизировать, необязательно, несколько, достаточно, поддержка и обычно* (Wiegers, Beatty 2013).
- **Достоверность.** Существует некий объективный, однозначный и эффективный способ определить, удовлетворяет ли решение требованиям. Наиболее распространенный способ проверить достоверность — тестирование, поэтому некоторые называют эту характеристику *тестируемостью*.



Невозможно создать идеальный набор требований. Но в этом и нет необходимости, если процесс разработки проекта подразумевает использование механизмов для быстрого выявления и исправления ошибок в требованиях до того, как команда их реализует. Получение отзывов

от многочисленных читателей требований поможет вам избежать чрезмерных затрат на доработку.

Урок 10

Требования должны быть достаточно хорошими,
чтобы разработка могла продолжаться
с приемлемым уровнем риска

Как я уже говорил, невозможно создать идеальный набор требований. Некоторые требования могут быть неполными, неправильными, ненужными, невыполнимыми, двусмысленными или вообще отсутствовать. Иногда требования противоречат друг другу. Но в любом случае программное обеспечение должно создаваться на основе требований.

Соответственно, ваша задача — разработать требования, достаточно хорошие для того, чтобы можно было перейти к следующему этапу разработки. Это вопрос риска. Вы должны приложить все силы, чтобы разработать требования и снизить риск чрезмерных незапланированных переделок из-за проблем с ними.



К сожалению, нет явного индикатора, позволяющего судить, достаточно ли хороши ваши требования. Бизнес-аналитику сложно судить, были ли выявлены все соответствующие требования и достаточно ли точно они изложены. И все же кто-то должен решить, в какой момент требования к продукту смогут обеспечить достаточно прочную основу для его разработки. Системные архитекторы, проектировщики и разработчики могут помочь принять это решение.

Ваша задача — разработать требования, достаточно хорошие для того, чтобы можно было перейти к следующему этапу разработки.

Уровень детализации

Понятие «достаточно хорошие» включает как количество представленной информации, так и ее качество. В минимальном наборе идеально написанных требований может не быть подробной информации,



необходимой разработчикам и тестировщикам, однако обширный перечень плохо написанных и неточных требований бесполезен. Эксперт по требованиям Алан Дэвис (Alan Davis) прекрасно сформулировал цель спецификации требований: «Определить желаемое поведение системы достаточно подробно, чтобы разработчики системы, маркетологи, клиенты, пользователи и руководство интерпретировали его более или менее одинаково» (Davis, 2005). Ключевые слова — *достаточно подробно*. Полноту требований можно представить в трех измерениях: типы включенной информации, широта знаний и глубина детализации каждого элемента.

- **Типы информации.** Участники проекта обычно сосредоточиваются на функциональности, необходимой пользователям для достижения их целей, но набор требований должен включать не только описание функциональности. Разработчики должны знать об источниках данных, ограничениях, бизнес-правилах, требованиях к качеству и внешнему интерфейсу. Простого набора функциональных требований или пользовательских историй недостаточно.
- **Широта знаний.** Это измерение определяет объем информации, содержащейся в спецификации. В нее входят все требования пользователей или только высокоприоритетные? Учитываются ли все атрибуты качества или только те, которые имеют первостепенное значение? Может ли читатель быть уверенным, что перед ним полный спектр ожиданий, или ему нужно восполнить пробелы? Если набор требований неполный, то все ли читатели увидят одни и те же пробелы? Если подразумеваемые и предполагаемые требования нигде не фиксируются, то высока вероятность, что онистанутся незамеченными. (См. урок 13 «Две распространенные практики выявления требований — телепатия и ясновидение. Но они не работают».)
- **Глубина детализации.** Третье измерение касается степени детализации и точности каждого требования. Учитывают ли требования возможные исключения (ошибки) и определяют ли, как система должна их обрабатывать? Или они касаются только нормального поведения? Если спецификация определяет нефункциональное требование, такое как возможность установки, то охватывает ли она также удаление, повторную установку, восстановление и установку обновлений и исправлений? Любые требования, и функциональные, и нефункциональные, должны быть достаточно полными и точными, чтобы их можно было проверить в реализованном решении.



Достаточно — это сколько?

Не существует единственно правильного ответа на вопрос, сколько информации достаточно в той или иной ситуации. Однако везде, где существуют пробелы в знаниях, кто-то должен будет их восполнить. Чтобы решить, достаточно ли хороши требования, нужно определить необходимое количество деталей, а также того, кто их получит и когда. Свод правил по бизнес-анализу от ПВА содержит определение требований, в котором, в частности, говорится: «Они обеспечивают соответствующий уровень детализации, позволяющий разработать и внедрить решение» (ПВА, 2015). Слово «соответствующий» представляет собой суждение, которое будет по-разному интерпретироваться разными людьми.

Многие команды Agile-разработки ПО не оформляют все детали требований в письменном виде, но это не значит, что эти детали не нужны разработчикам и тестировщикам. Как мы видели в уроке 6, они нужны. Если письменная информация недоступна во время реализации, то должна быть возможность получить ее из нужного источника. В противном случае члены команды должны будут сами восполнить пробелы, что может быть отмечено заказчиком как недостаток. А это будет означать, что требования не полностью готовы для реализации.



Урок 11 Люди не просто так собирают требования

Люди часто говорят о сборе требований к программному проекту, но это создает неверное впечатление. Слово «сбор» предполагает, что требования уже лежат где-то в готовом виде и только и ждут, чтобы их собрали. Когда я слышу, как кто-то говорит «сбор требований», в моем воображении всплывает картинка сбора цветов. Боюсь, что все не так просто.

Сбор и выявление

Требования редко существуют в сознании пользователей в полностью сформированном виде, готовом для передачи бизнес-аналитику или группе разработчиков. Создание набора требований определенно подразумевает этап сбора некой информации, но также включает открытия и изобретательность. Термин «выявление требований» (*requirements elicitation*)

elicitation) точнее описывает, как разработчики программного обеспечения сотрудничают с заинтересованными сторонами, чтобы понять, как те работают сейчас, и определить, какие возможности должна предоставлять будущая программная система. Эксперты по требованиям Сюзанна Робертсон и Джеймс Робертсон в своей книге *Mastering the Requirements Process* (2013) называют этот процесс поиском требований:

Для описания деятельности по исследованию бизнеса подойдет слово «прочесывание». Оно напоминает о том, чем мы здесь занимаемся: ловлей рыбы. Мы не просто забрасываем удочку, надеясь хоть что-то поймать, а методично процеживаем акваторию бизнеса сетями, чтобы обнаружить все возможные потребности.

Согласно словарю *The American Heritage Dictionary of the English Language* (2020), под словом *elicitation* подразумевается выманивание, вытягивание или провокация. Слова «выманивание» и «вытягивание» точнее описывают процесс, чем слова «выявление» или «сбор». (Бизнес-аналитики не пытаются спровоцировать заинтересованные стороны, с которыми они работают, хотя иногда это случается непреднамеренно.) Основная работа бизнес-аналитика во время сбора информации состоит в том, чтобы задавать правильные вопросы и этими вопросами стимулировать обсуждение и побуждать заинтересованные стороны выходить за рамки поверхностных и очевидных решений.

Когда я слышу, как кто-то говорит «сбор требований», в моем воображении всплывает картина сбора цветов. На самом деле все не так просто.



Самые бесполезные вопросы, которые не следует задавать при изучении требований: «Чего вы хотите?» и «Каковы ваши требования?» Такие расплывчатые вопросы вызывают поток множества случайных — но важных — сведений, смешанных с посторонней информацией и приправленных невысказанными предположениями. Бизнес-аналитик не просто записывает все, что ему говорят заинтересованные стороны. Он выявляет детали, помогая участникам структурировать имеющиеся знания.

Бизнес-аналитик должен проанализировать и систематизировать собранную информацию, очистить ее от ненужного шума и затем

представить разработчикам и другим участникам проекта в удобной для них форме.

Когда выявлять требования

Как мы видели, обсуждая урок 5, выявление — это итеративный и последовательный процесс с циклами уточнения, прояснения и корректировки. Обсуждение может перейти от нечетких, общих концепций к деталям или начаться с определенных функциональных фрагментов, которые бизнес-аналитик затем должен скомпоновать на более высоком уровне абстракции. Информация из одного источника может противоречить информации из другого. Иногда бизнес-аналитик получает новые данные, заставляющие его вернуться к тому, что, по мнению команды, уже окончательно обсудили. Подобные возвраты могут расстраивать участников («Мы же уже решили этот вопрос»), но такова природа нелинейного человеческого общения и исследования. Выявление требований напоминает снятие шелухи с луковицы, в процессе которого открывается внутренняя часть; при этом чем больше шелухи вы снимаете, тем более крупной кажется луковица.



В жизненном цикле гипотетического чистого водопадного стиля разработки выявление требований выполняется только в начале проекта. В идеале бизнес-аналитик мог бы собрать все требования заранее и они оставались бы неизменными на протяжении всего периода создания системы. Проекты, к которым применим такой подход, встречаются, но их участники должны выделить значительное время для этапа выявления требований. Даже традиционные проектные группы знают, что требования, написанные на ранней стадии, необходимо пересматривать и уточнять в ходе проекта.

В Agile-проектах требования намеренно рассматриваются небольшими порциями и ожидается, что набор требований будет расширяться в процессе разработки. Каждая итерация разработки включает действия по выявлению новых требований. Проект начинается с изучения лишь небольшой части требований, и на первом этапе не ожидается, что все участники достигнут полного понимания всех деталей. Вместо этого команда постепенно накапливает знания, чтобы на ранних итерациях распределить требования по степени важности. Затем в ходе каждой итерации команда уточняет выделенные требования (обычно представленные в виде пользовательских историй и приемочных тестов) до уровня детализации, необходимого разработчикам и тестировщикам.



Контекст выявления

Документ о видении и масштабах проекта или устав проекта подготавливает почву для выявления требований. Он устанавливает бизнес-цели проекта, сферу охвата (то есть то, что явно включено) и ограничения (то есть то, что явно исключается). Чтобы начать процесс выявления, определите заинтересованные стороны, которые могут послужить источниками ценной информации. Эти заинтересованные стороны могли бы быть наделены правом вето на требования («Вы не можете этого сделать») или правом добавления новых требований («Вы также должны сделать это»). Вам следует тесно взаимодействовать с этими заинтересованными сторонами, чтобы понять их задачи, потребности и проблемы, а также их ожидания от новой или модифицированной системы.

Если вы бизнес-аналитик, то спланируйте свою стратегию сбора информации, прежде чем погрузиться в нее. Выбранные вами методы взаимодействия будут зависеть от доступности заинтересованных сторон, их местонахождения, от того, какие способы коллективного или индивидуального обсуждения являются наиболее подходящими и сколько времени стороны могут выделить на эти обсуждения. Планируйте каждую встречу, чтобы гарантировать получение информации, необходимой команде разработчиков. Возможно, вам придется скорректировать методы взаимодействия в зависимости от уровня конструктивности общения с заинтересованными сторонами.

Методы выявления

Необходимые знания о бизнесе или проекте можно найти во многих местах и с помощью многих способов. Ниже перечислены несколько методов выявления, которые могут пригодиться в большинстве проектов (Davis, 2005; Robertson, Robertson, 2013; Wiegers, Beatty, 2013).

Собеседования

Для собеседований с заинтересованными сторонами один на один характерны эффективность и целенаправленность. Они позволяют углубиться в детали, не отвлекаясь на второстепенные уточнения. Однако им не хватает синергетического взаимодействия, которое часто способствует появлению новых идей в групповых дискуссиях. И для индивидуального, и для группового собеседования бизнес-аналитик дол-

жен подготовить список исследуемых тем и вопросов, которые необходимо задать (Podeswa, 2009).

Групповые семинары

Проведение семинаров, на которых бизнес-аналитик встречается с некоторыми представителями пользователей и другими заинтересованными сторонами, — обычный способ выявления информации. Семинары обычно лучше подходят для изучения требований пользователей и помогают понять, какие задачи пользователи хотели бы решать с помощью системы. Однако любые коллективные дискуссии склонны уходить в сторону, за запланированные рамки встречи. Группе легко сбиться с пути, увязнув в деталях, в то время как следовало бы обсуждать вопросы на более общем уровне. Опытный организатор удерживает участников в теме и обеспечивает получение полезной информации в ходе семинара. (См. урок 14 «Большая группа людей не способна организованно покинуть горячую комнату, не говоря уже о том, чтобы сформулировать какое-то требование».)



Наблюдение

Наблюдение за работой пользователей в привычной обстановке позволяет получить информацию, поделиться которой им бы и не пришло в голову, если бы бизнес-аналитик просто задавал вопросы об их работе. Наблюдательный бизнес-аналитик может подметить проблемы и узкие места, требующие решения в новой системе, чтобы сделать бизнес-процессы более эффективными. Пользователи часто компенсируют недостатки программных систем с помощью обходных путей, поэтому наблюдение может помочь выявить улучшения, которые желательно внести в замещающую систему. Дизайнеры пользовательского интерфейса, играющие роль бизнес-аналитиков в некоторых проектах, тоже часто считают полезным понаблюдать за пользователями во время их работы.

Анализ документов

Документация к существующим системам, продуктам и бизнес-процессам может оказаться богатым источником потенциальных требований. Изучение такой документации помогает бизнес-аналитику быстро освоиться в новой для себя прикладной области. Документы предоставляют информацию о действующих бизнес-правилах: корпоративных политиках, правительственные постановлениях и отраслевых стандар-

так. Сюзанна и Джеймс Робертсоны в своей книге *Mastering the Requirements Process* назвали процесс реинжиниринга новых требований из существующих текстов археологией документов. Но имейте в виду, что информация, полученная из исторических источников, обязательно должна проверяться на актуальность.

Опросы

К собеседованиям и семинарам можно привлекать только ограниченное количество участников. Опросы же позволяют узнать мнение о текущих продуктах у большей части пользователей. Онлайн-опросы в отношении коммерческих продуктов особенно полезны, когда у команды разработчиков нет прямого доступа к представителям пользователей. Создавать опросы, помогающие выявлять искомую информацию и побуждающие пользователей заполнять их, — целое искусство. Опросы должны содержать минимально возможное количество вопросов, позволяющее вам узнать то, что нужно.

Онлайн-опросы часто составляются так, что пользователь вынужден ответить на каждый вопрос, чтобы пройти их. Бывает, я отказываюсь от опросов, когда замечаю, что они получаются слишком длинными. Многие пользователи готовы поделиться своим мнением, но не хотят тратить много времени на изучение длинных списков вопросов.

Вики

Вики и другие инструменты для совместной работы позволяют собирать информацию и идеи среди более широкого круга людей, чем это позволяют семинары. Сообщение одного человека может вызвать у других одобрение, несогласие или желание дополнить информацию. Недостаток такого подхода в том, что бизнес-аналитику приходится фильтровать обсуждение, чтобы собрать действительно ценную информацию.

Прототипы

Исходя из абстрактных обсуждений и списков требований, людям трудно представить, каким может быть предлагаемое решение. Прототип делает требования более осозаемыми. Даже простые эскизы пользовательского интерфейса могут помочь участникам семинара получить наглядное представление. Но создавать прототипы в начале исследования требований чересчур рискованно, поскольку люди могут преждевременно зациклиться на конкретном (и, возможно, неидеальном) решении.

Закладка фундамента

Этап выявления является основным при разработке требований к проекту. Если вы не имеете прочного фундамента знаний о требованиях, полученных путем эффективного выявления, проект будет оставаться в шатком положении.

Урок 12

Выявление требований должно помочь разработчикам услышать голос клиента

В ходе одного из самых продуктивных периодов моей карьеры разработчика программного обеспечения я создал несколько приложений для ученого по имени Шон, работавшего в исследовательской лаборатории Kodak. Шон был единственным пользователем, а я — командой разработчиков. В одиночку я сделал все, что необходимо: сформулировал требования, спроектировал пользовательский интерфейс, реализовал код с тестами и написал документацию. Одним из приложений был сложный инструмент для работы с электронными таблицами, который позволял Шону моделировать результаты получения фотоснимков на основе многочисленных параметров, определяющих характеристики камеры и фотопленки. Другим приложением была программа, работавшая на большой ЭВМ и анализировавшая экспериментальные данные Шона.



Нас с Шоном разделяло несколько метров. И я был очень продуктивен, поскольку мог общаться с ним часто, неформально и в любой момент. Я мог показать ему, что делаю, получить ответы на свои вопросы и выслушать его идеи об организации пользовательского интерфейса. Эта близость и тот факт, что Шон был единственным пользователем проекта, позволили нам сотрудничать короткими и быстрыми итерациями. Нам не нужны были письменные требования, так как необходимые детали я мог быстро уточнить у него.

У нас с Шоном сформировалась идеальная среда для разработки программного обеспечения: один разработчик и один заказчик, сидящие рядом. Это редкость. У большинства проектов много заказчиков, несколько классов пользователей, множество источников требований и большое количество лиц, принимающих решения. Они реализуются командами разработчиков, насчитывающими от нескольких человек, собранных в одном месте, до сотен, разбросанных по всему миру. Этим

гораздо более сложным проектам нужны другие способы взаимодействия, чтобы разработчики могли услышать голос клиента, выявить требования, установить приоритеты, сообщить об изменениях и принять решения.

Способы взаимодействий



Создавая программное обеспечение не для себя, вы всегда будете сталкиваться с препятствиями, которые мешают клиентам, имеющим потребности, общаться с разработчиками, создающими решения. Каждая команда должна создать эффективные каналы общения между двумя сообществами на самых ранних этапах работы над проектом. Возможные варианты зависят от количества задействованных участников, от того, кто они и где находятся, насколько хорошо сообщества понимают друг друга, а также от знаний и умений команды разработчиков.

Идентифицировав группы своих пользователей, определите, кто из них будет представлять каждую группу.

На рис. 2.4 показано несколько моделей общения, помогающих разработчикам услышать голос клиента. Моя ситуация с Шоном соответствует модели А, прямой связи между пользователями и разработчиками. При использовании этой модели потребность в создании подробных письменных требований крайне мала, как и вероятность недопонимания — при условии, что разработчик и пользователь понимают терминологию друг друга. Однако чаще общение будет протекать через посредников.

Когда пользователей слишком много и у всех разные потребности, они не могут напрямую общаться с разработчиками. Это верный путь к хаосу, поскольку разработчики могут утонуть в разнообразии входной информации, не зная, какие источники являются авторитетными. Разработчикам трудно фильтровать противоречивые данные, поступающие из разных источников. Чтобы справиться с этим разнообразием, многие из моих клиентов, которых я консультировал, успешно использовали модель, обозначенную на рис. 2.4 буквой Б.

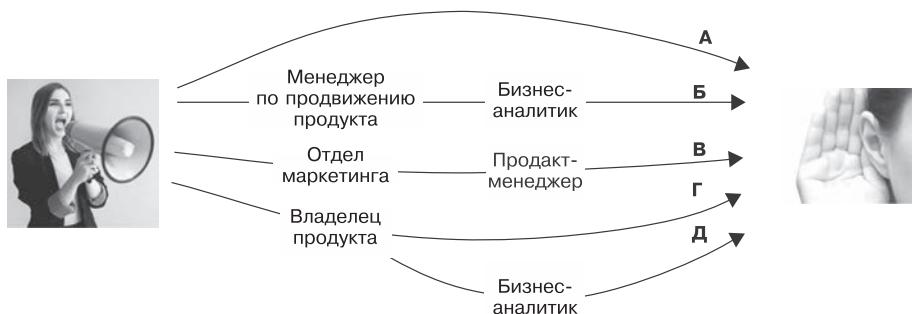


Рис. 2.4. Есть множество способов организовать взаимодействия, которые помогут разработчикам услышать голос клиента

При анализе заинтересованных сторон обычно выявляют несколько групп пользователей, потребности которых могут существенно различаться. Представители разных групп пользователей могут нуждаться в разных возможностях продукта, решать разные задачи, различаться частотой использования или иметь другие особенности. Идентифицировав группы своих пользователей, определите, кто из них будет представлять каждую группу.



Менеджер по продвижению продукта

В модели Б функции основного канала передачи информации о требованиях выполняют один или несколько ключевых представителей пользователей, называемых *менеджерами по продвижению продукта* (product champions), которые сотрудничают с одним или несколькими бизнес-аналитиками (Wiegers, Beatty, 2013). Менеджеры по продвижению продукта знают предметную область и понимают бизнес-цели проекта. Они взаимодействуют со своими коллегами, собирают их требования и отзывы, а также информируют других пользователей о ходе выполнения проекта. Бизнес-аналитик помогает преодолеть разрыв в общении между менеджерами по продвижению продукта и командой разработчиков.

Обратите внимание, что связь работает и в обратном направлении, противоположном указанному стрелкой на рис. 2.4 (см. выше). Если у разработчиков или других участников процесса возникают вопросы или им требуются разъяснения, то они обращаются к источнику тре-

бования для принятия решения. Чтобы разработчики могли быстро получить нужные ответы, полезно записывать, от кого поступило каждое требование.

Другие пути передачи требований

Компании, создающие коммерческие продукты, часто используют канал связи, обозначенный на рис. 2.4 буквой В. Отдел маркетинга оценивает потребности рынка и потенциал продаж нового или улучшенного продукта. Он может работать с продакт-менеджером, несущим основную ответственность за определение характеристик продукта, которые приведут к успеху в бизнесе. В разных организациях по-разному распределяют ответственность за определение продукта между отделом маркетинга и продакт-менеджером.

Продакт-менеджер выполняет функции бизнес-аналитика в ИТ-проекте. Вот краткое описание его роли (280 Group, 2021):

Продакт-менеджер отвечает за вывод на рынок специализированного продукта, отвечающего потребностям рынка и представляющего реальную ценность для бизнеса. Ключевая функция продакт-менеджера — обеспечение соответствия продукта общей стратегии и целям компании.

Проекты, практикующие методы Agile-разработки, особенно Scrum, часто используют путь Г, показанный на рис. 2.4. Владелец продукта определяет видение и цель продукта, а также создает и сообщает список требований (Cohn, 2010). Кроме того, владелец составляет перспективный план развития продукта — от концепции через ранние выпуски к зрелой версии, представляющей ценность для потребителя (McGreal, Jocham, 2018). В этом случае владелец продукта озвучивает запросы клиента.

Ответственные владельцы продукта, пока не являющиеся экспертами во всех областях, будут обращаться за советом к менеджерам по продвижению, которых я упоминал выше. В Scrum владелец продукта — отдельное лицо. Он несет единоличную ответственность за управление требованиями к продукту, даже если какую-то часть работы делегирует другим людям (Schwaber, Sutherland, 2020). Владелец продукта также взаимодействует с менеджерами по маркетингу и бизнесу и учитывает их мнения, определяя приоритеты требований.

Как видите, владелец продукта выполняет большую часть функций, которые мог бы выполнять бизнес-аналитик в IT-проекте, и даже больше. Тем не менее некоторые Agile-команды признают ценность наличия в команде квалифицированного бизнес-аналитика, который может взаимодействовать с владельцем продукта.

Когда есть и аналитик, и владелец, они могут сотрудничать различными способами. Бизнес-аналитик часто действует как помощник или заместитель владельца продукта. Владелец продукта может deleгировать бизнес-аналитику часть ответственности, например работу с определенными группами пользователей. И в таком случае уже бизнес-аналитик отвечает за все, что касается этой области, кроме расстановки приоритетов — она остается прерогативой владельца продукта.

Иногда владелец в большей степени ориентирован на продукт и рынок, тогда как бизнес-аналитик — на технические аспекты, формируя требования к решениям на основе требований пользователей (Datt, 2020a). Эта модель обозначена на рис. 2.4 буквой Д.

В других случаях верно обратное. Характер сотрудничества двух этих людей сводится к тому, что, по мнению владельца продукта, бизнес-аналитик может принести большую пользу проекту.

Преодоление разрыва

Все проекты должны связывать пользователей продукта с его создателями через бизнес-аналитика, инженера по требованиям, продакт-менеджера, владельца продукта или даже самих разработчиков, работающих с требованиями. Название должности не столь важно, как наличие роли и четкое определение ее обязанностей и полномочий. Лица, выполняющие эту функцию, должны обладать знаниями, навыками, опытом и личными качествами, необходимыми для работы как с клиентами, так и с разработчиками. Они должны установить с обоими сообществами отношения, основанные на взаимном доверии и уважении.



Эффективная разработка требований гарантирует, что разработчики будут четко и ясно слышать голос клиентов. Эти связи определяют, достигнет ли проект грандиозного успеха или просто не дойдет до финиша.

Урок 13

**Две распространенные практики выявления требований — телепатия и ясновидение.
Но они не работают**

Согласно словарю *The American Heritage Dictionary of the English Language* (2020), телепатия — это «предполагаемый процесс общения с помощью средств, отличных от органов чувств, например путем прямого обмена мыслями». Ясновидение — это «предполагаемая способность видеть объекты или события, которые не могут быть восприняты органами чувств». Эти два навыка, безусловно, значительно облегчили бы разработку программного обеспечения — будь они реальными. Телепатии и ясновидения не существует, но, по всей видимости, в некоторых проектах они являются технической основой.

Угадай это требование!

Иногда люди считают требования слишком очевидными, чтобы их озвучивать. Некоторые пользователи боятся показаться оскорбительно снисходительными по отношению к бизнес-аналитику, определяя то, что, по их мнению, он уже знает. Я предпочел бы услышать что-то, о чем уже знаю, во второй или третий раз (и укрепиться в убеждении, что я правильно все понимаю), чем заставить кого-то предположить, что я уже владею нужной информацией, когда на самом деле это не так.

Некоторые занятые пользователи не хотят тратить время на обсуждение требований с бизнес-аналитиком, владельцем продукта или разработчиком. Они рассуждают так: «Вы уже должны знать, что мне нужно. Позвоните мне, когда закончите». Такое отношение предполагает, что бизнес-аналитик владеет телепатией и ясновидением.



Предполагаемые и подразумеваемые требования — две большие области риска. *Предполагаемые требования* — это требования, которые предполагаются, но не озвучиваются. *Подразумеваемые требования* происходят из существования других требований, но также не указываются явно. Неразумно ожидать, что бизнес-аналитик обладает способностью читать мысли или заглядывать за горизонт, чтобы получать эти скрытые знания. Как мы уже видели, невозможно получить исчерпывающий набор требований, но участники проекта должны решить, что можно спокойно оставить недосказанным.

Старайтесь выражаться ясно

Я предпочитаю открыто сообщать об известных ожиданиях и не надеяться, что кто-то догадается, о чем я думаю. Очень хорошо, когда все заинтересованные стороны проекта имеют единый взгляд на задачу, чтобы создать правильный продукт без подробного описания требований. Чем больше люди работают вместе и чем больше команда разработчиков знает о предметной области, тем легче им достичь такого единения разумов. Но я придерживаюсь философии, что если требования не описывают конкретную возможность или характеристику, то никто не должен ожидать, что она будет реализована в продукте.



Бизнес-аналитик должен постараться раскрыть и подтвердить невысказанные предположения, которые иногда могут оказываться ошибочными или устаревшими.

Иногда мы выражаем требования неформально, предполагая, что у читателя есть «фильтр восприятия», подобный нашему, но люди могут интерпретировать одни и те же утверждения по-разному. Эта двусмысленность приводит к несоответствию ожиданий и неприятным сюрпризам. Вы можете исходить из другого набора предположений, чем я. *Предположение* — это утверждение, которое мы считаем истинным, не зная точно, что оно истинно. Бизнес-аналитик должен постараться раскрыть и подтвердить невысказанные предположения, которые иногда могут оказываться ошибочными или устаревшими.

Риск недопонимания возрастает, когда система реализуется сторонними организациями. Однажды я просматривал документ с требованиями к проекту, разработку которого планировал передать на аутсорсинг. Документ содержал множество требований, начинавшихся со слов: «Система должна поддерживать...» Я спросил автора документа, как разработчики компании-подрядчика узнают, какую именно функциональность подразумевает слово «поддерживать» в каждом случае. Подумав немного, она дала правильный ответ: «Думаю, никак». Она мудро решила уточнить, что именно имеется в виду под *поддержкой*, чтобы устраниТЬ двусмысленность. Это намного лучше, чем полагаться на телепатию или ясновидение.



Приведу пример подразумеваемого требования. Вы предлагаете предусмотреть возможность отмены некоторых операций, выполняемых в приложении. Разработчик реализует такую возможность, а вы ее тестируете. Она работает нормально. Но затем вы спрашиваете разработчика, как выполнить повтор операции.

«Но вы не просили реализовать возможность повтора», — отвечает разработчик.

«Я думал, что реализация функции отмены также подразумевает возможность повторного выполнения. Не могли бы вы добавить ее?» — просите вы. Разработчик добавляет функцию повтора, и она работает. Но потом вы удивляйтесь, почему повтор применяется только к последней отмененной операции. Это приводит к дальнейшему обсуждению: сколько последних отмененных операций должна поддерживать функция повтора? Вы хотите иметь возможность перейти к любой точке в последовательности отмен и повторить все отмененные действия, начиная с этой точки? Когда очищать очередь истории отмен? И так далее.

Если разработчики и пользователи находятся в тесном контакте, то могут начать с простого требования отмены и договориться, как именно должна вести себя функция отмены/повтора. В этом случае вам не потребуется несколько итераций, чтобы заказчик получил то, что хотел. Однако если вы передаете разработку на аутсорсинг, то о подобных вещах лучше подумать заранее и включить все особенности в требования. В противном случае не удивляйтесь, если разработчик интерпретирует скучно описанные требования совсем не так, как ожидает заказчик.

Подрядчик может даже обнаружить эту подразумеваемую функциональность в описании, но будет основываться только на оригинальном требовании, ожидая, что вы предоставите дополнительную информацию. В таком случае он может запросить больше денег и времени, чтобы реализовать «расширенные» требования.

Телепатия не срабатывает

Невозможно проработать все нюансы функциональных возможностей исключительно путем размышлений и обсуждений. Иногда только циклы разработки или прототипирования позволяют пользователям понять, что им нужно. Однако предполагаемые требования и итоговый

выбор дизайна программного обеспечения могут привести к дорогостоящей доработке. Недавно я прочитал о неудачном дизайнерском решении, принятом инженерами в отношении ручки управления истребителя F-16 Fighting Falcon (Aleshire, 2004):

Первоначально инженеры жестко закрепили ручку управления, сделав ее неподвижной, поскольку компьютер мог считывать давление руки пилота на ручку так же легко, как и фактическое движение ручки. Но пилоты возненавидели это решение. Они хотели, чтобы ручка двигалась, так как это давало им чувство управления машиной.

Ничто не заменит получение информации о требованиях и предлагаемых решениях от людей, которые будут использовать продукт. И нет никакого оправдания нежеланию фиксировать полученные знания в письменном виде. Только с их помощью можно гарантировать, что разработчики продукта смогут удовлетворить потребности клиентов.

Урок 14

Большая группа людей не способна организованно покинуть горящую комнату, не говоря уже о том, чтобы сформулировать какое-то требование

Одно время я работал ведущим бизнес-аналитиком в проекте информационной системы среднего размера. Два других бизнес-аналитика и я общались с представителями разных групп пользователей, пытаясь понять их требования. Однажды моя коллега Линнетт позвонила мне и сообщила, что на ее первом семинаре по выявлению информации было озвучено гораздо меньше вопросов, чем планировалось. Разочарованные участники были обеспокоены тем, сколько времени займет весь процесс. Линнетт попросила совета.



Я спросил ее, сколько человек было на семинаре. «Двенадцать», — ответила она. В этом и заключалась ее проблема. Большой группе людей трудно договариваться и принимать решения. Участники легко отвлекаются на посторонние разговоры. В большой группе всегда найдется несколько человек, которым будет что сказать по теме, что приведет к более длительному, но не всегда плодотворному обсуждению. Легко втянуться в дискуссию на излюбленную тему одного участника

и погрязнуть в деталях, которые могут не способствовать достижению цели дня. Разногласия могут перерасти в продолжительные споры. Одни участники могут доминировать в обсуждении, а другие — полностью отключиться.

Я предложил Линнетт сократить размер группы вдвое. Не нужно было приглашать шестерых представителей пользователей, достаточно было двух или трех. Некоторые люди, присутствовавшие на семинаре в качестве наблюдателей или защитников своих интересов, не вносили ничего полезного в исследование требований. Я рекомендую привлекать к обсуждению людей с опытом разработки и тестирования программного обеспечения, так как они имеют представление о выполнимости и возможности проверки предлагаемых требований. В данном случае опыт Линнетт позволял ей самой представить эти точки зрения. Линнетт сократила размер группы для последующих семинаров, и все участники наполнились оптимизмом, поскольку продвижение вперед ускорилось.

Концентрируйте внимание!

Четыре человека могут вести продуктивную дискуссию, не отвлекаясь на посторонние темы. Как показано на рис. 2.5, парных взаимодействий в группе из четырех человек совсем мало. Однако их количество растет, как снежный ком, по мере увеличения численности группы. На рис. 2.6 видно, сколько парных взаимодействий будет в группе из десяти человек. Неудивительно, что некоторые участники скатываются в личные беседы, особенно если их не интересует текущая тема. Идея уменьшать численность групп, чтобы добиться быстрого прогресса, применима к семинарам по выявлению требований, коллегиальным проверкам и аналогичным коллективным мероприятиям.

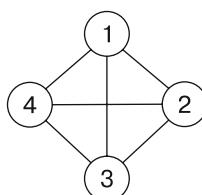


Рис. 2.5. В группе из четырех человек создаются два парных взаимодействия

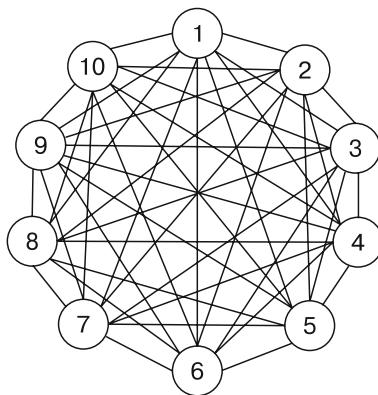


Рис. 2.6. В большой группе количество парных взаимодействий резко возрастает и появляется масса возможностей для отвлеченных споров

Мы обнаружили, что, следуя стратегии ограничения численности групп, можно проводить отдельные семинары для представителей разных классов пользователей. Требования разных классов часто различаются. Если собрать всех представителей заинтересованных сторон на одном семинаре, то любая возникающая тема будет интересна только части группы. Остальные участники могут заскучать и посчитать, что зря тратят время. Проведение отдельных семинаров с разными группами пользователей помогает повысить заинтересованность всех участников. Однако разумное объединение членов разных групп пользователей может создать эффект синергии и помочь выявить связи, вопросы и инновации, которые ни одна из групп не могла бы придумать независимо друг от друга.

Координатор во спасение

Удержание обсуждения в нужном русле в большой группе требует умелой координации. Иногда она осуществляется специально: кто-то берет на себя инициативу, чтобы встать у доски с маркером в руке и навести порядок. В роли такого координатора может выступить бизнес-аналитик, или же группа может привлечь беспристрастного внешнего координатора. Он должен заранее ознакомиться с целями и повесткой дня для каждого семинара. Ограничение по времени способствует более быстрому обсуждению, и группа не успеет запутаться в одной теме и не оставит без внимания другие из-за нехватки времени.

Координатор может решить при необходимости продолжить обсуждение дольше запланированного, если это приведет к получению дополнительной ценности. Кроме того, он решает, когда уместно углубиться в детали, а когда лучше перейти к следующей теме. Хороший координатор заметит, когда обсуждение требований сменяется поиском решений, и вернет группу в нужное русло. В книге Эллен Готтесдинер *Requirements by Collaboration* (2002) содержится подробное руководство по планированию и проведению семинаров, посвященных требованиям.

Фокус, фокус, фокус



Одна из самых больших проблем при проведении семинара с большим количеством участников — удержание обсуждения в определенных рамках. Как показано на рис. 2.7, координатор должен сначала рассмотреть горизонтальный охват — подмножество возможных требований пользователей, которые группа должна обсудить на конкретном семинаре. Существует также вертикальный охват — глубина исследования выбранных элементов.

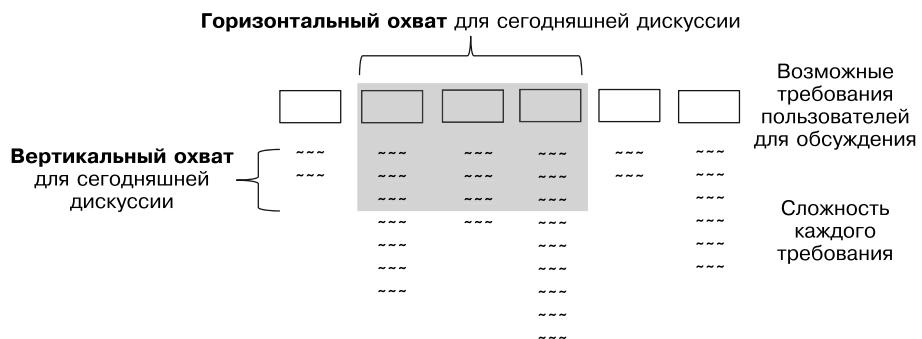


Рис. 2.7. Участники семинара по выявлению требований должны сосредоточиться на конкретном подмножестве требований, достигая определенной глубины понимания

Семинары не предназначены для выявления всех деталей каждого требования. На начальном этапе достаточно узнать о требованиях ровно столько, чтобы команда могла оценить их размеры и определить порядок реализации. Уделите достаточно времени обсуждению формулировок, чтобы все участники поняли суть каждого требования.

А детали бизнес-аналитик обсудит отдельно с разными пользователями в нужное время, например непосредственно перед реализацией требования, назначенной в конкретной итерации разработки.

Координатор должен уважать мнение участников семинара и не только стараться сосредоточить обсуждение на текущей теме, но и позволять другим реализовать потребность быть услышанными. Если кто-то поднимает вопрос, выходящий за рамки обсуждения, то координатор должен записать идею, чтобы запланировать ее обсуждение в будущем, а затем быстро вернуться к рассматриваемой теме.



Не ограничивайтесь обсуждениями в группах

Вы легко можете упустить что-то важное, обсуждая вопросы с ограниченным количеством участников семинара. Имейте в виду, что цель ограничения численности группы не исключить вклад других, а ускорить прогресс. Чтобы заполнить пробелы, используйте помимо семинаров асинхронные методы выявления требований, такие как обсуждение на форумах, стремясь получить информацию от большего количества людей. Я считаю полезным записывать информацию, полученную на каждом семинаре, и быстро распространять ее среди участников в целях ознакомления, а также среди большей группы в целях получения комментариев, исправлений и доработок. Вовлечение других заинтересованных сторон в процесс поможет вам расширить знания, подтвердить результаты семинара, информировать сообщество о направлении проекта и обеспечить их участие.

Переговоры для достижения консенсуса требуют обсуждения более длительного, чем простое голосование, делегирование принятия решения одному человеку или подбрасывание монеты.



В процессе сбора информации из различных источников кто-то должен разрешать конфликты между требованиями. Противоречивой может быть даже терминология. Два человека могут использовать разные слова для обозначения одного и того же или наделять один термин

разными смыслами, что часто приводит к путанице. Кто-то также должен согласовать приоритеты требований, полученных от разных групп пользователей, и решить, какие предлагаемые изменения принять. Для каждого проекта следует определить, кто принимает решения по этим вопросам и как эти решения должны приниматься, то есть выработать свои правила принятия решений (Gottesdiener, 2002). Одни процессы принятия решений занимают больше времени, чем другие. Переговоры для достижения консенсуса требуют обсуждения более длительного, чем простое голосование, делегирование принятия решения одному человеку или подбрасывание монеты.

Проведение семинаров — эффективный способ совместного изучения требований. Он позволяет бизнес-аналитику привести команду представителей заинтересованных сторон к общему пониманию требований и их решений. Но если группа слишком большая, то вы можете оказаться в затруднительном положении, когда делаете все возможное, чтобы люди не теряли интереса к теме, но они уходят от вас уставшими и разочарованными. Небольшие группы продвигаются к цели гораздо быстрее. И они всегда смогут договориться о том, какой пожарный выход использовать в случае необходимости.

Урок 15

Когда принимаете решение о добавлении функций, избегайте расстановки приоритетов по децибелам

Вы, наверное, слышали поговорку: «Смазывают то колесо, которое скрипит». По аналогии тот, кто громче всех отстаивает свои требования, получает наивысший приоритет. Я называю это *расстановкой приоритетов по децибелам*. Это не лучшая стратегия.

Большинство команд сталкиваются с тем, что функциональных возможностей так много, что все их сложно добавить в проект. Лица, принимающие решения, должны выбрать, что добавить прямо сейчас, что отложить, от чего вовсе отказаться. Даже имея возможность реализовать все, что запрошено, вы должны определить наиболее подходящую последовательность реализации. Одни запрошенные возможности являются важными и срочными; они идут первыми. Вторые важны, но их можно отложить на потом, а третья не являются ни важными, ни срочными. Расстановка приоритетов — обязательный элемент планирования проекта. Она помогает распределить требования между итерациями, сборками или выпусками.

Цель расстановки приоритетов — быстрое и недорогое предоставление максимальной ценности для клиента.

Каждому заинтересованному лицу нравится думать, что его потребности являются самыми важными. Влиятельные, волевые или громкоголосые менеджеры и клиенты могут оказывать сильное давление, требуя в первую очередь удовлетворить их потребности. Однако самые громкоголосые клиенты не всегда отстаивают самые важные с точки зрения бизнеса функции. Цель расстановки приоритетов — быстрое и недорогое предоставление максимальной ценности для клиента. Не нужно тратить силы на реализацию функций, не способствующих успеху продукта. Соответственно, продуманная расстановка приоритетов подразумевает анализ множества факторов, помимо громкости заявлений.



Методы определения приоритетов

Люди разработали множество методов определения приоритетов требований к программному обеспечению. Несколько исследователей выяснили, какие из них наиболее практичные и эффективные (Hasan et al., 2010; Kukteja et al., 2012; Achimugu et al., 2014). Вот некоторые из них:

- трехуровневая классификация, когда приоритеты сводятся к трем уровням: высокому, среднему и низкому;
- классификация MoSCoW (Must, Should, Could, Won't — обязательно, желательно, возможно, не нужно);
- попарное сравнение отдельных функций, требований, вариантов использования или историй для сортировки по приоритетам;
- упорядочивание приоритетов похожих требований;
- распределение 100 баллов между отдельными требованиями и присваивание наибольшего количества баллов требованиям с наивысшим приоритетом;
- игра в планирование; обычно используется в Agile-проектах, где клиенты и разработчики совместно составляют список пользовательских историй в порядке их относительного приоритета (Shore, 2010);

- аналитические методы ранжирования требований на основе их ценности для продукта и стоимости реализации (Wiegers, Beatty, 2013).

Критерии выбора приоритетов

При назначении приоритетов вариантам использования, сценариям отдельных вариантов, функциям и подфункциям, функциональным требованиям или пользовательским историям мыслительный процесс протекает одинаково.

Принимая решение о том, какие требования являются обязательными, какие желательными, а какие необязательными, учитывайте следующие факторы.

Бизнес-цели

Наиболее весомый критерий — степень, в которой каждое требование поможет организации достичь ее бизнес-целей. Соответствующие суждения основываются на бизнес-требованиях проекта, которые спонсор должен определить на ранней стадии проекта. Они описывают создаваемую или используемую бизнес-возможность и количественно определяют бизнес-цели проекта. Не имея четких целей, трудно принимать рациональные решения о том, что и когда реализовать.

Группы пользователей

Не все группы пользователей равнозначны. Удовлетворение требований привилегированных групп пользователей больше способствует успеху бизнеса, чем удовлетворение потребностей других групп. В анализ заинтересованных сторон входит определение групп пользователей, наиболее важных (предпочтительных) для проекта, чтобы вы могли присвоить их требованиям более высокий приоритет.

Частота использования



Знание, как часто используются те или иные функции, помогает решить, какие из них следует реализовать в первую очередь. Один из способов оценить частоту использования — изучение *особенностей применения* приложения с определением. В нем описывается, какой процент времени пользователи будут выполнять каждую операцию в ходе сеанса работы с продуктом (Musa, 1993). Например, какой процент пользовательских сеансов на сайте авиакомпании связан

с бронированием авиабилетов, изменением или отменой бронирования, проверкой статуса рейса или отслеживанием пропавшего багажа? При прочих равных условиях — хотя и не всегда — наиболее часто используемые операции будут иметь наивысший приоритет реализации.

Соответствие нормативным требованиям

Требования, позволяющие продукту достичь соответствия нормативным документам или сертификации, должны иметь высокий приоритет. Очевидно, что независимо от наличия востребованных функций вы не сможете продать свой продукт, если он не будет сертифицирован. Эти требования влияют на функциональность, которая нужна определенным заинтересованным сторонам, но не видна большинству пользователей. Примерами могут служить требования безопасности, запись истории доступа и создание журналов аудита. Некоторые из этих внутренних функций могут иметь высокий приоритет, даже если ни один конечный пользователь не запрашивает их.

Основополагающая функциональность

Некоторые возможности следует реализовать на раннем этапе, даже если они не приносят немедленную выгоду тому, кто будет их использовать. Это нужно потому, что данные возможности служат основой для последующих функций. То есть, определяя последовательность реализации, нужно учитывать зависимости между требованиями. От некоторых функций может зависеть архитектурная надежность сложного продукта. Добавление подобной функциональности на более поздних этапах проекта может иметь разрушительные последствия, поэтому ее нужно создавать раньше, а не позже.

Рискованная функциональность

Требования, подразумевающие высокий риск с точки зрения реализации, следует реализовывать на ранней стадии, чтобы оценить их выполнимость и снизить общий технический риск проекта.

Анализ превыше громкости

Выбирая любой из методов расстановки приоритетов, предпочтительнее учитывать вышеперечисленные факторы, а не громкость голосов в комнате. Самое скрипучее колесо необязательно смазывать первым.

Урок 16

Не задокументировав и не согласовав содержимое проекта, нельзя узнать, увеличивается ли его объем

Когда я спрашиваю своих студентов, кому из них приходилось работать над проектом, который постепенно разбухал, почти все поднимают руку. Когда затем я спрашиваю, в скольких из этих проектов был четко определен их объем, руки поднимаются единицы. Что вообще означает «разбухание проекта», если масштаб последнего никогда не был четко определен?

Призрак разбухания проекта

Проблема разбухания (непрерывного и неконтролируемого расширения функциональности) преследует программные проекты с незапамятных времен. Разбухание часто называют главной причиной того, почему реализация проекта не укладывается в запланированный график. Но вы даже не узнаете, имеет ли место разбухание проекта, если у вас не будет согласованной точки отсчета, где утверждается: «Вот что мы намерены сделать в этот период времени».

Всякий запланированный этап работы начинается с базовой функциональности, которую команда должна реализовать на данном этапе.



Объем проекта можно определить как набор возможностей, с реализацией которого в конкретной итерации, сборке или выпуске продукта соглашаются заинтересованные стороны. Объем определяет границу между тем, что есть, и тем, чего нет. Объем любой части проекта определяется как подмножество функций продукта, ступенька на пути от начала проекта до окончательного выпуска продукта. Всякий запланированный этап работы начинается с базовой функциональности, которую команда должна реализовать на данном этапе. Эта базовая линия является отправной точкой, позволяющей определять изменение объема.

Люди традиционно считают разбухание проекта чем-то плохим. Оно свидетельствует о том, что выявленные требования были неполными

или неточными и это повлекло непрекращающийся поток дополнительных требований. Как мы уже видели, в любом крупном проекте невозможно заранее определить все требования, а ожидания, что их перечень останется неизменным, нереалистичны. В каждом проекте должны предполагаться изменение и рост количества требований по мере того, как пользователи пробуют ранние версии, получают новые идеи и лучше понимают задачу.

Сдерживание изменений из-за боязни разбухания проекта может привести к созданию продуктов, которые реализуют свое первоначальное видение, но не удовлетворяют потребности клиентов. Однако и непрерывное разбухание может навредить проекту, если планы и графики не учитывают возможности изменений и не уделяют времени разрешению непредвиденных обстоятельств и постоянной расстановке приоритетов. (См. урок 25 «Айсберги всегда больше, чем кажутся».) Добавление каждого нового требования гарантирует перерасход графика и бюджета.



В Agile-проектах участники намеренно не пытаются выявить объем всего проекта и определяют объем каждой итерации с учетом требований, которые владелец продукта наметил для реализации. В список требований постоянно добавляются новые пункты. Владелец продукта расставляет приоритеты для новых требований с учетом еще не реализованных, чтобы определить, какие из них должны быть реализованы в ближайших итерациях. Этот подход помогает точно выявить потребности пользователей, но может привести к неопределенности времени доставки конечного продукта.

Как документировать объем

Простейший способ представить объем — перечислить требования, функции или пожелания, запланированные для реализации в течение определенного цикла разработки, как это предполагает методология Agile-разработки (Thomas, 2008а). К другим полезным способам представления объема на различных уровнях детализации относятся:

- **контекстная диаграмма**, которая ничего не говорит о внутреннем устройстве системы, но идентифицирует объекты вне системы (пользователей, другие программные системы, аппаратные устройства), которые подключаются к ней (Wiegers, Beatty, 2013);
- **диаграмма вариантов использования**, на которой изображены действующие лица за границами системы и варианты использова-

ния, с помощью которых они взаимодействуют с системой (Ambler, 2005);

- **диаграмма экосистемы**, на которой показана взаимосвязь нескольких систем, что позволяет оценить волновой эффект изменений в системах, которые не взаимодействуют напрямую с вашей (Beatty, Chen, 2012);
- **журнал реализации требований в итерации**, содержащий набор требований, которые Agile-команда планирует завершить в течение одной итерации (Scaled Agile, 2021a); в спринте методологии Scrum называется незавершенными работами (backlog);
- **карта пользовательских историй**, на которой показаны действия, шаги и детали пользовательских историй, определяющие объем всего продукта, итерации, конкретной функции или части пользовательского опыта (Kaley, 2021);
- **дорожная карта функций**, помогающая определить несколько уровней наращивания возможностей для каждой функции, а затем описать область применения конкретного выпуска и перечислить уровни расширения отдельных функций, добавленных в этот выпуск (Wieggers, 2006a);
- **дерево функций**, визуально разбивающее основные функции на подфункции, которые планировщик может сгруппировать в целях определения объема каждого цикла разработки (Beatty, Chen, 2012);
- **список событий**, определяющий внешние события, которые будут обрабатываться при каждом выпуске (Wieggers, Beatty, 2013).



В каждом случае цель определения объема состоит в том, чтобы выяснить, какие возможности должна предоставлять конкретная часть проекта. Выявление этих возможностей помогает установить границу области применения и задать точку отсчета для рассмотрения изменений в этой области в процессе разработки.

Это входит в рамки проекта?

Поскольку объем так или иначе будет меняться, в каждом проекте должен быть определен практический процесс управления изменениями. Простое добавление требований в список без их анализа может только навредить. Необходим налаженный механизм, с помощью которого заинтересованные стороны смогут запрашивать внесение изменений, чтобы специалисты могли оценить их влияние и решить,

следует ли их внедрять. Формальный процесс внесения изменений не должен начинаться, пока команда не установит базовые границы определенного объема работы. До этого момента требования и объем проекта динамично меняются. Во время выполнения каждого этапа работы контроль над изменениями должен становиться все более строгим, чтобы повысить вероятность достижения базовых целей в соответствии с графиком.

Когда кто-то предлагает новое требование, следует задать вопрос: «Вписывается ли оно в объем?» Есть три возможных ответа.



- **Да, это требование явно вписывается в рамки проекта.** Предлагаемый функционал необходим для достижения наших целей в текущем цикле разработки. Поэтому мы должны добавить это требование в наш список.
- **Нет, это требование явно выходит за рамки проекта.** Предложенное требование не способствует достижению бизнес-целей в текущем цикле, поэтому его не нужно принимать в реализацию прямо сейчас. Но мы можем добавить его в список требований, чтобы рассмотреть в дальнейшем или отклонить.
- **Это требование не вписывается в текущие рамки проекта, но относится к категории желательных.** Спонсор проекта должен принять бизнес-решение: следует ли увеличивать объем проекта ради реализации новых возможностей. Если запрошенное изменение представляет определенную ценность для бизнеса, то правильным будет увеличить объем проекта. Но такие решения всегда имеют свою цену. Если объем увеличивается, то нужно изменить и что-то еще: другие функции, расписание, стоимость, численность персонала или качество. Изменения никогда не бывают бесплатными.



Расплывчатые требования = расплывчатый объем проекта

Расплывчатое описание требований чревато нечетким пониманием объема проекта. Из-за двусмысленности одна сторона может думать, что определенная функция явно входит в рамки проекта, а другая будет иметь противоположное мнение. Расплывчатые термины, такие как «поддержка», «и т. д.», «например» и «включая» (или того хуже: «включая, но не ограничиваясь»), по своей сути являются неопределенными. Они вызывают у меня чувство отторжения, когда я вижу их в заявлениях о требованиях. Приведу некоторые примеры.



- «Система должна обрабатывать символы А, Б, В и т. д.». Все ли читатели будут единодушны в том, где заканчивается этот список и что в нем содержится? Пространное *и т. д.* заставляет меня нервничать.
- «Система должна поддерживать документы Microsoft Word». У читателей могут быть очень разные представления о том, какие именно функции подразумеваются под *поддержкой*. Предположим, что руководитель проекта или владелец продукта строит планы на основе своей интерпретации слова *«поддерживать»*, а позже обнаруживает, что клиент, запросивший эту функцию, имел гораздо более широкие ожидания. Это ведет к разбуханию проекта? Или просто является уточнением первоначального ожидания, обнажившего подводную часть айсберга? Тщательное описание требований перед принятием обязательств помогает избежать таких проблем.



Неоднозначные и неполные требования могут вызвать проблемы при обсуждении объема контрактных проектов. В одном проекте требовалось, чтобы поставщик перенес несколько наборов данных из существующей информационной системы клиента в свой новый программный пакет (Wiegers, 2003). После того как проект перешел в стадию реализации, клиент выявил у себя еще несколько наборов данных для преобразования. Он посчитал, что реализация функций преобразования данных входит в первоначальный объем проекта, и отказался доплачивать за разработку этих функций. Поставщик не согласился. В результате этой и некоторых других проблем проект был отменен — и начался дорогостоящий судебный процесс. Еще один пример: моего друга-консультанта наняли в качестве свидетеля-эксперта в пять контрактных проектов, которые привели к судебным разбирательствам и многомилионным выплатам. Четыре проекта закончились неудачно вследствие плохо выявленных требований, и два из них были связаны с проблемами определения объема.



Мудрые подрядчики предусматривают дополнительное время на разрешение непредвиденных обстоятельств, стараясь учесть возможность некоторого разбухания проекта и расплывчатости требований. Однако увеличение цены проекта, вызываемое такой предусмотрительностью, может оттолкнуть потенциального клиента. В контрактах должно быть четко указано, как следует оценивать увеличение объема работ и кто будет за это платить. Чем четче вы определите, что включено, а что нет, тем легче пройдут дебаты.

У меня есть друг, который следит за тем, чтобы цель проекта была изложена настолько кратко, чтобы могла уместиться на футболке. Он ясно

выражает эту цель в повестках собраний и заметках, документах и других публичных источниках. Я также знаю людей, писавших свои заявления о видении и объеме работ на плакатах, которые они приносили на обсуждение требований. Четкая определенность цели помогает людям ответить на вопрос: «Необходимо ли новое требование для достижения этой цели?» Это основная сложность, заключающаяся в управлении требованиями.

Изменения случаются, но чрезмерные изменения предполагают, что изначально никто не проработал задачу должным образом. Четкое определение запланированного объема работ позволяет команде сосредоточиться на создании ценного результата в рамках графика и бюджетных ограничений и принимать важные бизнес-решения, когда поступают запросы на изменение.

СЛЕДУЮЩИЕ ШАГИ: ТРЕБОВАНИЯ



1. Определите, какие уроки, описанные в этой главе, имеют отношение к вашему опыту разработки требований и управления ими.
2. Можете ли вы, опираясь на свой опыт, вспомнить какие-либо другие, связанные с требованиями, уроки, которыми стоит поделиться с коллегами?
3. Перечислите описанные в этой главе методы, способные помочь в решении связанных с требованиями задач, которые мы определили во врезке «Первые шаги» в начале главы. Как каждый метод может улучшить работу с требованиями ваших проектных групп?
4. Как вы определили, приносит ли желаемые результаты каждый метод, перечисленный на шаге 3? Насколько ценны для вас эти результаты?
5. Определите любые препятствия, которые могут затруднить применение методов, перечисленных на шаге 3. Как бы вы справились с ними? Заручились бы поддержкой коллег, готовых помочь вам в реализации этих методов?
6. Внедрите описание процессов, шаблоны, руководящие документы, контрольные списки и другие инструменты, чтобы помочь будущим проектным группам эффективно применять ваш передовой опыт по работе с локальными требованиями.

Глава 3

Проектирование

ВВЕДЕНИЕ

Если цель выявления требований состоит в определении характеристик, которыми должны обладать решения, то суть проектирования — в разработке этих решений. Некоторые говорят, что требования отвечают на вопрос «*что?*», а проектирование — на вопрос «*как?*». Но на самом деле все не так однозначно.

Граница между требованиями и проектным решением — это не четкая черная линия, а размытая серая область (рис. 3.1) (Wiegers, 2006а). Во время исследования требований полезно предпринять некоторые предварительные действия в области проектирования, например создать прототипы. Размышление о том, как исследование проблемы может привести к решению, помогает людям уточнить требования к продукту.

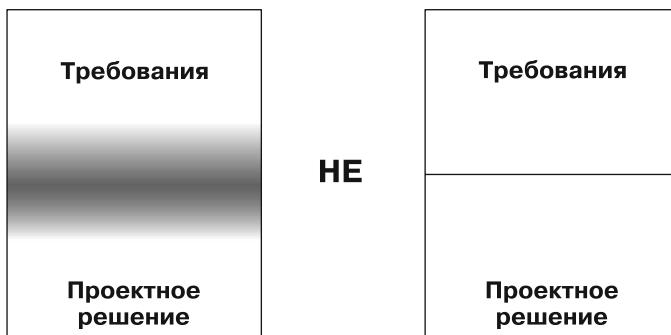


Рис. 3.1. Граница между требованиями и проектным решением — не четкая черная линия, а размытая серая область

Пользователи, взаимодействующие с прототипом, часто замечают, что это проясняет их мышление и способствует появлению новых идей, поскольку прототип более осязаем, чем абстрактный список требований.

Переход даже от четко описанных требований к конкретным элементам проектного решения не является ни простым, ни очевидным.

Суть проектирования программного обеспечения заключается в том, чтобы связать требования с фрагментами кода, но переход даже от четко описанных требований к конкретным элементам проектного решения не является ни простым, ни очевидным (Davis, 1995). Всякий раз, когда меня спрашивают, как люди разрабатывают ПО, я вспоминаю старую карикатуру Сидни Харриса (Sidney Harris), на которой изображены двое ученых, стоящих перед доской, исписанной уравнениями. Один ученый указывает на место на доске, где написано: «И затем происходит чудо», — и предлагает уточнить этот раздел.

Некоторые этапы проектирования программного обеспечения кажутся неким волшебством, которое стало возможным благодаря опыту и интуиции разработчика. Одни виды деятельности, такие как проектирование баз данных, носят систематический и аналитический характер. Другие более органичны: проектное решение выкристаллизовывается постепенно, по мере того как разработчики исследуют переход от задачи к решению. Проектирование пользовательского опыта (*user experience design*) включает художественно-творческие подходы, основанные на глубоком понимании человеческого фактора. Разработчики часто полагаются на общие паттерны, которые повторяют при проектировании ПО, чтобы уменьшить количество необходимых изобретений (Gamma et al., 1995). Кен Пью (Pugh, 2005) дает некоторое представление о мыслительном процессе разработчика в своей книге *Prefactoring*.

Различные аспекты проектирования

Проектирование программного обеспечения состоит из: архитектурного, технического (или низкоуровневого), проектирования баз данных и проектирования взаимодействия с пользователем (рис. 3.2). У всех этих видов проектирования есть многочисленные ограничения, сужающие выбор вариантов, доступных разработчику. Сложности могут возникать



из-за требований совместимости с другими продуктами, применимых стандартов, технологических ограничений, бизнес-политик, правил, стоимости и других факторов. Физические продукты, содержащие встроенное ПО, подвержены другим ограничениям, которые касаются размеров, веса, материалов и интерфейсов. Ограничения усложняют проектирование, сообщая разработчикам, чего они *не могут* делать, точно так же как требования диктуют, что *должна* делать реализация.

Наша система

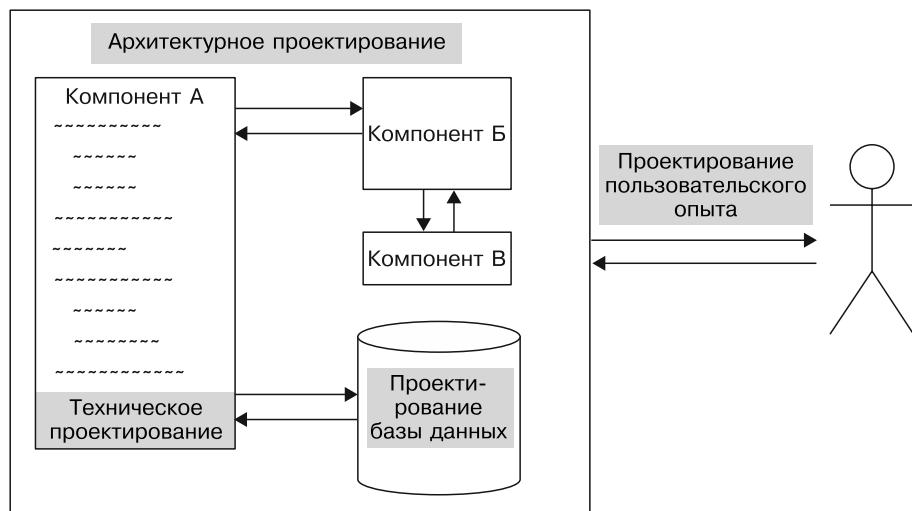


Рис. 3.2. В программных системах предусматривается четыре вида проектирования: архитектурное, техническое, базы данных и взаимодействия с пользователем

Архитектурное проектирование определяет структуру системы и ее компоненты, или архитектурные элементы (Rozanski, Woods, 2005). Эти элементы состоят из модулей кода программных систем, которые могут быть объединены в несколько взаимосвязанных подсистем в крупном продукте. Физические продукты со встроенным ПО будут включать механические и электрические аппаратные компоненты. Архитектурное проектирование предусматривает разделение системы на компоненты, определение обязанностей каждого компонента и назначение конкретных требований соответствующим компонентам. Определение интерфейсов между компонентами — еще один аспект архитектурного проектирования. (См. урок 22 «Проблемы многих систем скрываются в интерфейсах».)



Техническое проектирование подразумевает проработку логической структуры отдельных компонентов программы (модулей кода, классов и их методов, сценариев и т. д.) и деталей интерфейсов между модулями. Одна из важнейших сторон технического проектирования — разработка алгоритмов.

Проектирование базы данных необходимо, когда приложение создает, изменяет базу данных или обращается к ней. На этом этапе определяются сущности или классы данных и отношения между ними, а также перечисление элементов данных каждой сущности, их типов, свойств и логических связей. Определение процедур для создания, чтения, изменения и удаления хранимых данных (иногда их все вместе называют CRUD-процедурами (*create, read, update, delete*)) тоже является частью проектирования базы данных. Разработка функций отчетности и макетов отчетов затрагивает сразу два аспекта: проектирование базы данных и проектирование пользовательского опыта. В конечном счете данные помещаются в компьютер, только чтобы люди могли снова получить их в некой полезной форме.

Разработка любого приложения, используемого людьми, подразумевает *проектирование пользовательского опыта*, который сам по себе является весьма обширной дисциплиной. *Проектирование пользовательского интерфейса*, также называемое проектированием человеко-машинных взаимодействий, является подмножеством проектирования пользовательского опыта и подразумевает архитектурное и техническое проектирование пользовательского интерфейса. Архитектура пользовательского интерфейса определяет диалоговые элементы (точки взаимодействия пользователя с системой) и пути навигации между ними, повторяющие сценарии решения пользовательских задач. При техническом проектировании пользовательского интерфейса учитывается специфика взаимодействий пользователя с продуктом, в том числе продумываются эскизы экрана, внешний вид, элементы управления и свойства отдельных текстовых блоков, графики, поля ввода и выходные экраны. Оба этапа проектирования пользовательского интерфейса — архитектурное и техническое — определяют воспринимаемую пользователем простоту обучения и использования или в совокупности — удобство использования.

У вас хороший проект?

Проектирование предполагает разработку оптимального решения, которое будет соответствовать множеству требований в течение всего срока службы продукта. Проектное решение должно позволять реали-





звать правильную функциональность и достичь ожидаемых характеристик всех атрибутов качества. (См. урок 20 «Невозможно оптимизировать все желаемые атрибуты качества».) Кроме того, проектное решение должно эффективно обеспечивать расширение и изменение как в процессе разработки, так и после выпуска.

За многие годы пионеры проектирования программного обеспечения, такие как Эдсгер Дейкстра (Edsger Dijkstra), Дэвид Парнас (David Parnas), Барбара Лисков (Barbara Liskov), Ларри Константин (Larry Constantine) и Гленфорд Майерс (Glenford Myers), вывели принципы, помогающие разработчикам добиваться лучших результатов, а другие разработчики, в свою очередь, объединили их в полезные ресурсы (Davis, 1995; Gamma et al., 1995; Pugh, 2005). Соблюдение принципов, изложенных ниже, способствует уменьшению сложности проектного решения, повышению его надежности и упрощению понимания, изменения, расширения и повторного использования.

- **Разделение обязанностей.** Проект должен быть разделен на модули, независимые друг от друга и имеющие четко определенные и не пересекающиеся обязанности.
- **Сокрытие информации.** Каждый модуль должен скрывать внутренние детали своих данных и алгоритмов от остальной системы. Другие модули должны получать доступ к данным и службам данного модуля только через четко определенный интерфейс. В этом случае реализацию каждого модуля можно изменить при необходимости, не затрагивая другие модули, которые его вызывают.
- **Слабая связанность.** Под связанностью понимается степень переплетения программных компонентов. Хорошо продуманный модульный проект демонстрирует слабую связанность между компонентами, поэтому изменение одного модуля требует минимальных изменений других (TutorialsPoint, 2021).
- **Сильная связность.** Связность определяет степень логической согласованности функций в модуле. Модуль, для которого характерна сильная связность, идеально выполняет одну четко определенную задачу (Mancuso, 2016).
- **Абстракция.** Позволяет разработчикам писать код, не зависящий от конкретных деталей реализации, таких как тип операционной системы или пользовательский интерфейс. Абстракция упрощает переносимость и повторное использование.
- **Четко определенные и удобные интерфейсы.** Четко определенный интерфейс модуля позволяет разработчикам других модулей

кода легко обращаться к службам данного модуля, а также упрощает замену модуля, когда это необходимо, поскольку интерфейс, который он предоставляет остальной части системы, остается неизменным. Тот же принцип применяется к внешним интерфейсам, которые система предоставляет вовне.

При обсуждении проектирования его иногда рассматривают как прямое расширение требований или связывают с реализацией как «разработку», но лучше рассматривать его как отдельный вид деятельности. И многие, кстати, тщательно проектируют каждый программный продукт независимо от того, считается ли проектирование самостоятельным видом деятельности и фиксируется ли оно в той или иной форме.

Я работал над проектом, в котором программирование непосредственно на основе требований привело бы к созданию гораздо более сложной программы, чем у нас получилось после предварительного изучения вариантов проектных решений. Из требований это не было очевидно, но три из восьми модулей системы использовали один и тот же алгоритм, еще три — универсальный алгоритм, а последние два — третий алгоритм. В какой-то момент мы бы заметили, что пишем один и тот же код несколько раз, но хорошо, что повторение обнаружилось до этапа реализации.



Не спешите переходить от требований сразу к коду. Найдите время, оцените альтернативные варианты проектного решения и выберите наиболее подходящее. В этой главе описываются шесть ценных уроков, которые я извлек из своего опыта разработки программного обеспечения.

ПЕРВЫЕ ШАГИ: ПРОЕКТИРОВАНИЕ



Прежде чем вы перейдете к изучению уроков, связанных с проектированием, предлагаю вам потратить несколько минут на следующие действия. По мере чтения подумайте, в какой степени каждый из этих пунктов применим к вашей организации или команде.

1. Перечислите приемы проектирования, в которых особенно преуспела ваша организация. Задокументирована ли информация об этих приемах? Доступна ли она другим членам команды, чтобы они могли ознакомиться с ними и применять на практике?
2. Определите любые проблемы (болевые точки), отмечавшиеся в ваших командах и связанные с архитектурным, техническим проектированием, проектированием базы данных и пользовательского опыта.

3. Опишите, как каждая проблема влияет на вашу способность успешно завершать проекты. Как они мешают достижению успеха в бизнесе и разработчикам, и их клиентам? Огрехи в проектном решении могут привести к снижению надежности системы, что нелегко исправить, низкой производительности, дублированию кода, несоответствиям внутри продукта или между связанными продуктами и к проблемам с удобством использования.
4. Для каждой проблемы, выявленной на шаге 2, определите основные причины, провоцирующие или усугубляющие ее. Проблемы, влияния и первопричины могут сливатся, поэтому постарайтесь разделить их и увидеть, как они связаны. Вы можете найти несколько основных причин, способствующих появлению одной и той же проблемы, или несколько проблем, обусловленных одной общей причиной.
5. Читая эту главу, отмечайте любые практики, которые могут быть полезны вашей команде.

Урок 17 Проектирование — итеративный процесс

В своей классической книге «Мифический человеко-месяц» Фредерик П. Брукс-младший советует: «Планируйте выбросить первую версию — вам все равно придется это сделать». Брукс имеет в виду, что в крупных проектах желательно создавать пилотную или подготовительную систему, чтобы выяснить, как лучше построить основную. Это недешевое удовольствие, особенно если в систему входят аппаратные компоненты. Тем не менее пилотная система может пригодиться, если есть сомнения в технической осуществимости или подходящая стратегия проектирования изначально неясна. Пилотная система также может помочь выявить неизвестные факторы, о существовании которых вы даже не догадывались.

Вряд ли вы согласитесь создать, а потом выбросить первую версию продукта, и все же я советую пересмотреть потенциальное проектное решение, прежде чем команда слишком далеко продвинется в разработке. Идея создания максимально простого проекта выглядит привлекательно и действительно ускоряет доставку решения. Быстрая доставка может повысить ценность продукта для покупателя в краткосрочной перспективе, но редко когда оказывается лучшей долгосрочной стратегией, особенно если продукт продолжает развиваться.

Любая программная задача имеет несколько проектных решений, и редко лучшим оказывается только одно из них (Glass, 2003). Первый проект, который вы создадите, наверняка получится не самым лучшим. Норман Керт, опытный разработчик программного обеспечения, хорошо объяснил мне:

Работу по проектированию можно считать выполненной, только если вы придумали хотя бы три решения, отбросили их все, поскольку они недостаточно хороши, а затем объединили лучшие их части в превосходное четвертое решение. Иногда, только рассмотрев три варианта, начинаешь осознавать, насколько плохо понимаешь проблему. Поразмышляв, вы сможете найти простое решение, обобщив задачу.



Разработка программного обеспечения не является линейным, упорядоченным, систематическим и предсказуемым процессом. Лучшие разработчики часто сначала сосредотачиваются на трудных частях, где решение может быть неочевидным или даже неосуществимым (Glass, 2003). Имеется несколько методов, помогающих разработчикам переходить от первоначальной концепции к эффективному решению. Один из них — создание и уточнение графических моделей (диаграмм) проектов. Этот метод рассматривается в уроке 18 «Чем выше уровень абстракции, тем проще выполнять итерации». Прототипирование — еще один ценный метод, помогающий последовательно выполнять техническое проектирование и улучшать пользовательский опыт.



Сила прототипов

Прототип — это частичное, предварительное или возможное решение. В прототипе вы воплощаете часть системы, проверяя, насколько правильно вы понимаете, как спроектировать решение. Если эксперимент не удался, то вы меняете подход и пробуете снова. Прототип помогает оценить риски и снизить их, особенно если используется новый архитектурный или проектный шаблон, который следует проверить, прежде чем принять его за основу.

Если вы намереваетесь превратить прототип в продукт, то должны с самого начала его создания придавать ему промышленное качество.



Прежде чем создавать прототип, определите, что вы собираетесь делать: отказаться от него и затем разработать реальное решение или превратить это предварительное решение в окончательный продукт. Если вы намереваетесь превратить прототип в продукт, то должны с самого начала его создания придавать ему промышленное качество. Это потребует больше усилий, чем создание чего-то временного, от чего вы откажетесь после выполнения своей задачи. Чем больше сил вы вкладываете в прототип, тем с меньшей охотой поменяете его или откажетесь от него, что мешает поступательному движению вперед. Подход к прототипированию должен поощрять циклическую доработку и даже помогать начинать все сначала, если потребуется.

Прежде чем приступить к конкретному решению, Agile-команды иногда создают истории, называемые *спайками* (spike¹), предназначенные для исследования технических подходов, устранения неопределенности и снижения риска (Leffingwell, 2011). В отличие от других пользовательских историй, основной результат спайка — не рабочий код, а знания. В спайки может входить создание технических прототипов, прототипов пользовательского интерфейса или того и другого в зависимости от искомой информации. Спайк должен иметь четкую цель, как и научный эксперимент. У разработчика есть гипотеза, и для получения доказательств ее верности или неверности, подтверждения обоснованности какого-либо подхода или быстрого принятия технического решения реализуется спайк.

Проверка концепции



Прототипы для проверки концепции, также называемые *вертикальными прототипами*, полезны для проверки предлагаемой архитектуры. Однажды я работал над проектом, предполагавшим реализацию нетрадиционного клиент-серверного подхода. Эта архитектура имела определенный смысл для нашей вычислительной среды, но мы хотели убедиться, что не загоняем себя в угол. Мы построили прототип для проверки концепции с вертикальным срезом функциональности от пользовательского интерфейса через уровни связи до вычислительного движка. Он подтвердил верность нашей идеи и помог нам убедиться в работоспособности этого проектного решения.

Экспериментирование с прототипом для проверки концепции — относительно недорогой способ итерации, даже притом что требует

¹ Spike — «всплеск». Здесь имеется в виду всплеск некой активности. — Примеч. пер.

создания некоего действующего программного обеспечения. Такие прототипы полезны для оценки технических аспектов проекта: архитектуры, алгоритмов, структуры базы данных, системных интерфейсов и взаимодействий. С помощью прототипа можно оценить свойства архитектуры, такие как производительность, безопасность, защищенность и надежность, а затем постепенно улучшать их.

Макеты

Проектирование пользовательского интерфейса всегда должно выполняться итеративно. Даже следуя принятым соглашениям о пользовательском интерфейсе, вы должны провести хотя бы неформальное тестирование удобства использования, чтобы выбрать подходящие элементы управления и схемы их размещения, обеспечивающие простоту освоения и использования, а также высокую доступность. Например, А/В-тестирование — это подход, при котором вы предоставляете пользователям два альтернативных интерфейса и даете им возможность выбрать наиболее удобный для них. Люди, которые проводят А/В-тестирование, могут наблюдать за поведением пользователей с помощью различных подходов и определять, какой вариант понятнее или быстрее приводит к успешным результатам. Проводить такие эксперименты проще, быстрее и дешевле, находясь на этапе изучения дизайна, чем после доставки решения клиентам, реагируя на их жалобы или на более низкую, чем ожидалось, частоту щелчков на веб-странице.

Как и в случае с требованиями, проектирование пользовательского опыта продвигается успешнее при постепенном уточнении деталей с помощью прототипирования. С этой целью можно создавать *макеты*, также называемые *горизонтальными прототипами*, поскольку они представляют собой тонкий слой пользовательского интерфейса без какой-либо функциональной основы под ним. Макетами могут быть и простые эскизы экрана, и действующие интерфейсы, которые выглядят аутентично, но не выполняют никакой реальной работы (Coleman, Goodwin, 2017). Ценны даже простые бумажные прототипы — их можно быстро создавать и изменять. Вы можете использовать текстовый редактор или даже каталожные карточки, чтобы разместить элементы данных в прямоугольных областях, представляющих потенциальные экраны, посмотреть, как они соотносятся друг с другом, и отметить, какие из них принимают ввод пользователя, а какие отображают результаты. Но при прототипировании пользовательского интерфейса осторегайтесь следующих ловушек.



- Не тратьте слишком много времени на совершенствование пользовательского интерфейса («Может, для этого текста лучше выбрать темно-красный цвет?») до того, как организуете информацию и элементы управления на экране и создадите функциональные макеты. Сначала нанесите широкие мазки.
- Клиенты или руководители могут подумать, что программное обеспечение почти готово, увидев удобный и красивый пользовательский интерфейс, даже если за ним нет ничего, кроме имитации функциональности. Менее изысканный прототип показывает, что ПО еще не готово.
- Не стремитесь подсказывать, как правильно действовать, тем, кто оценивает прототип и пытается выполнить неочевидную для них задачу. Вы не сможете судить об удобстве использования, помогая участникам тестирования изучать и применять прототип.

Если вы не потратили время на итеративное изучение пользовательского опыта и техническое проектирование до реализации, то рискуете создать продукт, который не понравится клиентам. Бездумно спроектированные продукты раздражают клиентов, вынуждают их неэффективно тратить свое время, подрывают их хорошее отношение к вашему продукту и компании и заставляют писать негативные отзывы (Wiegers, 2021). Потратив чуть больше времени на проектирование, вы значительно приблизитесь к созданию полезного и удобного решения.

Урок 18

Чем выше уровень абстракции, тем проще выполнять итерации

Один из способов усовершенствовать проект — несколько раз создать продукт целиком, улучшая его с каждым циклом. Но это непрактично. Другой способ — реализовать решение постепенно, добавляя сначала сложные или малопонятные части, и определить, какие подходы дают наилучший результат. В этом и заключается идея прототипирования, как было показано в предыдущем уроке.

Однако есть еще одна, третья стратегия, суть которой состоит в построении операционной части системы, чтобы пользователи могли работать с ней и оставлять отзывы, на основе которых можно улучшать последующие расширения.

Этот поэтапный подход лежит в основе Agile-разработки программного обеспечения. Он помогает получить от пользователя информацию о чем-то осязаемом и скорректировать работу, чтобы лучше удовлетворить потребности клиентов. Вы можете обнаружить, что первоначальный проект соответствует первичной реализации продукта, но сдерживает его дальнейшее развитие. Вдобавок вы можете обнаружить, что в свое время команда недостаточно хорошо продумала проект, торопясь выпустить действующее ПО, и теперь назрела необходимость вернуться к этим решениям. (См. урок 50 «Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения».) Устранение недостатков проектирования архитектуры и базы данных часто требует больших затрат. Поэтому поспешная реализация на первых нескольких итерациях и отсутствие тщательного изучения технических основ могут привести к болезненным последствиям для команды.



Общей чертой всех трех стратегий проектирования является создание действующего программного обеспечения для оценки идей. Поэтому поэтапное совершенствование проектного решения протекает относительно медленно и обходится дорого. Вы можете переделывать созданное несколько раз, чтобы получить подходящий проект.

Альтернативный подход — выполнение итераций на более высоком уровне абстракции, чем программное обеспечение. Как показано на рис. 3.3, итерации на низком уровне абстракции обходятся дороже,

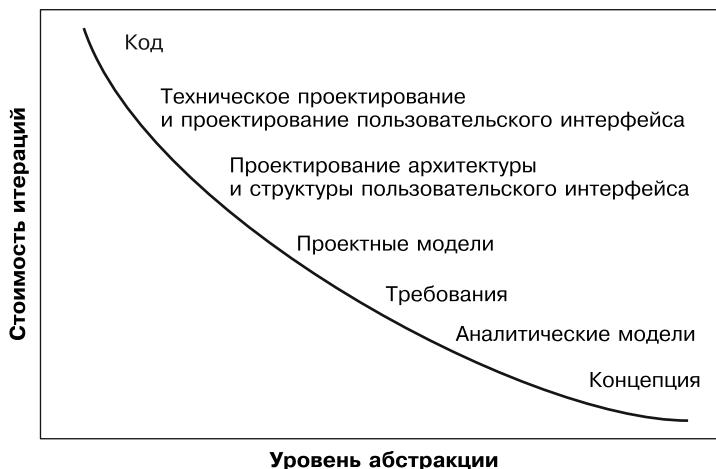


Рис. 3.3. Стоимость итераций тем ниже, чем выше уровень абстракции

поскольку приходится больше работать над созданием оцениваемых артефактов. Проектное моделирование обеспечивает менее дорогую альтернативу.

Отступая от деталей

Как для выявления требований, так и для проектирования большое значение имеет рисование схем, представляющих различные аспекты системы, и их последующее поэтапное уточнение. Нарисованную схему изменить проще, чем переписать код. Работающее программное обеспечение осозаемо; а аналитические и проектные модели абстрактны. Диаграммы, изображающие информацию на высоком уровне абстракции, позволяют людям отдалиться от деревьев и увидеть лес целиком.

Используя простые наброски, нарисованные от руки, или диаграммы, созданные с помощью программных средств моделирования, вы получаете возможность изменять проектное решение на концептуальном, а не физическом уровне. Модели не отображают мельчайших деталей реального продукта, но помогают визуально представить, как эти части сочетаются друг с другом. Эта ценная особенность моделирования делает его важным навыком для любого бизнес-аналитика или разработчика программного обеспечения (Wiegers, 2019а).

Один мой клиент запротестовал, когда я предположил, что его команде было бы полезно составить схему конкретных компонентов их проекта. «Наша система слишком сложна для моделирования», — заявил он. «Но позвольте, — возразил я, — модель по определению проще того, что она моделирует. Если сложность модели кажется вам непреодолимой, то как вы собираетесь справиться со сложностью задачи?» Диаграммы, безусловно, могут стать запутанными, когда речь идет о сложных системах. Однако сам этот факт является сильным аргументом в пользу использования методов, помогающих понять концептуальную сложность и управлять ею.

Как для выявления требований, так и для проектирования большое значение имеют рисование изображений, представляющих различные аспекты системы, и их последующее поэтапное уточнение.

Быстрые визуальные итерации

Как мы видели выше, пользовательские интерфейсы имеют два уровня проектирования: архитектурное и техническое. Просматривая экран с пользовательским интерфейсом, вы видите техническую часть с элементами визуального оформления, особенностями размещения текста, изображений, ссылок, полей ввода и элементов управления. Если нужна большая точность, то можно создать подробную структуру экрана или веб-страницы с помощью такого инструмента, как модель «отображение — действие — отклик» (Display-Action-Response, DAR) (Beatty, Chen, 2012). Однако поэтапное техническое проектирование пользовательского интерфейса требует изменения отдельных отображаемых элементов. Эти изменения могут быть утомительными, если не использовать эффективный инструмент создания экранов.

Архитектурное проектирование пользовательского интерфейса раскрывается через параметры навигации, представленные на каждом экране. Вы можете быстро уточнить проект архитектуры, нарисовав *диалоговую карту* (Wiegers, Beatty, 2013). Она представляет архитектуру пользовательского интерфейса в форме диаграммы состояний или диаграммы переходов. Каждый экран, который система демонстрирует пользователю, — это отдельное состояние, в котором может находиться система.

На рис. 3.4 показана часть упрощенной диалоговой карты сайта моей консалтинговой компании. Каждый прямоугольник представляет диалоговый элемент, с помощью которого пользователь взаимодействует с системой. Диалоговый элемент может быть веб-страницей, рабочей областью, меню, диалоговым окном, окном с сообщением и даже строкой подсказки. Стрелки на диалоговой карте указывают определенные пути перехода от одного элемента к другому. Стрелки можно подписывать, чтобы указать условия и/или действия, которые запускают переход. (Здесь я этого не сделал.) Представление пользовательского интерфейса на данном уровне абстракции не позволяет людям отвлекаться на детали оформление каждого элемента диалога. Благодаря этому они могут сосредоточиться на общей картине того, как пользователь будет взаимодействовать с системой в рамках решения задачи, последовательно используя диалоговые элементы.

Однажды я провел исследование с участием нескольких пользователей, чтобы выяснить, какая конкретная последовательность задач позволяет добиться наибольшей эффективности при работе в создаваемой системе. В одной руке у меня был маркер, а в другой — ластик.





Рис. 3.4. Диалоговая карта показывает возможные переходы между элементами, такими как веб-страницы

Я быстро набросал на доске возможный процесс навигации, рисуя на диалоговой карте прямоугольники и стрелки. Нас не интересовало, как могут выглядеть экраны, — нам достаточно было их названий и общего представления об их назначении. По мере того как участники исследования критиковали мою схему и предлагали изменения, я стирал какие-то фрагменты и рисовал другие. Постепенно мы пришли к общему пониманию оптимального навигационного процесса. Попутно мы также обнаружили некоторые ошибки и упущения в первоначальных замыслах. Такое итеративное моделирование — мощный инструмент, помогающий уточнить первоначальный проектный замысел.

Такие модели, как диалоговая карта, являются статическими. Диаграммы со стрелками и прямоугольниками довольно наглядно показывают, как пользователь будет выполнять задачу. Следующий уровень итераций улучшения дизайна пользовательского интерфейса — динамическое моделирование. На данном этапе вы моделируете набор экранных форм в подходящем инструменте (возможно, даже в Microsoft PowerPoint), чтобы создать более или менее реалистичную раскладовку пользовательского интерфейса. Такие модели позволяют более точно имитировать взаимодействие с пользователем, переходя в потоке задач от одного такого снимка экрана к другому. Переход к ди-

намическому моделированию перемещает итерации на один уровень абстракции ниже, на котором создается простой макет пользовательского интерфейса. Избирательное сочетание быстрого проектного моделирования, имитации взаимодействия с пользователем и прототипирования потребует меньше усилий, чем реализация всего пользовательского интерфейса с последующим поэтапным его изменением до тех пор, пока ваши пользователи не останутся довольны.

Итерации стали проще

Начав моделировать программные системы, я быстро сделал два открытия. Во-первых, мне требовалось выполнить несколько циклов, поскольку первая попытка никогда не давала идеального результата. Во-вторых, мне понадобились инструменты, упрощающие правку диаграмм. Если бы мне приходилось полностью перерисовывать диаграмму каждый раз, когда я задумывал что-то изменить, то я бы не вносил больше одного исправления.

Инструменты моделирования программного обеспечения стали обретать популярность в 1980–1990-х годах. Они упрощают редактирование диаграмм, например, позволяют перемещать стрелки вместе с объектами, к которым те прикреплены, при изменении положения или размера объектов. Инструменты поддерживают систему обозначений и синтаксис, принятые в некоторых видах анализа и проектирования. Они могут проверять диаграммы и указывать на возможные ошибки. Универсальные пакеты для построения диаграмм, такие как Microsoft Visio, теперь включают некоторые стандартные наборы символов для построения моделей программ. Однако в таких инструментах отсутствуют возможности проверки, придающие дополнительную ценность специализированным инструментам моделирования, а также нет возможности интеграции нескольких диаграмм и связанных с ними определений данных для всей системы.

Моделирование позволяет быстро исследовать несколько вариантов и разработать лучший проект, чем тот, который можно создать с первой попытки. Имейте в виду, что вам не нужно создавать идеальные модели. Вы также не должны моделировать всю систему — достаточно смоделировать только особенно сложные или неопределенные части. Инструменты построения диаграмм упрощают итеративное улучшение, но при этом легко попасть в бесконечный цикл попыток усовершенствовать модель. Такой аналитический паралич доводит итеративный подход до крайности и лишает его эффективности.



Визуальные модели — это средства коммуникации, способы представления знаний и обмена ими. Если мы собираемся общаться, то должны говорить на одном языке. Поэтому я настоятельно рекомендую при моделировании требований или проекта использовать общепринятые обозначения. Предлагаемую системную архитектуру можно смоделировать в виде простой блок-схемы, но элементы проекта более низкого уровня требуют применения специализированных символов. Наиболее популярным выбором для объектно-ориентированного проектирования является унифицированный язык моделирования (Unified Modeling Language, UML) (Page-Jones, 2000). Чтобы исследовать, улучшать и документировать свои идеи, а также делиться ими с другими, используйте готовый стандарт, такой как UML, а не изобретайте собственные обозначения, которые могут быть непонятны другим. Моделирование не может полностью заменить создание прототипов, но любой метод, упрощающий просмотр и изменение проекта на высоком уровне абстракции, поможет вам создавать более качественные продукты.

Урок 19

Разрабатывайте продукты так, чтобы их легко было использовать правильно и трудно — неправильно



Недавно я опробовал некоторые онлайн-калькуляторы прогнозирования продолжительности жизни. Многие такие калькуляторы упрощены и запрашивают всего несколько входных данных, а затем выдают расплывчатый прогноз. Я был рад найти калькулятор, который запрашивал не менее 35 единиц информации обо мне лично, семейном положении, болезнях и образе жизни. На сайте имелись раскрывающиеся списки, позволяющие выбирать конкретные значения из множества вариантов. Однако калькулятор имел одну небольшую проблему с дизайном пользовательского интерфейса (рис. 3.5).

The form consists of three main sections. On the left is a text input field labeled 'Ваш нынешний возраст' (Your current age). In the center is a horizontal row of two buttons: 'Рассчитать' (Calculate) above 'Сбросить' (Reset). On the right is another text input field labeled 'Ожидаемая продолжительность жизни' (Expected lifespan).

Рис. 3.5. В этой форме слишком легко нажать кнопку «Сбросить» вместо «Рассчитать»

Введя все данные, я нажал кнопку Рассчитать, чтобы узнать, сколько мне еще осталось. Однако оказалось, что вместо этого я случайно нажал кнопку Сбросить. Как показано на рис. 3.5, эти две кнопки имеют одинаковый стиль, плохо различимы и фактически соприкасаются друг с другом, что редко можно увидеть в пользовательском интерфейсе. Мало того, подсказка для запуска расчета появляется под кнопкой Сбросить, а не рядом с кнопкой Рассчитать, поэтому я инстинктивно нажал кнопку над подсказкой с текстом «Рассчитать». Когда я случайно нажал кнопку Сбросить, все введенные мною данные тут же исчезли. Мне пришлось начать процесс заново, так как меня все еще волновало мое будущее.

Пользователю этого сайта слишком легко ошибиться. Такие проблемы с дизайном раздражают. Может быть, я единственный пользователь, который когда-либо случайно нажал кнопку Сбросить, и тогда это моя проблема, а не сайта. Однако даже неформальное тестирование удобства использования могло бы выявить риск нажать не ту кнопку по ошибке. Три простых изменения могли бы улучшить этот дизайн.

1. Разместить кнопки Рассчитать и Сбросить подальше друг от друга и поместить подсказки рядом с соответствующими им кнопками, чтобы пользователь с большей вероятностью нажал нужную ему кнопку.
2. По-разному оформить кнопки Рассчитать и Сбросить, например, более рискованную кнопку Сбросить можно сделать меньше и окрасить ее в красный цвет, а кнопку Рассчитать — крупнее и окрасить в зеленый.
3. Защитить пользователя от случайности, попросив подтвердить выполнение разрушительного действия, такого как удаление всех введенных им данных, если вдруг он по ошибке нажмет не ту кнопку.

Хорошо продуманный пользовательский интерфейс делает продукт одновременно простым для правильного использования и трудным для неправильного. Подсказки и пункты меню четко описывают действия, при этом используется терминология, понятная предполагаемым пользователям. Дизайнер предоставляет варианты возврата, позволяющие пользователю вернуться к предыдущему экрану или запустить выполнение задачи с самого начала, желательно без повторного ввода уже предоставленной информации. Ввод данных осуществляется в логической последовательности. Значения для ввода в каждое поле

очевидно вытекают из оформления этих полей: с помощью раскрывающихся списков или из расположенных рядом текстовых инструкций.

Пользователи ценят системы, которые им понятны, предотвращают или исправляют их ошибки и взаимодействуют с ними, услужливо предоставляя ясные подсказки.

Эти свойства характерны для эффективного пользовательского интерфейса. Они помогают пользователям выполнять необходимые действия при работе с сайтом или приложением. Помимо удобства использования, дизайнеры также должны учитывать возможность ошибок и стараться предотвращать или помогать исправлять их. У дизайнёров есть четыре возможных варианта обращения с потенциальными ошибками (Wiegers, 2021).



1. Не дать пользователю возможности ошибиться.
2. Затруднить пользователю возможность ошибиться.
3. Упростить исправление допущенной ошибки.
4. Просто позволить ошибкам случиться. (Не поступайте так!)

Не давайте пользователю возможности ошибиться

Предотвращение ошибок — предпочтительная стратегия. Если пользователь должен вводить данные в изначально пустое поле ввода, то такое поле предполагает возможность ввода произвольных данных, которые программа обязательно должна проверить. Раскрывающиеся списки (или другие элементы управления) с допустимыми вариантами позволяют ввести только допустимые значения. Не давайте возможности ввести недопустимые варианты. Я видел раскрывающиеся списки для выбора даты окончания действия кредитной карты, которые включали годы, предшествующие текущему, что логически бессмысленно. Точно так же я видел элементы управления, позволяющие вводить несуществующие даты, например 30 февраля. Разрешение ввода неверных данных приведет к ошибке, когда приложение или веб-страница попытается обработать информацию.

Затрудните пользователю возможность ошибиться

Если нельзя лишить пользователя возможности ошибиться, то хотя бы затрудните ее. В примере с калькулятором ожидаемой продолжительности жизни, упомянутом выше, я предложил три способа уменьшить вероятность нажать по ошибке не ту кнопку. Другая хорошая практика — добавить поясняющий текст в диалоговых окнах, чтобы исключить двусмысленность в понимании реакции системы на каждый выбор. Не заставляйте пользователя вводить одну и ту же информацию дважды, потому что это удваивает вероятность ошибиться и занимает вдвое больше времени. Например, если форма запрашивает два адреса — доставки и выставления счета, — то дайте пользователю возможность указать, что они совпадают, установив флажок.

Упростите исправление допущенной ошибки

Несмотря на все ваши усилия, ошибки (допущенные пользователем или системой во время работы) все равно будут возникать. Предусмотрите для пользователя простую возможность исправлять такие ошибки. Простота исправления — характеристика устойчивой программной системы, а *устойчивость* — атрибут качества, описывающий, насколько хорошо продукт справляется с неожиданными входными данными, событиями и условиями работы. Особенно полезны в этом отношении многоуровневая функция отмены/возврата и четкие, содержательные сообщения, помогающие пользователю исправить любые ошибки. Загадочные числовые коды ошибок HTTP, запросов к базе данных или сетевых сбоев могут помочь при технической диагностике, но бесполезны для обычного пользователя.

Просто позвольте ошибкам случиться

Наименее желательный вариант — просто позволить ошибкам случиться и заставить пользователя самому разбираться с последствиями. Предположим, пользователь просит запустить некую процедуру, имеющую определенные предварительные условия, которые должны быть выполнены. Программное обеспечение должно проверять такие предварительные условия и помогать пользователю выполнить их, если это необходимо, а не просто продолжать работать в надежде на лучшее. Более того, оно вообще не должно запускать запрошеннную процедуру,



если предварительные условия не выполнены. Система должна сама обнаруживать невыполненные предварительные условия и сообщать о них как можно раньше, чтобы не тратить время пользователя впустую. Пользователи ценят системы, которые им понятны, предотвращают или исправляют их ошибки и взаимодействуют с ними, усердно предоставляя ясные подсказки.

Кстати, калькулятор ожидаемой продолжительности жизни, который я попробовал, показал, что я, вероятно, проживу еще несколько лет. Это была хорошая новость, даже притом что на его использование ушло в два раза больше моего времени, чем предполагалось, из-за далеко не идеального дизайна пользовательского интерфейса.

Урок 20

Невозможно оптимизировать все желаемые атрибуты качества

Следующее приложение, которое я хотел бы получить, не должно иметь ошибок: никаких ошибок 404 «Страница не найдена» и никаких экранов со справкой, не соответствующих форме, с которой я работаю. Приложение не должно использовать много памяти или замедлять мой компьютер и обязано освободить всю использованную память по завершении. Приложение должно быть полностью безопасным: никто не должен иметь возможности украсть мои данные или выдать себя за меня. Оно должно мгновенно реагировать на каждую мою команду и быть абсолютно надежным. Я не хочу видеть никаких сообщений «Внутренняя ошибка сервера» или «Приложение не отвечает». Пользовательский интерфейс никогда не должен позволять мне ошибаться. Приложение должно позволять работать с ним на любом устройстве, мгновенно загружаться, не прерывать работу по тайм-ауту, импортировать и экспортировать любые данные из других источников. Ах да, чуть не забыл — приложение должно быть бесплатным.

Похоже на сказочное приложение? Верно! Мои желания разумны? Конечно, нет!

Невозможно объединить в одном приложении все лучшие достижения и передовые возможности. Различные качественные характеристики неизбежно вступают в конфликт друг с другом: улучшение одной часто приводит к ухудшению другой. Следовательно, важной частью анализа требований является определение наиболее важных характеристик, чтобы дизайнеры могли учесть это.

Измерения качества

Команды разработчиков программного обеспечения при изучении требований должны учитывать широкий набор атрибутов качества. Атрибуты качества также называются *факторами качества и требованиями к качеству обслуживания*. Термины *Design for Excellence* и *DfX* (проектирование с целью добиться наилучших характеристик) также относятся к атрибутам качества, где X — это свойство, которое разработчики стремятся оптимизировать (Wikipedia, 2021а). Говоря о нефункциональных требованиях, люди обычно имеют в виду атрибуты качества.

Нефункциональные требования не реализуются напрямую в программном или аппаратном обеспечении. Они служат лишь источником производной функциональности, архитектурных решений или подходов к проектированию и реализации. Некоторые нефункциональные требования ограничивают выбор вариантов, доступных дизайнеру или разработчику. Например, требование функциональной совместимости может ограничивать разработку продукта тем, что приходится использовать определенные стандартные интерфейсы.

Я видел списки из более чем 50 атрибутов качества, организованных в группы и иерархии. Далеко не во всех проектах приходится беспокоиться о таком их количестве. В табл. 3.1 перечислены атрибуты качества, которые каждая команда разработчиков программного обеспечения должна учитывать, изучая значение понятия качества для их продукта (Wiegers, Beatty, 2013). Физические продукты, содержащие встроенное программное обеспечение, имеют некоторые дополнительные атрибуты качества, например перечисленные в табл. 3.2 (Koopman, 2010; Sas, Avgeriou, 2020).

Таблица 3.1. Некоторые важные атрибуты качества программных систем

Атрибут качества	Основной вопрос, на который отвечает атрибут
Доступность	Смогу ли я использовать систему, когда и где мне нужно?
Соответствие стандартам	Соответствует ли система всем применимым стандартам в отношении функциональности, безопасности, связи, сертификации и интерфейсов?
Эффективность	Экономно ли система использует ресурсы компьютера?

Таблица 3.1 (окончание)

Атрибут качества	Основной вопрос, на который отвечает атрибут
Возможность установки	Смогу ли я легко установить, удалить и переустановить систему и ее обновления?
Целостность	Имеет ли система защиту от неточных данных, от их повреждения и потери?
Совместимость	Достаточно ли хорошо система взаимодействует с окружением для обмена данными и услугами?
Сопровождаемость	Смогут ли разработчики легко менять, исправлять и улучшать систему?
Производительность	Достаточно ли быстро система реагирует на действия пользователя и внешние события?
Переносимость	Можно ли легко перенести систему на другие платформы?
Надежность	Отсутствуют ли сбои в работе системы?
Повторное использование	Смогут ли разработчики повторно использовать части системы в других продуктах?
Устойчивость	Реагирует ли система на ошибочные входные данные и непредвиденные условия работы?
Безопасность	Защищает ли система пользователей от вреда и имущество — от повреждения?
Масштабируемость	Может ли система легко расширяться для обслуживания большего количества пользователей, данных или транзакций?
Зашщищенность	Защищена ли система от атак вредоносных программ, злоумышленников, неавторизованных пользователей и кражи данных?
Удобство использования	Смогут ли пользователи быстро научиться работать с системой и эффективно выполнять свои задачи?
Проверяемость	Смогут ли тестировщики определить, насколько правильно реализовано программное обеспечение?

По аналогии с функциональностью разработчики должны сбалансировать ценность некоего качества и стоимость его достижения.

Таблица 3.2. Некоторые дополнительные атрибуты качества физических продуктов со встроенным программным обеспечением

Атрибут качества	Основной вопрос, на который отвечает атрибут
Долговечность	Сохранится ли работоспособность продукта в нормальных условиях эксплуатации?
Расширяемость	Можно ли с легкостью добавить в продукт новые функции, датчики или другое оборудование, не нарушая его функционирования?
Обработка ошибок	Способен ли продукт обнаруживать, исправлять и регистрировать возникающие ошибки?
Технологичность	Является ли продукт простым и рентабельным в производстве?
Потребление ресурсов	Сохраняет ли продукт достаточный резерв потребляемых ресурсов: памяти, пропускной способности сети, процессорного времени, потребляемой мощности электропитания и т. д.?
Удобство обслуживания	Смогут ли люди эффективно выполнять профилактическое обслуживание продукта?
Устойчивое развитие	Оказывает ли продукт хотя бы минимальное неблагоприятное воздействие на окружающую среду в течение своего жизненного цикла — от добычи сырья до производства, использования и утилизации?
Возможность обновления	Можно ли с легкостью улучшить продукт, добавив или заменив компоненты?

В этих двух таблицах отсутствует одно важное свойство — стоимость. По аналогии с функциональностью, разработчики должны сбалансировать ценность некоего качества и стоимость его достижения. Например, всем хотелось бы, чтобы программное обеспечение всегда было доступно для использования, но достижение этого может стоить очень дорого.



У одного из моих клиентов была компьютерная система управления производством с требованием доступности 24/7 с допустимым нулевым временем простоя. Разработчики выполнили это требование, добавив резервную систему. Для обновления программного обеспечения они сначала устанавливали все необходимое в автономной системе, выполняли тестирование, переключали эту систему в оперативный режим, а затем обновляли вторую систему. Иметь две



независимые компьютерные системы — дорогое удовольствие, но это дешевле, чем останавливать производство продукта, когда система управления выходит из строя.

Определение атрибутов качества



Разработчики должны знать, какие атрибуты качества наиболее важны, какие аспекты этих часто многомерных атрибутов имеют первостепенное значение, а также целевые характеристики. Недостаточно просто сказать: «Система должна быть надежной» или «Система должна быть удобной для пользователя». На этапе выявления требований бизнес-аналитик должен выяснить, что именно заинтересованные стороны имеют в виду под надежностью или удобством. По каким характеристикам можно судить о надежности и удобстве системы? Какие примеры ненадежности или неудобства можно привести?

Чем точнее бизнес-аналитик сформулирует ожидания заинтересованных сторон в отношении качества, тем легче разработчикам будет сделать правильный выбор и оценить достижение целевых показателей. Roxanne Miller (Roxanne Miller, 2009) приводит множество примеров четко сформулированных требований к атрибутам качества в многочисленных категориях. Когда это возможно, формулируйте цели в области качества измеримыми и проверяемыми способами. Рассмотрите возможность использования Planguage — языка ключевых слов, позволяющего количественно определить такие расплывчатые атрибуты, как доступность и производительность (Simmons, 2001; Gilb, 2005). Для такой тщательной проработки требований нужно время, но оно будет потрачено с пользой по сравнению с переделкой продукта после того, как он не оправдает ожиданий клиентов.

Проектирование для качества

Разработчики могут оптимизировать свой подход к решению для практически любого параметра качества в зависимости от степени его важности. Без руководства со стороны один разработчик может оптимизировать производительность, другой — удобство использования, а третий — переносимость между платформами. Чтобы этого не происходило, на этапе исследования требований к проекту нужно определить наиболее важные атрибуты, чтобы направлять усилия разра-

ботчиков туда, где они наиболее важны для успеха бизнеса. А это означает, что необходимо расставлять приоритеты для нефункциональных требований по аналогии с функциональными.

Приоритизация нужна для согласования компромиссов между определенными парами атрибутов качества. Увеличение одного атрибута часто требует от разработчика пойти на компромисс в некоторых других областях (Wiegers, Beatty, 2013). Приведу несколько примеров конфликтов атрибутов качества, которые требуют компромиссных решений.



- Многофакторная аутентификация более надежна, чем простой пароль для входа, но снижает удобство использования из-за дополнительных шагов и, возможно, задействованных устройств.
- Продукт или компонент, предназначенный для повторного использования, может оказаться менее эффективным, чем если бы код, реализующий необходимую функциональность, был оптимизирован для одного приложения. При условии приемлемости потери производительности разумнее отдать предпочтение повторно используемым компонентам.
- Оптимизация системы в целях повышения производительности может ухудшить ее переносимость, если разработчики будут использовать определенные свойства операционной системы или языка, чтобы достичь максимальной производительности.
- Оптимизация одних аспектов сложного атрибута качества может привести к ухудшению других. Например, внедрение решений, упрощающих обучение новых или случайных пользователей, может сделать систему неудобной для экспертов.

В то же время некоторые пары атрибутов качества создают эффект синергии. Проектирование системы с учетом высокой надежности способствует улучшению таких характеристик, как:

- доступность (если система не дает сбоев, то остается доступной для использования);
- целостность (снижается риск потери или повреждения данных из-за сбоя);
- устойчивость (меньше вероятность отказа продукта из-за неожиданного действия пользователя или состояния окружающей среды);
- безопасность (если механизмы безопасности продукта работают надежно, то никто не пострадает).



Взаимозависимость атрибутов качества наглядно показывает, почему проектная группа должна заранее выяснить, что означает понятие качества для основных заинтересованных сторон, и направить работу каждого на достижение этих целей. Заинтересованные стороны, не обсуждающие с бизнес-аналитиками эти вопросы, оказываются зависимыми от догадок и предположений разработчиков. Если вы не проанализируете нефункциональные требования в процессе сбора информации и не определите их, то вам очень повезет, если разработчики встроят функции, ценные для клиентов.

Архитектура и атрибуты качества

Чтобы выбрать правильную архитектуру, команда разработчиков должна заранее понять, какие атрибуты требуют пристального внимания. Архитектура влияет на множество атрибутов, в том числе доступность, эффективность, функциональную совместимость, производительность, переносимость, надежность, безопасность, масштабируемость и защищенность. Поскольку необходимость компромиссов — частое явление, архитекторы должны знать, какие атрибуты наиболее важны, иначе не смогут принять решения, ведущие к желаемым результатам.

Возвращение на поздних стадиях разработки или после выпуска и переделка архитектуры системы в рамках исправления недостатков качества — дорогое удовольствие. Поэтапное построение систем при отсутствии предварительного знания наиболее важных целей в области качества может привести к проблемам, которые трудно исправить, особенно если речь идет не только о программном, но и об аппаратном обеспечении. Как это часто бывает с программными проектами, потратив чуть больше времени на то, чтобы лучше понять цели в области качества, можно прийти к менее дорогим и более надежным решениям.

Урок 21

Проблемы легче предупредить, чем исправить



Начиная писать свою первую книгу 25 лет назад, я плохо понимал, что делаю. Я начал с комически скучного наброска; в первоначальной структуре моей книги было много ошибок. Под руководством чрезвы-

чайно терпеливого редактора (спасибо, Венди!) я изменил структуру рукописи, сделав ее более читабельной. На копирование, исправление и приглаживание фрагментов текста ушел целый месяц. Ценность содержимого не увеличилась, но подача значительно улучшилась.

Этот болезненный опыт крепко врезался мне в память. С тех пор как я начал вкладывать гораздо больше сил в организацию книги на архитектурном и детальном уровнях, мне никогда не приходилось совершать крупных переделок, разве что незначительно менять последовательность изложения. Благодаря этому я могу сосредоточиться на содержании, а не структуре. Как мы видели в уроке 18 «Чем выше уровень абстракции, тем проще выполнять итерации», перемещать элементы в макете книги гораздо проще, чем реорганизовывать и переписывать предложения.



Тот же урок применим к проектированию программного обеспечения. Время, потраченное на вдумчивое рассмотрение проекта, с лихвой окупается временем, которое не было потрачено на исправление проблем позже. Вам может понадобиться дополнительное время на то, чтобы усовершенствовать проект в условиях неопределенности, поэтому старайтесь распределять усилия в соответствии с характером проблемы. Даже приложив максимум усилий для создания безупречного проектного решения, позднее вы можете обнаружить недостатки и вам придется их корректировать.



Технический долг и рефакторинг

Проекты, выполненные в спешке, могут привести к появлению *технического долга* — так называют недостатки, которые кто-то должен устраниить в будущем, чтобы обеспечить надлежащее функционирование продукта и его расширяемость. (Подробнее о техническом долге см. в уроке 50 «Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения».) Небольшой технический долг может быть приемлемым компромиссом, если экономия на проектировании и разработке кода ускоряет достижение текущей бизнес-цели. Однако недостатки остаются. Чем дольше команда откладывает их решение, тем более масштабной, дорогостоящей и разрушительной будет доработка. Как и любой кредит, технический долг следует рассматривать как временный и требующий постепенного погашения.



Доработка в целях уменьшения технического долга часто принимает форму *рефакторинга* — процесса реорганизации существующего кода для улучшения его структуры без изменения функциональности. Вы можете упростить код, сделать его более удобным для сопровождения или расширения, повысить его эффективность, удалить повторяющиеся и ненужные части или внести другие улучшения.

Существенные изменения проекта могут потребовать значительных усилий по реорганизации, тогда как команды обычно предпочитают создавать новые полезные функции. Мне становится не по себе, когда я вижу, что приставка *re-* (пере-) используется слишком часто. Она означает, что мы снова делаем то, что уже однажды делали.

Внесение изменений в проект требует усилий, притом что уровень ценности для клиента остается прежним, но этот труд помогает поддерживать стабильную основу, необходимую для дальнейшего развития продукта. Хорошее проектирование сводит к минимуму возникающий технический долг, а рефакторинг устраниет его. Разумный баланс этих двух факторов дает наилучшие результаты. Недостаточное проектирование может привести к значительным доработкам; на слишком детальную проработку проекта может потребоваться через чур много времени, при этом хороший результат не гарантирован. Два высказывания экспертов по проектированию раскрывают эту двойственность:

Постоянное совершенствование проектного решения упрощает работу с кодом. Однако на практике обычно выбирается другой путь: небольшой рефакторинг и большое внимание к целесообразности добавления новых функций (Kerievsky, 2005).

Продумать или узнать все в начале проекта практически невозможно... Однако вы можете использовать свой опыт и опыт других, чтобы выбрать определенное направление. Сегодня вы можете принимать решения, которые завтра минимизируют потребность внесения изменений (Pugh, 2005).

Как указывает Кен Пью в предыдущей цитате, цель проектирования состоит в том, чтобы принять разумные решения сейчас и предотвратить ненужные изменения в будущем. Опираясь на свои рассуждения и мнение заинтересованных сторон, выбирайте такие решения, которые уменьшают вероятность появления необходимости изменять те или иные части продукта в будущем.

Накопление технического долга из-за того, что у команды нет времени на надлежащее проектирование, просто отодвигает проблему в будущее.

Архитектурные недостатки

Внесение небольших изменений в проект по мере продвижения — не слишком болезненный процесс, при этом продукт непрерывно и постепенно улучшается. Серьезная реорганизация архитектуры ради увеличения устойчивости продукта или улучшения пользовательского опыта дастся гораздо сложнее.

Примером несовершенного архитектурного проектирования, влияющего на опыт пользователя, может служить разнообразие способов удаления единицы информации из смартфона. Действия пользователя, подсказки и значки различаются в зависимости от того, что удаляется: текстовое или почтовое сообщение, сохраненное местоположение на карте, фотография, заметка, событие календаря, будильник, контакт, пропущенный телефонный звонок или целое приложение. Одни операции удаления требуют подтверждения, другие — нет. Иногда процесс может немного меняться в зависимости от того, удаляется один экземпляр объекта или несколько. Эти отличия могут сбивать пользователя с толку.

Дизайнеры могли бы избежать многих несоответствий, применяя в работе общие стандарты организации пользовательского интерфейса и дизайна архитектуры в целом, как, например, повторное использование некоего кода. Повторное использование — отличный способ улучшить качество, повысить производительность разработчиков и сократить время обучения пользователя. Унификация операций удаления на столь поздней стадии развития продукта потребовала бы чрезмерного объема работы. И это только для одной операции, которая присутствует в той или иной форме почти в каждой программной системе.

Разработчики ПО всегда создают проект либо в процессе работы, либо в ходе тщательного обдумывания. Накопление технического долга из-за того, что у команды нет времени на надлежащее проектирование, просто отодвигает проблему в будущее, где ее влияние продолжает расти. Инвестирование в проектное решение (то, что Кен Пью называет *prefactoring* (Pugh, 2005)) позволяет сэкономить силы и время на реструктуризацию и повторную реализацию в будущем, когда предпочтительнее работать над чем-то другим.



Урок 22**Проблемы многих систем скрываются в интерфейсах**

Простейшая программная система состоит из единственного модуля кода и интерфейса для пользователя. *Интерфейс* описывает, как пересекаются два архитектурных элемента внутри многокомпонентной системы или системы с внешним окружением. Некоторые интерфейсы должны соответствовать установленным стандартам и протоколам, например интерфейсы для подключения оборудования или включения модуля из библиотеки. Другие интерфейсы характерны для конкретного приложения.

Любая крупная программная система состоит из множества модулей и имеет массу внутренних интерфейсов, связывающих компоненты системы, которые вызывают друг друга для получения какого-либо сервиса. Система также может предоставлять внешние интерфейсы пользователям-людям, другим программным системам, аппаратным устройствам, сетям и операционной системе компьютера, как показано на рис. 3.6. Продукты, содержащие и аппаратные, и программные компоненты, создают дополнительные сложности с точки зрения интерфейса.

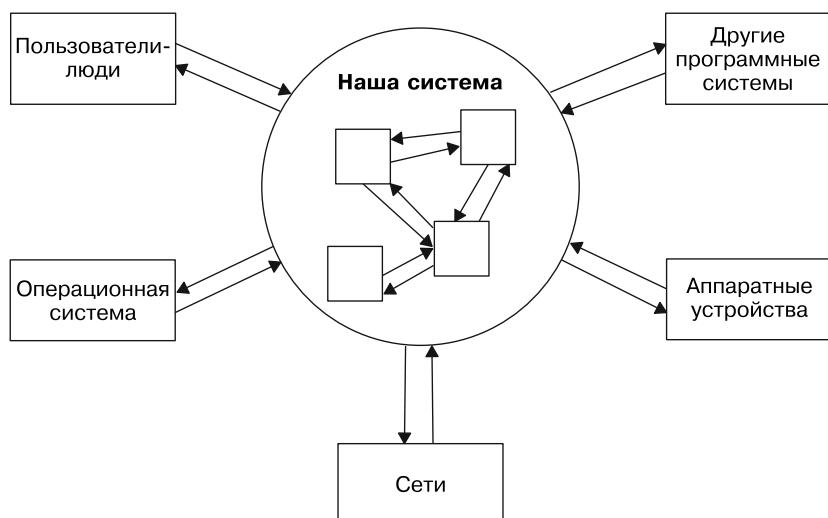


Рис. 3.6. Программные системы имеют внутренние интерфейсы между своими компонентами и внешние — для связи с другими объектами

Внутренние и внешние интерфейсы — распространенные источники проблем. Например, многократно используемые библиотеки с плохо или неправильно описанными в документации интерфейсами могут увеличить время разработки, поскольку разработчикам придется потратить время и силы, чтобы правильно интегрировать компоненты в свою систему. Добросовестный дизайнер гарантирует правильное сочетание всех частей сложной системы через их взаимные интерфейсы. Новые компоненты, которые разработчики интегрируют в существующую систему, должны соответствовать установленным соглашениям об интерфейсе.

Технические проблемы с интерфейсами

Интерфейс определяет контракт или соглашение о том, как два архитектурных элемента (один запрашивающий, а другой предоставляющий сервис) подключаются друг к другу для обмена данными, сервисами или и тем и другим. Каждый элемент имеет четко очерченные границы и набор функций или сервисов, которые он предоставляет. Определение интерфейса включает в себя нечто большее, чем простое назначение порядка вызова операций через него. Полное описание интерфейса содержит множество элементов (Pugh, 2005; Rozanski, Woods, 2005):



- синтаксис (тип и формат) и семантику (смыслоное значение) запросов и ответов, проходящих через интерфейс, в том числе данные и управляющие воздействия;
- ограничения на типы данных или значения, которые могут передаваться через интерфейс;
- механизм или протоколы, на основе которых действует интерфейс, например обмен сообщениями или вызов удаленных процедур;
- предварительные условия, которые должны быть выполнены в начале взаимодействия через интерфейс;
- постусловия, которые должны быть выполнены обеими сторонами после взаимодействия как в случае успеха, так и при неудаче.

Неясность обязанностей компонентов с обеих сторон, использующих общий интерфейс, может приводить к проблемам. Функциональность может дублироваться или вообще отсутствовать, поскольку люди, работающие над одной частью программы, думали, что эта функция

будет реализована в другом компоненте. Архитектурные компоненты всегда должны неукоснительно следовать установленным интерфейсам. Например, один модуль никогда не должен пытаться получить доступ к коду или данным в другом модуле, кроме как через взаимный интерфейс.



Каждая реализация интерфейса должна соответствовать определению контракта (Pugh, 2005). Кроме того, реализация не должна причинять вред, например потреблять избыточное количество памяти или без необходимости блокировать доступ к объектам данных. Вдобавок дизайн должен предусматривать обработку ошибок интерфейса. Если реализация интерфейса по какой-либо причине не сможет выполнить свои обязанности, то она должна вернуть соответствующее уведомление, чтобы помочь вызывающей стороне исправить ошибку.



Недавно я на своем iPad начал читать электронную книгу, которую скачал в библиотеке. Я много раз пытался скачать файл для автономного доступа, используя предназначенную для этого кнопку. Загрузка начиналась, но затем появлялось неинформативное сообщение об ошибке, показанное на рис. 3.7. Судя по всему, в интерфейсе между



Рис. 3.7. Это сообщение не помогло мне исправить ошибку интерфейса между моим iPad и сервером электронных книг

моим iPad и сервером, на котором размещена электронная книга, происходил воспроизводимый сбой, о чем и сообщало программное обеспечение. Но из этого сообщения было непонятно, в чем проблема и как ее исправить. Я так и не смог скачать эту электронную книгу, как и другие, к которым пытался получить доступ таким же способом.

Дизайнеры должны тщательно спланировать и изучить внутренние и внешние интерфейсы системы, чтобы предотвратить подобные неудобства для пользователей. Сложные системы, состоящие из множества взаимосвязанных компонентов, сложно модифицировать. Изменение одного из интерфейсов может запустить каскад изменений во множестве компонентов, использующих его. Если система разрабатывается без четко определенных интерфейсов между компонентами, то по мере добавления новых возможностей, требующих изменения интерфейса, технический долг может быстро накапливаться. Проблемы также могут возникнуть, если новая функциональность не соответствует существующим интерфейсам.

При разработке интерфейса принято начинать со всего, что, по мнению дизайнера, может понадобиться пользователю — человеку или другой системе. Но при таком подходе интерфейсы получаются переполненными функциями, которые пользователи никогда не будут вызывать.

Предпочтительнее использовать дизайн, ориентированный на запрашивающую сторону, задав вопрос: «Какие функции действительно понадобятся пользователям моего интерфейса?» Написание тестов перед реализацией поможет продумать порядок использования интерфейса и добавить в него только необходимые возможности, убрав ненужные элементы. Понимание задач, решаемых пользователями с помощью программного обеспечения, также способствует созданию оптимизированного интерфейса.

Постарайтесь предугадать вероятные изменения, которые разработчики будут вносить в систему с течением времени, и подумайте, как они могут повлиять на интерфейсы. Это особенно пригодится при использовании итеративных методологий разработки. Благодаря приоритету запланированных улучшений разработчики будут знать о тех частях системы, которые с большей вероятностью изменятся, и о тех, которые должны оставаться более стабильными. Выбор правильного архитектурного решения с самого начала способствует устойчивому росту продукта и частым выпускам (Scaled Agile, 2021b).



Проверка входных данных



Каждый компонент, участвующий во взаимодействии, должен проверять получаемые входные данные перед их обработкой. Многие уязвимости безопасности обусловлены тем, что вредоносный код, вне-дремый злоумышленником через интерфейс, не отклоняется как недопустимый ввод. В журнале ошибок моего сайта иногда появляются сообщения о том, что пользователь пытался получить доступ к сайту, используя неверные входные данные. К счастью, мой хостинг-провайдер выявляет такие попытки и блокирует их. Microsoft (2017) рекомендует некоторые методы проверки пользовательского ввода в целях защиты от подобных атак вредоносного ПО. Соблюдение стандартов безопасного программирования и использование инструментов для сканирования уязвимостей в интерфейсах тоже снижают риск появления уязвимостей безопасности в системах (SEI, 2020).

На рис. 3.8 представлена схема, которая поможет вам оценить поведение интерфейса. Следование правильной стратегии проектирования интерфейса по контракту гарантирует, что ваши компоненты будут находиться в двух квадрантах внутри жирной пунктирной линии.

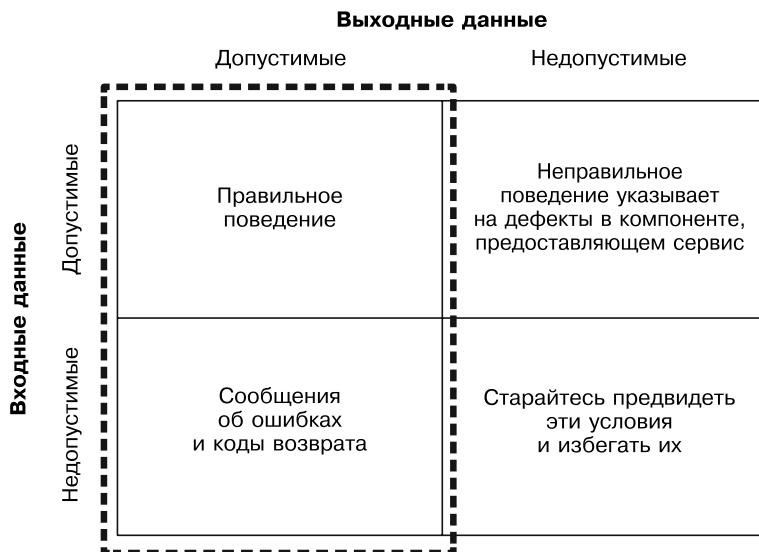


Рис. 3.8. Проверка допустимости входных и выходных данных позволяет оценить правильность поведения интерфейса (схема любезно предоставлена Гари К. Эвансом)

Каждый компонент, участвующий во взаимодействии, должен проверять получаемые входные данные перед их обработкой.

Хорошо спроектированная система будет правильно обрабатывать исключения, возникающие во внутренних и внешних интерфейсах. Недавно я пытался напечатать документ со своего ПК с Windows на принтере, подключенном к домашней сети Wi-Fi. Принтер был включен и подключен к сети, но мой компьютер сообщал мне, что принтер отключен. Мне пришлось перезагрузить компьютер, после чего он правильно обнаружил принтер как подключенный к сети и отправил задание на печать. По всей видимости, какая-то необработанная ошибка, возникшая в интерфейсе между ПК и принтером, не позволяла установить соединение между ними, и у меня не было никакой другой возможности исправить ее, кроме как перезагрузить компьютер.



Проблемы с пользовательским интерфейсом

Пользователей интересует не внутренняя архитектура системы, а ее интерфейс. Продукты, имеющие недостатки интерфейса, пользователи считают недоработанными. Непоследовательное поведение интерфейса смущает и расстраивает пользователей, как это было показано в предыдущем уроке на примере различных операций удаления в смартфонах. Из-за плохо спроектированного интерфейса продукты получаются сложными в использовании, вынуждают тратить время впустую, допускают ошибки и плохо работают в реалистичных сценариях использования (Wiegers, 2021).

Определение стандартов интерфейса помогает обеспечить согласованное взаимодействие с пользователем как в одном приложении, так и в комплексе из нескольких связанных приложений. Когда я руководил небольшой группой разработчиков программного обеспечения, мы приняли рекомендации по организации пользовательского интерфейса, предназначенного для применения внутри компании. Эти рекомендации помогли всем нашим приложениям выглядеть и вести себя одинаково. Наши пользователи могли определить по интерфейсу, что приложение создано нашей группой, но из-за единства стиля они не могли сказать, кто из членов команды разработал тот или иной интерфейс.



Хорошо спроектированные интерфейсы не должны требовать вспомогательной документации в виде справочных экранов и руководств пользователя (Davis, 1995). Они облегчают пользователям процесс освоения нового приложения и оберегают их от ошибок. В настоящее время существует огромное количество литературы по проектированию пользовательского опыта. Одна из книг, посвященных этой теме, — классический труд Алана Купера (Alan Cooper) и других соавторов *About Face¹* (2014). Любой разработчик получит пользу от эвристик, предложенных экспертом в области проектирования пользовательских интерфейсов Якобом Нильсеном (Jakob Nielsen) (Nielsen, 2020). Ориентируясь на сценарии использования, а не на функции продукта, вы сможете избежать многих проблем с пользовательским опытом. (См. урок 4 «В требованиях в первую очередь важны особенности использования, а затем — функциональность».)



Сложности с интерфейсами

Иногда проблемы с интерфейсом обнаруживаются, когда команды пытаются интегрировать свои модули в продукт. Сбои при интеграционном тестировании после объединения нескольких модулей могут вызвать желание найти и наказать виноватых. Это нездоровий конфликт. Если архитектура правильно структурирована, интерфейсы хорошо определены, разработчики неукоснительно следуют правилам, определяемым интерфейсами, а модули успешно проходят созданные для них тесты, то интеграция должна протекать гладко и без обвинений.



СЛЕДУЮЩИЕ ШАГИ: ПРОЕКТИРОВАНИЕ

1. Определите, какие уроки, описанные в этой главе, имеют отношение к вашему опыту работы с различными аспектами разработки программного обеспечения.
2. Можете ли вы, опираясь на свой опыт, вспомнить какие-либо другие уроки, связанные с проектированием, которыми стоит поделиться с коллегами?
3. Перечислите описанные в этой главе методы, способные помочь в решении проблем проектирования, которые мы определили во врезке «Первые шаги» в начале главы. Как каждый метод может улучшить

¹ Купер А., Носсел К., Кронин Д. Интерфейс. Основы проектирования взаимодействия. — СПб.: Питер, 2022.

работу по проектированию, выполняемую вашими проектными группами?

4. Как бы вы определили, приносит ли желаемые результаты каждый метод, названный на шаге 3? Насколько ценные для вас эти результаты?
5. Определите любые препятствия, которые могут затруднить применение методов, перечисленных на шаге 3. Как бы вы справились с ними? Заручились бы поддержкой коллег, готовых помочь вам в реализации этих методов?
6. Внедрите руководящие документы, контрольные списки и другие инструменты, чтобы помочь будущим проектным группам эффективно применять ваш передовой опыт проектирования.

Глава 4

Управление проектами

ВВЕДЕНИЕ

Институт управления проектами (Project Management Institute, PMI) определяет *проект* как «временную попытку создать уникальный продукт, услугу или результат» (PMI, без даты), а *управление проектами* — как «применение знаний, навыков, инструментов и методов к проектной деятельности для удовлетворения требований проекта».



Эти определения звучат разумно, но я не думаю, что управление проектами является отдельной дисциплиной. Управление проектами предполагает управление многими действиями, которые в совокупности способствуют успеху проекта. Эти действия могут выполнять один или несколько руководителей (если речь идет о крупном проекте), все вместе или распределяя обязанности между собой. Например, в проекте, использующем методологию Scrum, различные обязанности по управлению проектом распределяются между скрам-мастером (Scrum Master), владельцем продукта и другими членами команды. В этой книге под словами «руководитель проекта» я подразумеваю любого, кто участвует в деятельности по управлению проектами от начала до успешного завершения, независимо от должности или других обязанностей.

Управление проектами предполагает управление многими действиями, которые в совокупности способствуют успеху проекта.

В этом введении описаны некоторые сферы управления проектами, и часть из них рассматривается более подробно в уроках данной главы. Даже если вы не являетесь специалистом в области управления проектами, то все равно так или иначе управляете своей работой и несете долю ответственности за выполнение проекта. Поэтому большая часть этой главы применима как к отдельным лицам, так и к проектным группам.

Управление персоналом

Проекты выполняются людьми: одним человеком, небольшой группой людей, находящихся в одном месте, или сотнями сотрудников, работающих над несколькими подпроектами в разных местах. Управлять проектом — значит определять, что и когда должно быть сделано, какой набор навыков для этого необходим, определять людей, которые будут играть различные роли в проекте, а также привлекать их в нужное время. После того как люди войдут в состав команды разработчиков проекта, ими нужно управлять, руководить и, возможно, обучать их. Иногда может возникнуть необходимость заменить кого-то из-за их занятости в других проектах или несоответствия выполняемой работе.

Управление требованиями

Как мы видели в главе 2, все проекты имеют требования. Большинство требований описывают сам продукт; другие — работу по проектированию, например подготовку учебных материалов или переход на новую систему. В одних случаях проекты начинаются с уже готовым базовым набором требований. В других разработка требований является постоянной частью проектной работы. В последнем случае в команде должны быть люди, имеющие опыт работы с требованиями, а также должно выделяться время, необходимое для выполнения требований. Корректировка планов проекта, позволяющая команде сосредоточиться на реализации необходимых возможностей и характеристик, является основной деятельностью по управлению проектом.



Управление ожиданиями

Важная часть управления проектом — формирование реалистичных ожиданий, которые понимают и принимают заинтересованные сторо-

ны. В число этих ожиданий входят свойства решения, которое предоставит проектная группа, и параметры доставки. Параметры доставки включают в себя график передачи промежуточных и окончательных версий, стоимость, качество, требуемые ресурсы и ограничения — то есть то, что не будет содержать решение. Откровенные обсуждения помогают участникам проекта знать, чего от них ожидают другие, что облегчает обсуждение взаимоприемлемых и достижимых обязательств (Karten, 1994).

Некоторые заинтересованные стороны могут быть недовольны конкретными ожиданиями. Однако руководители проектов должны быть реалистами, а не фантазерами, обещающими идеальные, но недостижимые результаты. Когда ситуация в проекте меняется, руководителям проектов необходимо обсудить скорректированные ожидания с соответствующими заинтересованными сторонами.

Управление задачами

Достижение ожидаемой ценности проекта требует, чтобы многие задачи выполнялись правильно и в верной последовательности. Некоторые задачи нельзя начать или завершить, не начав или не завершив другие. Участникам необходимо определить задачи с соответствующим уровнем детализации, выделить ресурсы для каждой из них и объединить наборы задач в контрольные точки, достижение которых свидетельствует о прогрессе. Руководитель проекта должен организовать работу для достижения целей проекта в кратчайшие сроки и с наименьшими затратами, избегая неоправданных задержек. Управлять задачами — значит постоянно жонглировать процессами и ресурсами.

Управление обязательствами

Руководители проектов принимают обязательства перед своими клиентами, руководителями и членами своей команды. Точно так же другие участники принимают обязательства перед руководителем проекта и друг перед другом. Обязательства часто основываются на оценках, которым присуща неопределенность. Как заметил один опытный руководитель проекта, «оценки со временем могут меняться». Каждый должен брать на себя обязательства, основываясь на доступной в данный момент информации. Все должны отслеживать прогресс в выполнении этих обязательств, честно корректировать их по мере изменения

реальности и принимать меры, когда какое-то обязательство оказывается невыполнимым.

Управление рисками

Все проекты имеют неизвестные параметры и риски. Жизненно важными факторами успеха являются предвидение потенциальных рисков, оценка их вероятного влияния на проект и управление ими в максимально возможной степени. Неспособность активно управлять рисками нередко приводит к проблемам, которые могут удивить участников и, возможно, сорвать проект. Его руководитель должен сосредоточиться на достижении успеха и внимательно следить за назревающими проблемами. Управление рисками заключается в поиске возможных вариантов решения до того, как встанет вопрос: «И что теперь делать?»

Управление коммуникациями

Управление любым проектом основывается на коммуникации. В рамках проекта генерируется информация о статусе, проблемах, расходовании и потребностях ресурсов, изменениях, ожиданиях и т. д. Управлять проектом — значит получать всю эту информацию, хранить ее и передавать нужным людям в нужное время. Крупные проекты, в которых участвуют несколько команд, располагающихся в разных местах, говорящих на различных языках, имеющих разные культуры и предпочтения в общении, подразумевают особые коммуникативные проблемы.

Управление изменениями

Можно с уверенностью предположить, что конечный продукт проекта будет не совсем таким, каким он виделся в самом начале. Руководитель проекта должен своевременно реагировать на изменения в требованиях, приоритетах, ресурсах, технологиях, процессах и правилах. Каждая команда с самого начала должна создать механизмы, которые помогут предвидеть изменения и адаптироваться к ним с минимальными задержками. Agile-проекты специально структурированы так, чтобы быстро адаптироваться к изменениям и появлению новых требований. В них оперативно корректируются списки требований, чтобы можно было гарантировать первоочередное выполнение задач с высоким приоритетом.

Управление ресурсами

Мне не нравится, когда руководители называют людей «ресурсами», но, вообще говоря, в большинстве программных проектов люди действительно являются самым важным и дорогостоящим ресурсом. Помимо подбора персонала, руководители должны предоставить помещения, компьютерную технику, коммуникационную инфраструктуру, испытательные лаборатории и привилегии доступа для подрядчиков. Управление бюджетом проекта — еще один важный вид деятельности.

Управление зависимостями

Многие проекты зависят от внешних и неподконтрольных факторов. Они могут ожидать доставки программных или аппаратных компонентов от третьей стороны. Проект по созданию нового принтера может быть отложен из-за того, что международный стандарт нового протокола аппаратного интерфейса еще не завершен. Задачи и действия также могут иметь зависимости внутри проекта. Его руководитель должен определить такие зависимости, включить соответствующие сроки выполнения в график и отслеживать их статус, чтобы знать, когда будет выполнена каждая зависимость. Кроме того, рекомендуется создать планы на случай непредвиденных обстоятельств и задержек в реализации зависимостей.



Зависимости могут быть направлены и в противоположную сторону. Если от текущего проекта зависит другой проект, то руководитель должен своевременно сообщать о статусе своего проекта другой команде, чтобы они знали, чего ожидать.

Управление контрактами

Контракты — это юридические соглашения между сторонами. Не все проекты предполагают формальные контракты, но кто-то должен внимательно следить за контрактами в проектах, где они имеются. Невыполнение условий контракта может иметь серьезные последствия.

У развивающейся организации могут быть контракты с заказчиками, поставщиками товаров и субподрядчиками, выполняющими отдельные виды работ по проекту. В контрактах должны быть указаны такие детали: кто будет платить за изменение объема работ по тре-

бованию клиента и какие последствия влечет невыполнение контрактных обязательств.

Даже в проектах, не связанных официальными контрактами, достигнутые между участниками соглашения подразумевают определенный уровень «контракта». Управлять этими неявными контрактами может быть сложнее и важнее просто потому, что они не обсуждаются и не документируются явно. Для руководителя проекта было бы неразумно игнорировать неявные соглашения, ожидания или обязательства только из-за отсутствия письменного контракта.

Управление поставщиками

Нередко часть работ в программных проектах выполняется сторонними поставщиками (с которыми, разумеется, заключаются контракты). Иногда сторонней компании может быть передан весь объем работ по разработке. Некоторые проекты отдают на аутсорсинг только определенные виды деятельности, например тестирование системы. Построение партнерских отношений с этими поставщиками подразумевает в себя заключение договорных соглашений, создание механизмов коммуникации и общих инструментов, определение требований к качеству и процессов разрешения споров. Вследствие договоренностей со сторонними поставщиками возникают риски и зависимости, на которые руководитель проекта может иметь не значительное влияние.

Устранение препятствий

Судя по этому длинному списку предметных областей, управление проектами, безусловно, является обширным и сложным процессом. Ответственные за успех проекта люди должны оценить, какие из этих областей применимы к нему, и убедиться, что соответствующий опыт, навыки и свободное время есть либо у них самих, либо у других членов команды.

Основная обязанность руководителя проекта — устранять препятствия, мешающие прогрессу команды. Я предпочитаю думать, что это руководитель проекта работает на команду, а не наоборот. Кто-то должен предоставлять ресурсы, разрешать конфликты, договариваться о результатах, координировать действия, вмешиваться и поддерживать бесперебойную работу команды. Любой проект с таким бы то ни было

названием подразумевает большую ответственность. В этой главе представлены 12 уроков, которые помогут облегчить работу руководителя проектов.



ПЕРВЫЕ ШАГИ: УПРАВЛЕНИЕ ПРОЕКТАМИ

Прежде чем вы перейдете к изучению уроков, связанных с управлением проектами, предлагаю вам потратить несколько минут на следующие действия. По мере чтения подумайте, в какой степени каждый из этих пунктов применим к вашей организации или команде.

1. Перечислите приемы управления проектами, в которых особенно преуспела ваша организация. Задокументирована ли информация об этих методах? Доступна ли она другим членам команды, чтобы они могли ознакомиться с этими методами и применять на практике?
2. Определите любые проблемы (болевые точки), которые можно отнести к недостаткам, в том, как ваши команды оценивают, планируют, координируют работу и следят за ее выполнением.
3. Опишите, как каждая проблема влияет на вашу способность успешно завершать проекты. Как проблемы мешают достижению успеха в бизнесе и разработчикам, и их клиентам? Вот некоторые общие проблемы:
 - неадекватное представление о необходимой работе и ее статусе;
 - пробелы в коммуникации;
 - недостатки в организации сотрудничества;
 - нереалистичные оценки и планы;
 - невыполненные обязательства;
 - неожиданные риски, которые материализуются в проблемы;
 - неудачные зависимости.
4. Для каждой проблемы, выявленной на шаге 2, определите основные причины, провоцирующие или усугубляющие проблему. Одни первоначальные причины могут скрываться внутри команды или организации; другие происходят извне и неподконтрольны вам. Проблемы, влияния и первоначальные причины могут сливаться, поэтому постарайтесь разделить их и увидеть, как они связаны. Вы можете найти несколько основных причин, способствующих появлению одной и той же проблемы, или несколько проблем, обусловленных одной общей причиной.
5. Читая эту главу, перечислите любые практики, которые могут быть полезны вашей команде.

Урок 23**При планировании работ нужно учитывать разногласия**

Однажды на работе я услышал такой разговор.



Менеджер Шеннон: «Джейми, я знаю, что вы сейчас тестируете удобство использования проекта Canary. У нас есть еще несколько проектов, ожидающих своей очереди на тестирование. Сколько времени вы тратите на это?»

Член команды Джейми: «Около восьми часов в неделю».

Менеджер Шеннон: «Хорошо, тогда вы могли бы за неделю протестировать пять проектов».

Видите ли вы какие-либо ошибки в рассуждениях менеджера? Пять умножить на восемь равно сорок, номинальное количество часов рабочей недели, поэтому на первый взгляд эти рассуждения кажутся разумными. Но здесь не учтены многие факторы, сокращающие время, которое люди могут потратить на работу над проектом в течение каждого дня: разногласия в рамках проекта (я не имею в виду межличностные разногласия и не обсуждаю их).

Существует разница между количеством рабочих часов и временем, фактически потраченным на выполнение работы. Эта разница — лишь один из множества факторов, которые должны учитывать и руководители проектов, и отдельные члены команды при переводе размеров или усилий в календарное время. Не учитывая эти факторы в своих планах, люди всегда будут недооценивать время, необходимое для выполнения работы.

Переключение между задачами и состояние потока



Люди не могут решать несколько задач одновременно, им приходится переключаться между ними. Когда многозадачные компьютеры переключаются с одной задачи на другую, время, необходимое для этого, расходуется непродуктивно. То же происходит и с людьми, только непродуктивный период оказывается намного длиннее. Человеку нужно время, чтобы собрать материалы, необходимые для решения другой задачи, получить доступ к нужным файлам и загрузить в мозг соответствующую информацию. Ему нужно изменить направление мысли, чтобы сосредоточиться на новой задаче и вспомнить, на чем он оста-

новился, когда работал над ней в прошлый раз. Все это занимает довольно много времени.

Кто-то лучше справляется с переключением между задачами, кто-то — хуже. Я, например, относительно недолго могу удерживать высокую концентрацию внимания, зато довольно быстро умею переключаться на что-то другое, а затем возвращаться к первоначальной задаче и продолжать с того места, на котором остановился. Однако для многих людей слишком частое переключение с одной задачи на другую влечет за собой снижение продуктивности. Как объясняет Джоэл Спольски (Joel Spolsky), программисты особенно восприимчивы к переключению между задачами, отнимающему много времени (Spolsky, 2001):

У программистов переключение между задачами занимает очень, очень много времени, поскольку программирование — такая задача, которая требует удерживать в голове множество самых разных сведений. Чем больше вы помните, тем продуктивнее программируете. Программист, пишущий код с максимальной интенсивностью, удерживает в голове миллионы вещей одновременно.

Люди не могут решать несколько задач одновременно, им приходится переключаться между ними.



Когда я был руководителем проекта, разработчик по имени Джордан пожаловался мне, что не понимает приоритеты пунктов в своем списке заданий. Он какое-то время работал над задачей А, затем почувствовал, что слишком долго пренебрегал задачей Б, поэтому переключался на нее. В результате ему удалось сделать совсем немного. Тогда мы вместе с Джорданом пересмотрели его приоритеты и разработали план распределения времени между задачами. Он перестал тратить время впустую, его продуктивность повысилась, и Джордан почувствовал себя намного увереннее. Издергки на переключение между задачами и путаница с приоритетами повлияли на продуктивность Джордана и его душевное состояние.



Когда вы глубоко погружены в какую-то работу, сосредоточены и не отвлекаетесь, то входите в психическое состояние, называемое *потоком*. Это состояние повышает продуктивность тех, кто занимается творческой интеллектуальной работой, такой как разработка программного обеспечения или написание книг (DeMarco, Lister, 2013). Вы понимает-

те, над чем работаете, нужная информация находится в вашей оперативной памяти, и вы знаете, чего хотите достичь. Определить состояние потока можно по таким признакам, как потеря счета времени, достижение хороших результатов и получение удовольствия. В какой-то момент ваш телефон уведомляет о получении сообщения, всплывает уведомление от клиента по электронной почте, ваш компьютер напоминает, что через пять минут начнется важная встреча, или кто-то зашел поговорить. Бум — и ваш поток прерван.

Внешние события могут прервать поток, и порой требуется несколько минут, чтобы вернуть мозг в то высокопродуктивное состояние, в котором он пребывал до события, прервавшего мыслительный процесс. В некоторых отчетах отмечается, что подобные прерывания и необходимость переключения между задачами могут привести к непродуктивному расходованию не менее 15 % времени работника умственного труда, то есть более одного часа в день (DeMarco, 2001). Реалистичная оценка эффективной работоспособности основывается не на количестве часов, которые вы находитесь на работе или тратите на выполнение задачи, а на количестве часов, в течение которых вы работаете *непрерывно* (DeMarco, Lister, 2013).

Чтобы достичь высокой продуктивности и получить удовлетворение от продолжительного состояния потока, необходимо активно управлять своим рабочим временем. Возможности отвлечения внимания есть всегда, если не заблокировать их. Джори Маккай (Jogy MacKay) предлагает несколько рекомендаций по уменьшению количества переключений контекста и сопутствующего снижения производительности (MacKay, 2021).

- **Составьте расписание и определите четкие границы периодов концентрации.** Планирование рабочего дня с периодами, выделенными для выполнения конкретной работы, создает возможности для продолжительной глубокой концентрации. Если характер вашей работы позволяет, то посвящайте отдельные полные дни недели сосредоточению на самых важных индивидуальных задачах, более активному общению с другими или наверстыванию упущенного.
- **Выработайте привычку решать только одну задачу в течение дня.** Один талантливый, но непродуктивный член моей команды смог делать больше, когда мы договорились, что он выделит половину дня, на протяжении которой вообще не будет отвечать на звонки, текстовые сообщения или электронные письма.

- **Используйте процедуры, избавляющие от остаточного внимания при переходе от одной задачи к другой.** Физический переход к следующей задаче не сразу отключает мозг от решения предыдущей, что может отвлекать внимание. Небольшой переходный ритуал или отвлечение (чашка кофе, забавное видео) может помочь переключить мышление на новую задачу.
- **Делайте регулярные перерывы для восстановления сил.** Концентрация в состоянии потока велика, но все имеет пределы. Вы должны время от времени выходить из этого состояния, чтобы «глотнуть воздуха». Помассируйте уставшую шею, руки и плечи. Чтобы уменьшить нагрузку на глаза, периодически отводите взгляд от экрана монитора и на несколько секунд фокусируйтесь на чем-то вдалеке. Короткие перерывы умственной деятельности помогают отдохнуть и с новыми силами погрузиться в продуктивное состояние потока.

Эффективные часы

На работе время утекает в самых разных направлениях. Вы посещаете встречи и видеочаты, отвечаете на электронные письма, ищете информацию в Интернете, участвуете в обзорах и просматриваете код своих товарищей по команде. Время уходит на неожиданное исправление ошибок, обсуждение идей с коллегами, административную деятельность и обычное здоровое общение.

Удаленная работа из дома сопряжена с риском отвлечения на другие занятия, более увлекательные, чем работа над проектом. Даже работая 40 часов в неделю, вы едва ли будете тратить все их до минуты на свой проект.



Одна моя группа разработчиков программного обеспечения измерила, сколько времени посвящалось проектам в течение нескольких лет (Wiegers, 1996). Люди подсчитывали время (с точностью до получаса), которое они тратили на работу над каждым проектом, по десяти категориям деятельности: планирование проекта, выявление требований, разработка дизайна, реализация, тестирование, документирование и четыре типа поддержки. Мы не стремились к тому, чтобы еженедельные цифры складывались в общую сумму, а просто хотели понять, как на самом деле расходовалось время и насколько наши представления и желания расходятся с реальностью.

Результаты оказались поразительными. В первый год, за который собирались данные, мы посвящали работе над проектом в среднем 26 часов в неделю. Это заставило нас всех более осознанно искать способы продуктивного расходования времени. И все же нам так и не удалось превысить 31-часовой рубеж.

Некоторые из моих коллег получили аналогичные результаты, подсчитав, что тратят на работу над проектом в среднем по пять-шесть часов в день. Другие источники также предполагают, что типичное среднее идеальное рабочее время — «время непрерывного внимания к задачам проекта» — составляет около пяти часов в день (Larman, 2004). Вместо того чтобы полагаться на опубликованные данные для оценки эффективно расходуемого времени, соберите собственные. Благодаря информации о работе в течение нескольких типичных недель вы получите представление о том, сколько часов в неделю сможете посвятить задачам проекта, что влияет на прогнозируемую производительность или скорость команды.



Цель этих подсчетов не в том, чтобы руководители могли увидеть, кто усердно работает. Руководители вообще не должны видеть данные, характеризующие отдельных лиц, только обобщенные сведения, характеризующие команду или организацию. Знание среднего эффективного рабочего времени команды в неделю помогает более трезво оценивать реальность, составлять планы и брать обязательства.

Другие источники разногласия в проекте

Помимо ежедневной траты времени на бесчисленное количество дел, проектные команды теряют время из-за других источников разногласий. Например, большинство корпоративных IT-организаций отвечают как за разработку новых производственных систем, так и за усовершенствование и обслуживание существующих. Поскольку никто не может предсказать, когда что-то сломается или потребуется внести какое-то изменение, из-за таких спорадических перерывов на техническое обслуживание члены команды тратят время на выполнение незапланированной работы.

Большое расстояние между участниками проекта может затормозить обмен информацией и принятие решений. (См. урок 39 «Даже небольшие физические расстояния препятствуют общению и совместной работе».) Даже при широком использовании инструментов организации совместной работы проекты с участием людей, находящихся



в разных местах и часовых поясах, могут замедляться из-за коммуникативных трений. Иногда не получается связаться с человеком (например, с ключевым представителем клиентов), который мог бы дать нужный вам ответ. Вы вынуждены либо ждать, когда он станет доступен, либо сделать все возможное, чтобы продолжить. Это замедляет работу, особенно когда неправильные предположения приводят к переделке.

Дополнительные разногласия могут возникнуть ввиду состава команды, если участники проекта говорят на разных языках и работают в разных культурах. Из-за неясных и изменчивых приоритетов требований к исследованиям, на обсуждение и корректировку могут уходить часы. Команде может понадобиться временно отложить некоторые незавершенные дела, если в расписании появится новая задача с более высоким приоритетом. Незапланированные доработки также требуют времени.



Приведу в качестве примера реальный проект, в котором участвовали заказчик из восточной части США и поставщик с запада Канады (Wiegers, 2003). В их план входило несколько экспертных оценок промежуточных результатов. Однако удаленные обзоры заняли больше времени, чем ожидалось, как и последующие действия, связанные с проверкой внесенных исправлений. Медленное принятие решений на расстоянии еще больше снизило темпы проекта. Несспешное решение вопросов о требованиях и неясность в выборе правильного контактного лица для каждого вопроса стали дополнительными препятствиями. Эти и другие факторы привели к отставанию проекта от графика уже после первой недели и в итоге стали причиной его провала.



Последствия планирования

Разногласия в проекте сильно влияют на оценку, и об этом следует помнить. Поэтому я преобразую идеальную оценку усилий в календарное время, опираясь на коэффициент эффективности моего рабочего времени. Я также обдумываю, могут ли какие-нибудь другие источники разногласий повлиять на мои оценки. Затем я пытаюсь организовать свою работу так, чтобы сосредоточиться на одной задаче за раз, пока она не будет завершена или не возникнет непреодолимое препятствие.



Мой коллега Дейв так описал, что происходит с его текущим проектом, руководитель которого не учитывает влияния времени, потерянного из-за чрезмерной многозадачности:

Руководитель любит перераспределять рабочее время людей между командами: 50 % сюда и 50 % туда, или 50, 25 и 25. Но когда наступает время подвести итоги, он, похоже, забывает о процентах и думает, что все в каждой команде заняты полный рабочий день, а потом удивляется тому, как много времени тратится впустую. Кроме того, работа в нескольких командах означает необходимость тратить больше времени на собрания и меньше — на программирование.

Если люди дают оценки, не принимая во внимание множество причин замедления, вызванного разделением времени и условиями проекта, то им суждено каждый раз не вписываться в свои временные рамки.

Урок 24 Не давайте оценок наугад

Вы — бизнес-аналитик или владелец продукта. Идя по коридору на свое рабочее место, вы встречаете Мелоди, одну из представительниц клиентов вашего проекта. «Я хотела бы добавить кое-что в проект, над которым мы работаем», — говорит Мелоди. Вы останавливаетесь и слышите ее описание новой функции. «Как вы думаете, сколько времени потребуется, чтобы сделать это?» — спрашивает она.

Вы задумаетесь на мгновение и отвечаете: «Около трех дней».

«Отлично, спасибо! Давайте сделаем это», — говорит Мелоди, и вы оба продолжаете свой путь по коридору.

Вы возвращаетесь к своему столу и начинаете обдумывать просьбу Мелоди. И чем дольше вы размышляете над новой функцией, тем больше сложностей обнаруживаете. Вы понимаете, что команда не сможет реализовать ее за три дня, как вы пообещали Мелоди. Функция оказалась гораздо сложнее, чем вы думали вначале. Более того, по мере углубления в детали вы начинаете беспокоиться по поводу того, что она может конфликтовать с другой функцией, которую команда запланировала реализовать в следующей итерации. Не слишком ли поздно изменить ответ, который вы дали Мелоди?

Поспешные прогнозы

Лучший ответ на вопрос, требующий оценки: «Я обдумаю это и вернусь к вам с ответом». Оценка, котораядается мимоходом на основе огра-



ниченной информации и поверхностного анализа, может оказаться ужасающе неточной, но она очень похожа на обязательство перед другим человеком. Мелоди поймала вас на слове, так что теперь вы должны объяснить, что на удовлетворение ее просьбы уйдет больше времени, чем вы предполагали. Потребуются переговоры и пересмотр планов, прежде чем команда сможет принять решение о добавлении этой новой функции. Может состояться неловкий разговор. Майк Кон отмечает (Cohn, 2010):

Слова «По нашим оценкам, это займет семь месяцев» были восприняты как «Мы обязуемся закончить через семь месяцев». И оценка, и обязательство важны, но не следует путать их.

Часто есть соблазн дать быструю оценку навскидку, но постарайтесь подавить это искушение, пока не изучите вопрос более подробно. Такие быстрые ответы не основаны ни на каком анализе; это скорее догадки, взятые из воздуха. Прежде чем дать оценку, убедитесь, что точно знаете, о чём вопрос. Затем оцените, что реально потребуется для его решения.

Быстрые — и часто не очень точные — оценки являются распространённой проблемой при работе с новыми требованиями и запросами на изменение. После их анализа часто оказывается, что проблема более обширная, чем предполагалось в первый момент. Если вы дадите реалистичную оценку, то заказчик может отказаться от своего запроса, поскольку его реализация не стоит времени или финансовых затрат. Лучше узнать об этом до того, как вы начнете реализовывать новую функциональность, чем отказаться от нее, когда она уже будет наполовину проработана и вдруг выяснится, что изменение оказалось неоправданно большим. Я видел, как это происходит; это довольно дорого.

Изучая проблему и формируя свою оценку затрат, учитывайте следующие вопросы и добавляйте ответы на них в свое сообщение с результатами оценки.

- Какие предположения повлияли на вашу оценку? Как можно проверить их достоверность?
- Известно ли, кто будет выполнять эту работу? У разных членов команды разные навыки; не все одинаково продуктивны. Если вы не знаете, кто справится с этой работой, то должны принять в расчет некий средний уровень производительности.

- Кто-то должен будет написать и выполнить тесты для новой функциональности. Вам может понадобиться выполнить реview кода и провести регрессионное тестирование, чтобы убедиться, что изменение ничего не нарушило. Уточните, касается ли ваша оценка всего объема работ или только собственно программирования.
- Учитывали ли вы неочевидные последствия и дополнительную работу, которая может потребоваться помимо реализации новой функциональности? Она может повлиять на другие функции или негативно отразиться на некоторых атрибутах качества. Для предотвращения негативных последствий может понадобиться изменить дизайн или интерфейс либо обновить пользовательскую документацию.
- Можете ли вы назвать какие-либо риски, способные омрачить ваш «солнечный день», когда «все идет хорошо»?

Чем более расплывчато сформулирована задача и чем менее четко определены допущения, тем более расплывчатой будет оценка.

Страх неопределенности

Иногда люди, получающие оценки, не осознают, насколько неопределенной может быть даже тщательно продуманная оценка. Чем более нечетко сформулирована задача и чем менее четко определены допущения, тем более расплывчатой будет оценка. Если клиенты услышат одно число, например «три дня», то запомнят и будут строить свои планы, исходя из этого срока. Оценка в виде диапазона — от лучшего до худшего случая — послужит напоминанием, что она является приблизительным прогнозом, а не гарантией (McConnell, 2006). Однако и в этом случае клиент может сосредоточиться на нижней границе диапазона: «Значит, вполне возможно, что вы закончите за три дня, верно?» — поэтому, озвучивая оценку, важно четко изложить ожидания.



Проблема необдуманных оценок, полученных в результате прикидок на скорую руку, заключается в том, что люди, услышавшие оценку, плохо понимают, как вы ее получили. Вы можете тщательно все обдумать и проанализировать, учесть непредвиденные обстоятельства, но в ответ услышать: «Это смешно. Такая работа не может занять так много времени». Поверхностное суждение, которому не хватает тщательного анализа, будет более оптимистичным и, следовательно, вос-

примется более благосклонно, но приведет к еще большему разочарованию, когда проявится реальность.

Нам всем нравится доставлять радость, когда кто-то о чем-то нас просит. Однако если вы потратите время на размышления о проблеме, прежде чем озвучить оценку ее решения, то не окажетесь в неудобном положении.

Урок 25

Айсберги всегда больше, чем кажутся



Однажды мне позвонил мой руководитель и спросил: «У тебя есть час свободного времени?» К нему обратился ученый, написавший на своем компьютере простую программу на языке BASIC для расчета формул химических растворов, которые он использовал в исследованиях. Ученый хотел, чтобы один из нас, программистов, перенес его программу на наш мейнфрейм, чтобы ею могли пользоваться другие люди.

На первый взгляд, в этой работе не было ничего сложного. Проверка программы, о которой просил руководитель, действительно заняла бы всего пару часов. Однако, просматривая программу, я понял, что проблема гораздо шире. Программа должна была содержать больше расчетов, чем предусмотрел ученый, поскольку разные исследователи работали с разными химическими веществами. Расчеты должны были быть более точными, чем предполагал реализованный в программе упрощенный подход. Вдобавок программе требовалась гибкий интерфейс, чтобы с ним могла работать большая группа пользователей, и возможность генерировать отчеты, подходящие для использования в лаборатории.

Я изучил весь спектр требований и исследовал задачу увеличения точности вычислений. Затем разработал решение и пользовательский интерфейс, написал код и протестировал приложение. Данный проект занял чуть более 100 часов моего времени, но это время было потрачено не напрасно, поскольку приложение активно использовалось в течение многих лет.

Конечно же, далеко не все программные проекты оказываются в 100 раз больше, чем предполагалось первоначально. Но практически все проекты выходят за рамки первоначальной концепции после более тщательного анализа проблем и запросов на изменение во время разработки.

Чем дольше длится проект, тем большего его роста можно ожидать. ИТ-аналитик Каперс Джонс обнаружил, что требования, предъявляемые к крупным проектам, обычно увеличиваются на 1–3 % в месяц во время разработки (Jones, 2006). Если в ваших планах не предусмотрен такой рост, то вы наверняка отстанете от графика.

Итеративные подходы к разработке подтверждают, что истинный размер айсберга неочевиден на первый взгляд.

Вы, наверное, слышали, что на поверхности воды находится меньшая часть айсberга. Точно так же большая часть работы над программным проектом изначально может быть неочевидна. Итеративные подходы к разработке подтверждают, что истинный размер айсберга неочевиден на первый взгляд. Поскольку программное обеспечение поставляется частями, заинтересованные стороны будут просить добавить дополнительные функции. То есть чем больше они увидят, тем больше им захочется (Davis, 1995). Айсберг всегда больше, чем виделось изначально, и продолжает расти.



Резерв времени

Один из способов не отстать от графика разработки по мере разрастания проекта — предусмотреть *резерв времени* на случай появления непредвиденных обстоятельств. Запас времени помогает справиться с неопределенностью. Планируя цикл разработки, вы основываете свои оценки на ограниченной информации о масштабах проекта, предположениях, которые могут оказаться ошибочными, и других переменных. Из-за этих неизвестных рекомендуется добавить в график некий резерв времени на случай непредвиденных обстоятельств. В противном случае первое поступившее новое требование, выявление первой заниженной оценки или возникновение первой непредвиденной задачи сорвут график.



Резерв времени — это не произвольный фиктивный фактор, вставленный в график, и не бессмысленное завышение всех отдельных оценок. Оправданный резерв времени можно рассчитать на основе предыдущего опыта выхода за плановые границы и наблюдавшегося роста



требований. Количественный анализ рисков — еще один метод, позволяющий продумать потенциальные риски, с которыми может столкнуться проект, вероятность их появления и как они могут затормозить работы. Управление проектами по методу критического пути — это метод добавления хорошо продуманных резервов времени (Goldratt, 1997). Они могут идти в конце последовательности взаимозависимых этапов (*питающие буферы*), а также в конце всего проекта (*буфер проекта*). На рис. 4.1 показано, как расположить буфера обоих видов в представлении расписания на диаграмме Ганта.

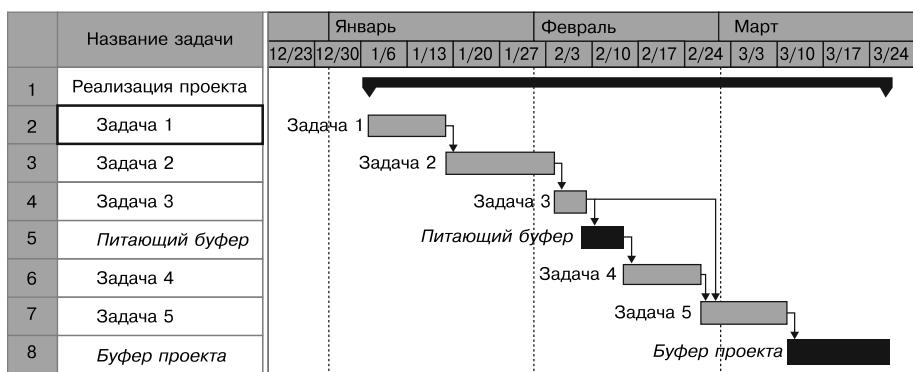


Рис. 4.1. На диаграмме Ганта для проекта за каждой группой задач следует питающий буфер, а за последней задачей — буфер проекта

Используя Agile-разработку, предусматривайте небольшие резервы времени в каждой итерации. Это поможет сгладить влияние неопределенностей в проекте. Такие резервы помогут удержать цикл итераций в нужном русле и не откладывать незавершенную работу на более поздние итерации. Вдобавок можно предусмотреть дополнительную резервную итерацию в конце проекта, предназначенную для реализации отложенных и дополнительных пользовательских требований и другой затянувшейся работы, как показано на рис. 4.2 (Thomas, 2008b).

Рассмотрим один из способов включения буферов в Agile-проект (Cohn, 2006). Пользовательские истории (или, в более общем смысле, дополнительные требования) бывают разных размеров и оцениваются командами в *относительных единицах* (Cohn, 2004). В Agile-проектах скорость команды измеряется в каждой итерации, имеющей продолжительность в несколько недель. Скорость определяется как количество относительных единиц работы, которую команда может выполнить за одну итерацию.

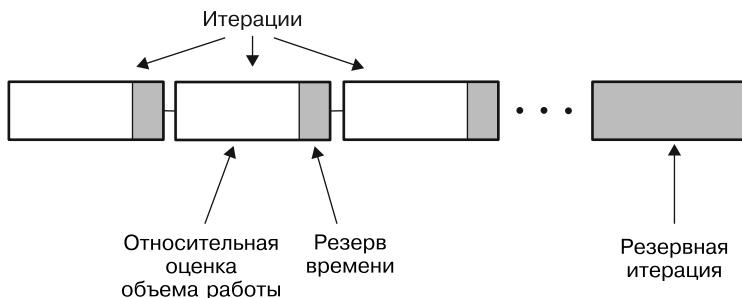


Рис. 4.2. При планировании проекта буферы резервного времени можно добавить и в конец каждой итерации, и в конец проекта в виде дополнительной итерации

Предположим, что список требований для вашего продукта содержит около 150 пунктов. Ваши измерения скорости показывают, что команда способна реализовать в среднем 30 пунктов за итерацию, что предполагает продолжительность проекта в пять итераций. Вы можете основывать свой план доставки на чуть более консервативной оценке скорости в 25 пунктов за итерацию, то есть на реализацию всего проекта предусмотреть шесть итераций ($150 \div 25$). Предложенный вами план доставки теперь предусматривает реализацию дополнительных пяти непредвиденных требований в каждой итерации.

Команда по-прежнему планирует итерации так, будто в каждой будет реализовано по 30 пунктов. То есть они работают над достижением внутренней цели (30), более амбициозной, чем внешнее обязательство (25). Разница дает тот самый резерв на выполнение непредвиденных работ, исправление ошибок, рефакторинг и другие действия, требующие времени. В измерениях скорости команды, вероятно, уже учтено влияние текущих исправлений ошибок и рефакторинга, но неприятные сюрпризы возможны в любое время в любом проекте (Cohn, 2014).



Рискованные оценки

Включение буферов резервного времени на случай непредвиденных обстоятельств растягивает ожидаемый график доставки, поэтому руководители и клиенты могут противиться принятию такого графика. «Это просто буфер, — возражают они. — Если убрать его, то вы сможете закончить раньше, верно?» Скорее всего, нет.



Резерв времени не влияет на конечный результат, он лишь обеспечивает запас прочности, позволяя учитывать неизвестные, непредвиденные обстоятельства и неточности оценки. Удаление резерва из графика не удаляет эти переменные, а просто снижает вашу способность справляться с ними и при этом выполнять обязательства. Руководитель, предлагающий убрать резерв времени на случай непредвиденных обстоятельств, делает несколько предположений.

- Информация, которой вы располагаете сегодня, понятна, точна и стабильна.
- Все оценки точны, или, по крайней мере, любые неточности будут уравновешивать друг друга.
- Вы знаете, кто будет работать над проектом, и состав команды не изменится на протяжении всего проекта.
- Члены команды не будут отвлекаться на работу по поддержке предыдущего продукта или другие посторонние работы.
- Никто не заболеет, не уйдет в отпуск и не покинет компанию.
- Никакие риски не превратятся в проблемы, и не появятся никакие новые риски.
- Все зависимости проекта и риски от внешних факторов будут устраниены вовремя.

Эти предположения загоняют команду разработчиков в угол и почти гарантированно приведут к перерасходу средств. Если вы столкнулись с нежеланием руководства предусмотреть в графике резерв времени на случай непредвиденных обстоятельств, то напомните о некоторых неожиданных событиях, возникших в предыдущих проектах. Спросите, есть ли основания полагать, что новый проект будет другим, что ни одна из неприятностей больше не повторится. Если нет, то резерв времени должен остаться.

Лучший способ обосновать добавление в график резерва времени — показать, как он рассчитывается на основе предыдущего опыта работы. Данные, показывающие, насколько обычно растет объем требований в ваших проектах, помогут вам обосновать необходимость включения в проект резерва времени.



Однажды я разговаривал со старшим менеджером одного из моих клиентов об огромном проекте, реализация которого, как они рассчитывали, должна была занять пять лет. Я рассказал ему о статистике по отрасли, согласно которой темпы роста требований для проектов по-

добных размеров составляют примерно 2 % в месяц. Такой темп роста может увеличить объем конечного продукта 60-месячного проекта более чем вдвое. «Выглядит вполне правдоподобно», — ответил старший менеджер.

Затем я спросил, предусматривает ли его план проекта какие-либо резервы на случай непредвиденных обстоятельств. И если да, то в каком объеме. Его ответ был предсказуем: «Нет». Я подозреваю, что на реализацию этого проекта ушло намного больше времени, чем пять лет.

Контракты на айсбергах

Феномен айсberга также влияет на контрактные проекты. Если вы являетесь подрядчиком и хотите получить работу, то у вас может возникнуть соблазн исключить резервное время на непредвиденные обстоятельства из вашей заявки, чтобы сделать цену более конкурентоспособной. Но что произойдет, когда выявится истинный размер айсберга? Или если заказчик потребует расширить объем работ, не согласившись при этом отложить выполнение каких-либо других обязательств?

Если бы в вашем плане был предусмотрен резерв времени на преодоление непредвиденных обстоятельств, то вы смогли бы справиться с небольшим расширением объема работ, оставаясь в рамках бюджета или графика. Когда клиент спрашивает: «Сможете ли вы добавить одно небольшое требование или внести одно дополнительное изменение? Вы же все равно закончите вовремя, верно?» — наличие резерва времени позволит вам ответить утвердительно. Клиент воспримет ваш ответ «да» как признак вашей гибкости и желание удовлетворить его потребности. Но при отсутствии запланированного резерва рост проекта может привести к перерасходу средств и, возможно, нарушению обязательств по времени.

Я слышал о практике предложения нереально низкой цены ради получения контракта, когда тендер выигрывает тот, кто озвучит самую сладкую ложь. Вам придется решить, хотите вы вести бизнес, основываясь на реалистичных оценках будущего или фантазиях, которые могут обернуться против вас. Подходы к поэтапной разработке помогают справиться с неопределенностью размера, позволяя вам заранее признавать, что вы еще не знаете многое из того, что может повлиять на результат проекта. Однако некоторые клиенты будут против такого проекта с открытой датой завершения.

Преимущества резервов времени

Планирование возможных событий не изменит того, что произойдет на самом деле, но поможет вам преодолеть все сложности, не нарушая своих планов. Если вы не знаете всего размера айсберга, то ожидайте, что он будет больше, чем кажется, и планируйте соответственно. Добавление резервного времени на непредвиденные обстоятельства на каждом этапе помогает выполнять договоренности в соответствии с графиком.

Предположим, что реализация проекта идет успешно и вы не полностью использовали выделенные резервы времени. В таком случае можно передать продукт досрочно — от этого выигрывают все.

Урок 26

Ваши переговорные позиции будут сильнее при наличии обосновывающих данных

Встречаясь с продавцом в автосалоне, чтобы договориться о цене, вы оказываетесь в невыгодном положении. Продавцы имеют под рукой множество данных, в том числе:

- рекомендованная изготовителем розничная цена интересующего вас автомобиля;
- сумма, которую дилер заплатил за автомобиль (цена в счете-фактуре);
- минимальная прибыль, которая не позволит дилеру остаться в убытке;
- цена дилера и наценка на любые аксессуары или дополнительное оборудование;
- любые специальные акции или поощрения, доступные для дилера или покупателя;
- стоимость вашего прежнего автомобиля, сдаваемого по программе обмена «трейд-ин»;
- сколько заплатили предыдущие покупатели, купившие такой же автомобиль.

У вас же, скорее всего, из информации будет иметься только цена. Если вы не поленились и провели предварительные исследования, то можете знать внутреннюю информацию о ценах из такого источника, как Consumer Reports¹. Мне такая подготовка очень помогла. В про-

¹ Периодическое издание Союза потребителей в США. — Примеч. пер.

тивном случае продавец на переговорах занимает гораздо более сильную позицию, чем вы, благодаря наличию у него дополнительной информации.

Откуда вы взяли эту цифру?

Программный проект не имеет фиксированной цены, которую можно использовать в качестве отправной точки. Тем не менее аналогичный дисбаланс в обладании информацией может повлиять на переговоры между ключевыми заинтересованными сторонами.

Предположим, я — руководитель проекта. Спонсор проекта (какой-нибудь старший менеджер, специалист по маркетингу или мой основной заказчик) спрашивает меня, сколько времени потребуется для выполнения нового проекта и сколько это будет стоить. Основываясь на своем понимании проекта и собственном опыте, я разрабатываю и представляю смету. Если спонсору не нравится моя оценка, то возможны два исхода. Первый: я пытаюсь защитить свою оценку, не имея данных, в то время как спонсор настаивает на сокращении затрат. Это не столько переговоры, сколько дебаты, которые я, скорее всего, проиграю.

Второй исход: я, основываясь на данных, а не на эмоциях, описываю спонсору, как получена моя оценка, например так:



- Я посоветовался со знающими и опытными членами команды, которые понимают, какого объема работы требует новый проект.
- Мы оценили размер проекта на основе имеющейся информации и при этом учли фактор неопределенности и ожидаемого разрастания, основываясь на прошлом опыте. Мы использовали метод групповой оценки, такой как Wideband Delphi (Wiegers, 2007), или один из нескольких Agile-методов оценки (Sliger, 2012).
- Мы сделали определенные предположения, которые повлияли на нашу оценку.
- Мы пересмотрели наш прошлый опыт работы с аналогичными проектами и сопоставили данные о размерах проектов, наши оценки, фактические результаты, риски и неожиданности, с которыми столкнулись, и т. д.
- Мы оценили, в какой степени предыдущие проекты могут служить моделями для оценки нового проекта, и определили, как различные характеристики нового проекта влияют на наши оценки.

- Основываясь на наших предыдущих измерениях продуктивности, мы рассчитали наиболее вероятные, пессимистичные и оптимистичные оценки продолжительности реализации и затрат.

Если спонсор по-прежнему не соглашается с оценкой даже после объяснения, как я к ней пришел, мы можем обсудить факты. Возможно, проект не похож ни на что сделанное нами ранее и наши исходные данные неактуальны. Это увеличивает риски и неопределенность оценок. Может быть, мы недостаточно хорошо понимаем проект, чтобы прийти к более реалистичной оценке. Нам придется двигаться постепенно, оценивая каждый фрагмент по мере продвижения. Это снова порождает некую неопределенность в отношении сроков завершения проекта и его конечной стоимости. Наши базовые данные и процесс аналитической оценки служат основой для сбалансированного обсуждения реального положения дел; а вот нравится оно нам или нет — отдельный вопрос.



Данные — ваш союзник. Однажды менеджер отклонила представленную мной смету. Я объяснил, как вывел оценку на основе опыта наших предыдущих проектов и почему посчитал ее предположение о значительном повышении продуктивности нереалистичным. Я спросил, почему она решила, что этот проект будет развиваться быстрее и столкнется с меньшим количеством неожиданностей, чем наши предыдущие проекты. У нее не нашлось хорошего объяснения. Ее оценки были больше связаны с надеждами, чем с реальными ожиданиями. Надежда — не стратегия.

Никто не может точно предсказать будущее. Лучшее, что можно сделать, — экстраполировать предыдущий опыт, внести корректиды там, где это оправданно, и признать неопределенность.



Принципиальные переговоры

Каждый раз, когда возникает разрыв между ожиданиями или требованиями заинтересованной стороны и вашим прогнозом в виде оценки, необходимо провести переговоры. *Принципиальные переговоры* — это метод достижения взаимоприемлемых соглашений (Fisher et al., 2011). Принципиальные переговоры базируются на четырех принципах.

- **Отделите людей от задач.** Если обсуждение сводится к личной борьбе между вами и кем-то обладающим большей властью, то вы гарантированно проиграете, если не сможете привести убедительные доводы в свою пользу.
- **Сосредоточьтесь на интересах, а не на позициях.** Не упорствуйте бездумно в защите своей оценки. Постарайтесь сначала понять интересы другой стороны. Каковы их цели, потребности, проблемы и ограничения? Возможно, есть способ удовлетворить и их, и ваши интересы, изменив формулировку задачи или предлагаемое решение.
- **Придумывайте варианты для взаимной выгоды.** Если вас вынуждают дать обещание, которое, как вы знаете, команда не сможет выполнить, то ищите альтернативные взаимоприемлемые исходы с реалистичными целями, которых вы действительно сможете достичь. Посмотрите, есть ли способ встретиться посередине. По результатам большинства успешных переговоров ни одна из сторон не получает всего, чего хочет, но обе стороны могут смириться с результатом — компромиссом.
- **Настаивайте на использовании объективных критериев.** Здесь пригодятся ваши данные. Факты и анализ убеждают лучше, чем громко высказываемые мнения, возражения и истории. Используйте данные для обоснования своих оценок и уважайте данные, представленные вашим партнером по переговорам.

Никто не может точно предсказать будущее. Лучшее, что можно сделать, — экстраполировать предыдущий опыт, внести корректиды там, где это оправданно, и признать неопределенность. Пусть за вас говорят ваши данные, и предложите другой переговорной стороне поделиться своими данными. Возможно, так вы достигнете согласия и получите приемлемый результат.



Урок 27

Не записывая оценки и не сверяя их с тем, что произошло на самом деле, вы всегда будете строить догадки, а не оценивать

Любой человек, столкнувшись с новой частью работы, захочет узнать, сколько времени займет ее выполнение. Этим человеком может быть клиент, руководитель, товарищ по команде или вы сами. Если работа не является точной копией той, что вы делали раньше, то вы должны

оценить ее, отталкиваясь от предыдущего аналогичного опыта. Однако память часто подводит людей. Даже помня, сколько времени заняло какое-то предыдущее действие, вы едва ли вспомните, сколько времени *предполагалось* потратить на него первоначально. Чтобы получать достаточно точные оценки, нужны данные, а не смутные воспоминания, необходим набор фактов, которые можно анализировать, чтобы понять, предсказать и улучшить.



Однажды меня спросили: «Если мы должны основывать наши оценки на данных за прошедший период, то где их взять?» Самый простой ответ: записывайте все, что сделали сегодня, и завтра эти записи станут ретроспективными данными. Отдельные лица, команды и организации должны выработать привычку записывать свои оценки и вести учет фактических результатов. Это единственный способ улучшить оценку затрат на выполнение предстоящих действий.

Если прогноз и конечная реальность сильно расходятся, то попробуйте выяснить причину и подумайте, как получить более реалистичную оценку для аналогичной работы в будущем. Пришлось ли вам выполнять больше задач, чем вы ожидали? Их решение потребовало больше времени, чем вы предполагали? Были ли ваши предположения о продуктивности чрезмерно оптимистичными? Не помешали ли работе неожиданные факторы? По мере накопления данных вы сможете вычислять некие средние значения и получать более весомые оценки для будущей работы. Без данных вы просто строите предположения.

Несколько источников данных за прошедший период

Есть четыре потенциальных источника данных за прошедший период. Первый — ваш личный опыт. Индивидуальная производительность существенно различается в зависимости от навыков и опыта того, кто выполняет работу. Вот уже более 20 лет я работаю в собственной консалтинговой и обучающей компании, в которой я — единственный сотрудник. Зная то, сколько времени потребуется другому консультанту для разработки учебного курса, я не смогу предсказать, сколько времени потребуется мне, чтобы сделать то же самое. Я должен полагаться на свою личную историю, поэтому веду учет своих планов, оценок и фактической продолжительности выполнения задач, с которыми могу столкнуться снова.



Несколько лет назад я решил создать электронные версии шести моих учебных курсов. Раньше я никогда не делал ничего подобного и понятия не имел, сколько времени это может занять. Я создал список задач со

всеми шагами, которые только мог придумать. Поскольку это был новый опыт, я упустил из виду кое-какие задачи, которые добавил в свой список планирования для последующих проектов. Я записал, сколько времени мне потребовалось для выполнения каждого пункта моего первого курса.

Исходя из этих данных, я оценил, сколько времени займет перенос в электронный формат следующего курса. Скорректировал оценку с учетом полученных знаний, вновь обнаруженных задач и различий между двумя курсами. Завершив работу над вторым курсом, я сравнил свои оценки с фактически потребовавшимся временем, чтобы получить более точные оценки для оставшихся курсов. Наличие этих данных помогло мне довольно точно оценить затраты, когда несколько лет спустя мне предложили контракт с фиксированной ценой по разработке специализированных программ электронного обучения.

Считаю, что мои записи помогают обдумывать работу, которую мне, возможно, придется выполнить в рамках конкретного проекта, и оценивать время выполнения каждого действия. Они напоминают мне о задачах, которые иначе я мог бы пропустить. Например, люди часто забывают запланировать время для доработки недостатков, выявленных после таких мероприятий по контролю качества, как тестирование и рецензирование. Я всегда добавляю в свои оценки время для доработки, даже если точно не знаю, сколько это может занять в каждой конкретной ситуации.

Три других источника данных за прошедший период — ваш текущий проект, коллективный опыт вашей организации и средние показатели по отрасли для проектов, подобных вашему. Наиболее значимые данные поступают из недавно завершенных частей текущего проекта. Они отображают влияние окружения, культуры, процессов и инструментов на продуктивность текущей команды. Проекты, использующие методы Agile-разработки, хорошо подходят для сбора таких данных. Наиболее точные прогнозы будущей производительности основаны на средней скорости последних итераций при условии, что состав команды, характер и качество выполняемой работы остаются постоянными.

Статистика по ранее завершенным проектам в вашей организации и средние показатели по отрасли плохо подходят для предсказания вашего будущего, поскольку различия между проектами, командами и организациями увеличивают неопределенность. И все же лучше полагаться на такого рода исторические данные, чем давать оценки, основываясь исключительно на смутных воспоминаниях или догадках.



Характеристики программного обеспечения

Чтобы проанализировать опыт предыдущих проектов и составить более обоснованные планы, необходимо собрать некоторые характеристики. Люди могут измерять различные аспекты программных проектов на трех уровнях: индивидуальный участник, проектная группа и организация-разработчик. Знание таких характеристик, как объем, трудозатраты, время и уровень качества, может дать довольно полное представление о вашей работе (Wiegers, 2007).

- **Объем.** Выберите некую меру размера, значимую для той работы, которую вы выполняете. Это может быть количество требований, пунктов в списке функциональных особенностей, пользовательских историй, классов и количество строк исходного кода (хотя последний пункт выглядит довольно сомнительным). Для своих электронных курсов обучения я подсчитывал количество модулей и слайдов.
- **Трудозатраты.** Фиксируйте трудозатраты, необходимые для создания каждого проекта или отдельной его части. Понимая возможные трудозатраты и зная объем работы, вы сможете рассчитать производительность.
- **Время.** Запишите расчетную и фактическую продолжительность выполнения работ в единицах календарного времени. Как мы видели в уроке 23 «При планировании работ нужно учитывать разногласия», на преобразование рабочих усилий в календарное время влияет большое количество факторов.
- **Уровень качества.** Подсчет дефектов, обнаруженных различными способами, показывает, с помощью каких методов обеспечения качества лучше выявлять ошибки и где можно изыскать дополнительные резервы для улучшения качества. Учет времени, затраченного на доработку и исправление дефектов, тоже помогает в планировании. Если вы знаете, где возникла ошибка (конкретная итерация, требование или действие), то будете более отчетливо понимать, на чем следует сосредоточить дополнительные усилия, чтобы обеспечить качество в будущем.

В своей книге *Software Estimation*¹ (2006) Стив Макконнелл указывает на многочисленные проблемы, связанные с мерами измерения в каждой из этих категорий. Макконнелл подчеркивает важность четкого определения каждой собираемой метрики, что позволяет людям записывать

¹ Макконнелл С. Сколько стоит программный проект. — М.: Русская редакция; СПб.: Питер, 2007.

данные в единообразной форме. Если вы не сравниваете результаты с внешним эталоном, то согласованность внутренних измерений важнее абсолютной истины. Например, все члены команды должны одинаково оценивать свои усилия и подсчитывать дефекты. Предположим, одни члены команды добавляют время на отладку и доработку в общее время разработки, другие относят их к этапу тестирования, а третьи вообще не учитывают его. Такое смешивание теплого с мягким не способствует обоснованному предсказанию будущего.

Золотое правило метрик гласит, что данные нужно использовать для изучения и совершенствования, а не для наказания или вознаграждения кого-либо.

Необходимость записывать метрики, характеризующие программное обеспечение, нервирует некоторых людей. Определение характеристик программного обеспечения действительно является деликатной областью с множеством потенциальных ловушек (Wiegers, 2007). Золотое правило гласит, что данные нужно использовать для изучения и совершенствования, а не для наказания или вознаграждения кого-либо. Людей также часто беспокоит необходимость тратить время на сбор и анализ данных, но, как показывает мой опыт, запись базовой информации о проектной деятельности не требует особых затрат. Эта привычка быстро становится частью рабочего процесса.



Спустя долгое время после завершения работы у вас не получится точно восстановить ее характеристики. Поэтому создайте механизмы для сбора и сохранения данных по мере продвижения. Ретроспектива в конце цикла разработки дает хорошую возможность собрать данные для оценки прошлого и планирования будущего. Если вы потратите немного времени на запись происходившего сегодня, то завтра у вас будут исторические данные, которые помогут оценить следующую часть работы.

Урок 28

Не меняйте оценку в зависимости от того, что хочет услышать получатель

Предположим, ваш клиент спрашивает, сколько времени вам потребуется, чтобы завершить следующую часть вашего проекта. Основываясь на анализе прошлого опыта, вы отвечаете: «Около двух месяцев».

«Два месяца? — усмехается клиент. — Моя двенадцатилетняя дочь сможет сделать то же самое за три недели!»

В ответ вы, сопротивляясь искушению предложить ему нанять свою дочь, делаете вторую попытку: «Хорошо, как насчет одного месяца?»

Что же изменилось за эти несколько секунд? Задача не уменьшилась. Вы не стали более продуктивным. У вас не появился вдруг добровольный помощник, готовый помочь выполнить работу. Просто клиенту не понравился ваш первый ответ, и вы предложили альтернативу, кажущуюся более привлекательной. Если ваша оценка и ожидания заказчика слишком расходятся, то вам придется провести переговоры, чтобы прийти к какому-либо соглашению. Однако нет никаких оснований изменять продуманную оценку только потому, что она кому-то не понравилась (Wiegers, 2006b).

Оценка — это предсказание будущего. Мы основываем оценки на том, что известно о текущей задаче, на предыдущем опыте решения аналогичных задач и некоторых предположениях. Будучи особенно скрупулезными, мы можем также принять во внимание любые зависимости от факторов, неподконтрольных нам, возможные риски, потенциальные изменения в постановке задачи и неопределенность, которая может повлиять на наши планы. Чем меньше мы знаем обо всех этих факторах, тем ниже вероятность, что оценка будет соответствовать реальному положению дел. Кто-то другой, учитывающий иной набор параметров, может получить совсем иную оценку времени выполнения той же работы.

Цели и оценки



Важно отличать оценки от целей. Однажды я был свидетелем обсуждения нового проекта между разработчиком и старшим менеджером. Продолжительность проекта, по оценке разработчика, в четыре раза превышала ожидания менеджера. Но, уступив давлению менеджера, разработчик согласился на гораздо более короткий срок, даже притом что уложиться в него не было никакой возможности.

Более рациональным подходом для разработчика было бы сказать: «Я пришел к своей оценке согласно таким-то и таким-то расчетам. А как вы получили свою?» Однако в данном случае у менеджера не было оценки — у него была цель. У разработчика тоже не было оценки — у него было предположение. Цель и предположение оказались далеки

друг от друга. Если бы одна из двух сторон разработала верную оценку, то они могли бы быть ближе друг к другу. Вместо этого обсуждение превратилось в напряженный спор, и человек, обладавший большей властью, выиграл его, получив обещание от другого.

Важно отличать оценку от цели.

Когда корректировать оценку

Ваша оценка не должна зависеть от того, что хочет услышать заказчик. Тем не менее иногда имеет смысл изменить оценку или, говоря точнее, выполнить переоценку:

- если обнаружится, что предположение или часть информации были неверными;
- после начала работы, когда вы лучше поймете задачу;
- когда объем работы изменится;
- если обнаружится, что вы продвигаетесь быстрее, чем ожидали (такое случается);
- когда меняются условия, например люди, работающие над проектом;
- когда материализуется риск или пропадает зависимость.



Если не произойдет каких-либо изменений, подобных этим, то не следует изменять свое предсказание будущего.

Если ваша оценка не совпадает с чьими-то целями, ожиданиями или ограничениями, то вы должны совместно исследовать это расхождение. Не следует оставлять конфликт неразрешенным. Вы можете подвергнуть сомнению свои предположения, обсудить риски и попробовать альтернативные методы расчетов, чтобы подтвердить или отвергнуть оценку. Возможно, исходные данные, на которые вы опирались, плохо подходят для оценки рассматриваемой работы. Вы можете договориться об объеме, ресурсах или качестве, чтобы увидеть, получится ли более приемлемая оценка. Но если ваша оценка получена в результате тщательного анализа, то не уступайте просто ради того, чтобы кто-то улыбнулся. Люди перестанут доверять вашим прогнозам, зная, что могут изменить их, оказав достаточное давление.

Урок 29**Держитесь подальше от критического пути**

Основу планирования проекта составляет определение задач, которые необходимо решить. Многие из этих задач должны выполняться в конкретной последовательности, а часть из них связана друг с другом, поэтому планировщик должен также определить временные зависимости между задачами. Это все усложняет достижение цели.

Определение критического пути

Планировщики проектов часто рисуют *сетевой график* (также называемый диаграммой PERT), чтобы показать временные отношения между задачами. На рис. 4.3 показан такой сетевой график для проекта с шестью задачами, отмеченными буквами от А до Е. Показана расчетная продолжительность каждой задачи. Сетевые графики содержат гораздо больше информации, например самые ранние и самые поздние даты начала и окончания каждой задачи. Для простоты предположим, что нельзя приступить ни к одной задаче, пока не будут выполнены все предшествующие ей. То есть задача Е не может начаться, пока не будут выполнены задачи А, Г и Д.

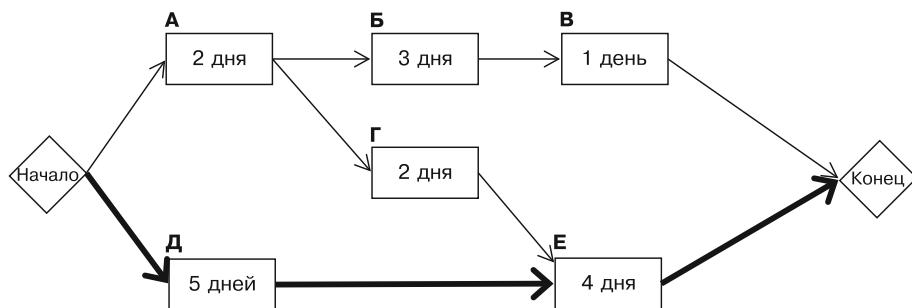


Рис. 4.3. Критический путь (жирные стрелки) — это самая длинная последовательность выполнения задач между началом и концом проекта



Если суммировать приблизительную продолжительность задач в различных путях, ведущих от начала до конца, то можно заметить, что путь ДЕ самый длинный: $5 + 4 = 9$ дней (жирные стрелки). Эта последовательность задач является *критическим путем* и определяет кратчайшую расчетную продолжительность реализации проекта (Cohen, 2018). Если на реализацию какой-либо задачи в критическом пути

потребуется больше времени, то дата завершения всего проекта сдвигается на время этой задержки. Новые задачи, добавляемые в критический путь, тоже задерживают завершение проекта.

Задачи, не лежащие на критическом пути, имеют *резервное время*. Они могут решаться долго, не задерживая весь проект, но, конечно же, до определенного момента. Например, задачи А и Г вместе имеют один резервный день: $2 + 2 = 4$ дня, что на один день меньше, чем продолжительность решения задачи Д, лежащей на критическом пути.

Однако критический путь может измениться. Предположим, что на решение задачи А потребовалось четыре дня, что вдвое превышает первоначальную оценку. Теперь критическим станет путь АГЕ протяженностью 10 дней ($4 + 2 + 4$), в результате чего срок окончания проекта на один день превысит девятидневный путь ДЕ.

Решение задач, добавляемых не в критический путь, требует дополнительных усилий, но такие задачи не задержат завершение проекта, если имеется достаточный резерв. Однако если для их решения нужно отвлекать людей, которые в противном случае работали бы над задачами критического пути, то эти дополнительные задачи могут увеличить его продолжительность.

Чтобы сократить срок реализации проекта, нужно найти способы ускорить решение задач критического пути.

Задачи на критическом пути не имеют резерва времени. Если вы хотите сократить срок реализации проекта, то найдите способы ускорить решение задач критического пути, например, решая некоторые из них параллельно. Опытные руководители проектов внимательно следят за критическим путем и стараются сделать так, чтобы задачи на этом пути были решены вовремя или даже раньше.



Держитесь в стороне

В своей работе я стараюсь держаться подальше от критического пути, так как не хочу оказаться слабым звеном, тормозящим чье-либо движение. Я расставляю приоритеты в своей повседневной работе, учитывая два аспекта: важность и срочность (Covey, 2020). Задача, лежащая

на критическом пути, более актуальна, чем задача, лежащая в стороне. Если моя задача может заставить других людей ждать, пока я ее закончу, то она перемещается в начало моей очереди приоритетов. Я стараюсь не быть причиной простоев и ожиданий, которые могут помешать всему проекту.

Эту же философию я применяю, когда пишу книги. Процесс замысла, планирования, предложения, написания, рецензирования, редактирования и публикации книги — это проект со множеством мероприятий, особенно если в нем участвуют несколько авторов. Большая часть работы может выполняться параллельно или в произвольной последовательности. Однако иногда один участник должен выполнить определенное действие, прежде чем кто-то другой сможет сделать следующий шаг. Например, я не могу считать главу завершенной, пока все мои бета-читатели (рецензенты) не предоставят свои отзывы. Если я не выделяю резервное время (я всегда выделяю его), чтобы не отставать от графика, то мне придется игнорировать запоздавшие рецензии, что приведет к пустой трате времени рецензента и, возможно, снижению качества книги.

Чтобы быстро сойти с критического пути, когда издатель присыпает мне макет для окончательной проверки, я бросаю все остальное и начинаю читать. Издатель не может вносить исправления или дорабатывать указатель, пока не получит мой ответ. Задержки в корректуре потенциально могут привести к задержке публикации в зависимости от того, сколько резервного времени мы добавили в график и определяем ли мы план в целом.

Соблюдение графика требует частого общения. Иногда две стороны ждут ответа третьей из-за путаницы в том, чья очередь действовать. Вы не получаете ответа на вопрос, отправленный по электронной почте, поэтому снова звоните или отправляете электронное письмо другому человеку: «Вы поняли мою просьбу в электронном письме, которое я отправил вам три дня назад? Или вы уже ответили, а я просто не получил вашего письма или проглядел ответ?» Один человек может задерживать другого, когда в этом нет необходимости. Есть много причин срыва графика проекта, когда люди жонглируют несколькими действиями, которые они должны закончить в определенной последовательности.

Я всегда испытываю положительные эмоции, когда быстро завершаю действие, которое, насколько мне известно, могло бы замедлить кого-то еще и, возможно, весь проект. Я стараюсь максимально быстро сойти с критического пути.

Урок 30

Задание либо полностью выполнено, либо не выполнено: частичное выполнение не засчитывается

«Привет, Фил, когда ты собираешься закончить реализацию этой подсистемы?»

«Очень скоро. Она готова примерно на 90 %».

«Подожди-ка, не ты ли говорил, что она готова на 90 %, еще пару недель назад?»

«Да, но теперь она *действительно* готова на 90 %!»

Заявление о том, что программный проект или задача уже давно готов(а) на 90 %, является чем-то вроде корпоративной шутки (Wiegers, 2019b). (В похожей шутке говорится, что первая половина программного проекта потребляет 90 % ресурсов, а вторая половина — оставшиеся 90 % ресурсов.) Такое оптимистичное, но вводящее в заблуждение заявление не позволяет оценить, когда данная часть работы завершится на самом деле. Мы все должны противиться искушению вычеркнуть пункт из нашего списка дел до того, как он будет полностью выполнен. Если вы говорите себе или кому-то еще: «Я закончил все, кроме...» — это значит, что вы еще не закончили.



Что означает «готово»?

Фундаментальный вопрос: что именно мы имеем в виду, когда говорим, что какая-то часть проекта готова? Выполнив все задачи, которые были прописаны в первоначальном плане, вы вдруг обнаружили, что работа требует больше усилий, чем предполагалось. Такое иногда случается при внесении изменений в существующую систему, поскольку постоянно появляются дополнительные компоненты, которые необходимо изменить. Планируя ту или иную часть работы, мы редко задумываемся обо всех необходимых действиях, и чем больше эта часть, тем выше вероятность, что мы что-то упустим из виду.

Если вы говорите себе или кому-то еще:
«Я закончил все, кроме...» — это значит,
что вы еще не закончили.



В Agile-сообществе этот вопрос решается путем включения *определения готовности* (Definition of Done, DoD) в план действий (Datt, 2020b; Agile Alliance, 2021a). Приступая к проекту, команда должна перечислить критерии, с помощью которых будут определяться завершенность конкретного элемента работы, задачи или итерации. Контрольные списки помогают установить минимальный объем работы, которую необходимо выполнить, чтобы считать конкретную задачу или приращение продукта завершенным.

Определение готовности дает объективные критерии, позволяющие оценить продвижение проекта или итерации к успешному завершению. «Готово» означает *полностью* выполнено. Например, дополнительный программный компонент полностью реализован, протестирован, интегрирован в продукт и задокументирован, то есть потенциально может быть выпущен для использования клиентами. Отдельные единицы работы, такие как реализация пользовательских историй, либо выполнены на 100 %, либо не выполнены; промежуточного состояния не существует (Gray, 2020).



Я часто пишу планы, в которых перечисляю шаги, связанные с выполнением повторяющихся действий. Эти планы уменьшают вероятность что-то упустить и помогают оценить, какой объем времени отвести на каждое действие. Письменные планы также позволяют отслеживать прогресс. В табл. 4.1 показана часть такого плана по преобразованию моих учебных курсов, оформленных в виде презентаций PowerPoint, в электронные курсы, как я описывал в уроке 27. Я использую столбец «Статус», чтобы отслеживать задачи, которые я начал, выполнил и к которым еще не приступил.



Таблица 4.1. Контрольный список некоторых задач, связанных с преобразованием презентаций PowerPoint в электронные обучающие курсы

Статус	Задача
	1. Разделить слайды курса на модули в отдельных файлах PowerPoint
	2. Расположить слайды для каждого модуля. Добавьте фон и колонтитулы
	3. Добавить ко всем слайдам примечания для инструктора
	4. Создать сценарии для всех слайдов с маркерами синхронизации анимации

Статус	Задача
	5. Записать аудиосценарии для всех модулей
	6. Импортировать аудиосценарии и синхронизировать анимацию
	7. Опубликовать все модули слайдов в формате электронного учебного курса
	8. Создать HTML-оболочку с меню модуля и другими ссылками
	9. Создать выдаваемые на руки материалы курса в формате PDF
	10. Протестировать всю презентацию
	11. Исправить ошибки, обнаруженные при тестировании
	12. Установить цены
	13. Разработать описание и рекламные материалы

Задачи в плане весьма сильно различаются по размеру. Установление цены (задача 12) занимает всего несколько минут; на запись аудиосценариев для всех модулей курса (задача 5) уйдет много часов. Мои курсы значительно отличаются по объему: от 4 до 18 модулей. Я хотел бы иметь более детальную информацию, чем та, которую дает форма в табл. 4.1, позволяющая отслеживать не только состояние, но и время, затраченное на задачи уровня модуля. Поэтому я создал электронную таблицу, позволяющую видеть, как я продвигаюсь по каждому из действий, необходимых для реализации любого модуля курса. Но единственные значения статуса, которые я использую, — это «ожидание», «в процессе» и «готово».

Не бывает частичной готовности

Одна из проблем оценки степени готовности — мы слишком часто считаем готовыми задачи, которые начали, но не завершили. Это помогает нам чувствовать себя счастливыми и чрезмерно оптимистичными. Однажды утром, обдумав алгоритм решения сложной задачи, вы можете решить, что выполнили около 30 %, поскольку алгоритм был сложной частью работы. Возможно, так оно и есть, но написание кода, его реview, тестирование и интеграция с приложением тоже могут потребовать много времени. Трудно точно оценить процент выполнения большой задачи. Может оказаться, что она намного больше, чем пред-



полагалось, или появится необходимость выполнить дополнительные действия, а кроме того, нет полной уверенности в том, как пойдет оставшаяся работа.



Первый шаг к решению проблемы определения готовности — разбить большие задачи на несколько маленьких подзадач, которые еще называют *дюймовыми камешками* (inch-pebbles) (Page-Jones, 1988; Rothman, 1999). На каждый дюймовый камешек может уйти от четырех до шести рабочих часов. Эта величина позволяет понять все, что нужно сделать, чтобы выполнить задачу. Будет полезно определять дюймовые камешки как действия, которые нельзя логически разделить на более мелкие части. Если у вас есть предыдущий опыт выполнения какого-либо действия, то вы наверняка сможете точно оценить, сколько времени понадобится для его выполнения. Для менее знакомых, более неопределенных или более сложных действий лучше подойдет разделение на более мелкие подзадачи — дюймовые камешки.

Следите за своим прогрессом по этим дюймовым камешкам в бинарной форме: «готово/не готово». Не бывает частичной готовности. Готовность незавершенной задачи — это ничто, нуль. Прогресс в большой задаче определяется количеством составляющих ее и полностью готовых дюймовых камешков. Такое отслеживание прогресса более информативно, чем попытки угадать выполненный процент большого и, возможно, неверно определенного объема работы (Rothman, 2004).

Под большой задачей я подразумеваю некую единицу работы, которая приносит пользу клиенту. Конечная цель моего проекта электронного обучения — получить готовый курс, который я мог бы продать. Разделение этой большой задачи на множество более мелких действий, как было показано в табл. 4.1, позволяло мне наблюдать за приближением к конечной цели. В рамках этой основной цели имелось несколько модулей курса, каждый со своим набором подзадач. Вместо того чтобы оценивать процент завершения всего проекта или отдельных подзадач, я использовал двоичное отслеживание работ «готово/не готово» на самом подробном уровне детализации.

Некоторые инструменты отслеживания проектов включают визуальный элемент, позволяющий сообщить процент выполнения. На рис. 4.4 показан пример диаграммы Ганта с несколькими задачами проекта. Серые столбики, подписанные как «Задача 1» — «Задача 4», составляют большую цель под названием «Создание модуля». Визуальный элемент, о котором я говорю, отображается в виде узких черных полос на серых

столбиках задач. Черные полосы показывают, какая часть каждой задачи выполнена. Я советую не использовать эти индикаторы. Они могут вводить вас в заблуждение, заставляя думать, что вы достигли большего, чем есть на самом деле. Я считаю, что подход, основанный на подсчете завершенных небольших дюймовых камешков, более точно сообщает о степени достижения результата.

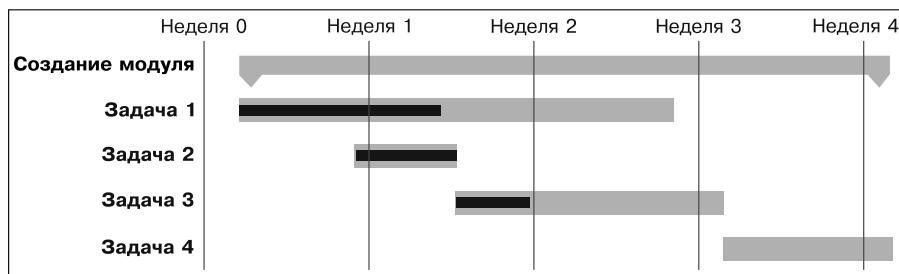


Рис. 4.4. Я не рекомендую использовать индикаторы процента выполнения отдельных задач (черные полосы внутри серых)

Отслеживание по статусу требований

Другой вариант мониторинга хода выполнения проекта — отслеживание статуса требований вместо оценки степени готовности реализации каждого требования (Wiegers, Beatty, 2013). Каждое требование (будь то функциональное требование, сценарий использования, пользовательская история, функция или подфункция), добавленное в объем работ, имеет определенный статус в данный момент времени. Возможные значения статуса: «предложено», «одобрено», «реализовано», «пропроверено», «отложено» и «удалено». При таком подходе запланированный объем работ считается завершенным, когда каждое добавленное в него требование имеет один из трех статусов:

- «пропроверено» (требование полностью реализовано и протестировано);
- «отложено» (требование было отложено для реализации позднее);
- «удалено» (требование больше не планируется к реализации).

Готовность ведет к ценности

Следить за продвижением проекта необходимо не только для того, чтобы убедиться в том, что сделана вся запланированная работа. Такой мониторинг также позволяет гарантировать выполнение отдельных



этапов, которые принесут пользу клиентам, когда они получат готовую реализацию. Лучший способ получить представление о том, насколько вы близки к достижению этой пользы, — изменять статус задачи на «готово», только когда она действительно завершена.

Урок 31

Команде проекта нужна гибкость в отношении хотя бы одного из пяти измерений: масштаба, плана, бюджета, персонала и качества

«Что вы хотите: хорошо, быстро или дешево? Выберите два пункта». Это разговорная форма описания треугольника ограничений, или железного треугольника, встречающегося во многих книгах, посвященных управлению проектами. Я видел несколько изображений треугольника с разными обозначениями вершин и ребер и различными предположениями о том, что остается постоянным. На мой взгляд, традиционный железный треугольник управления проектами слишком упрощен, хотя идея ограничений и компромиссов, безусловно, верна (Wiegers, 2019c).

Пять измерений проекта



В моем понимании, управляя проектами, группа должна помнить о пяти измерениях, показанных на рис. 4.5 (Wiegers, 1996). Во-первых, это масштаб или объем, описывающий функциональные возможности продукта. Важно отделять качество от масштаба. Я могу написать программу очень быстро, если она не должна работать правильно. Итак, в отличие от наиболее распространенных изображений железного треугольника, я показываю качество как отдельное измерение. Другие

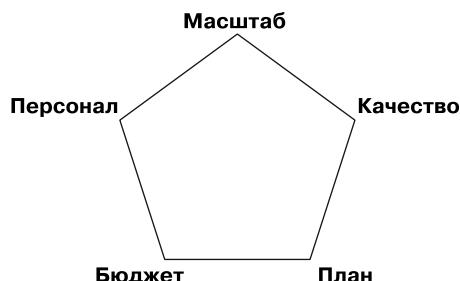


Рис. 4.5. Пять измерений программного проекта: масштаб, качество, план, бюджет и персонал

три измерения — это время, необходимое для реализации (план), бюджет (стоимость) и люди, реализующие проект. Некоторые добавляют шестое измерение — риск. Однако он не поддается регулированию, как другие пять параметров.

Люди часто объединяют персонал и бюджет в одно измерение — «ресурсы». Я предпочитаю разделять их. Большую часть стоимости проекта действительно составляет заработка персонала. Тем не менее иногда команда имеет достаточное финансирование, но ограничена численностью персонала. В этом случае руководитель проекта может использовать бюджет, чтобы купить комплексное решение, лицензировать некоторые программные компоненты, передать часть работ на аутсорсинг или нанять подрядчиков.

Для каждого проекта нужно решить, какие измерения являются наиболее важными, и сбалансировать другие так, чтобы основные цели были достигнуты. Поиск компромиссов между этими пятью измерениями — непростая задача. Например, по мере увеличения численности персонала стоимость может возрасти, а плановый срок — сократиться, хотя и не обязательно, как указывает Фредерик Брукс в законе Брукса: «Если проект не укладывается в сроки, то добавление рабочей силы задержит его еще больше» (Brooks, 1995). Распространенным компромиссом является сокращение графика или добавление функций за счет качества. Любой, кто стал жертвой ошибок в программном обеспечении, ставит под сомнение подобные компромиссы, но организации-разработчики порой делают такой выбор — иногда преднамеренно, иногда по умолчанию.

Каждое измерение может принимать одно из трех свойств проекта:

- ограничение;
- драйвер;
- степень свободы.



Ограничения определяют рамки, в которые должен уложиться проект. Проект негибок в отношении измерения ограничения. Если численность команды проекта не может изменяться, то это ограничение по персоналу. Стоимость — это ограничение, свойственное проектам, выполняемым по контракту с фиксированной ценой, по крайней мере, с точки зрения клиента. Качество — это ограничение для проектов, касающихся критически важных продуктов. Проекты, привязанные к контрактам или событиям с фиксированной датой (например, проблема 2000 года, выборы, Brexit), ограничены по сроку.

Драйвер — ключевая цель или критерий успеха проекта. Для продукта с желаемым маркетинговым окном возможностей плановый срок выпуска является драйвером. Коммерческие приложения часто имеют драйверы в виде возможностей, повышающих конкурентоспособность. Драйверы позволяют некоторым измерениям быть гибкими. Определенный набор возможностей может быть основным драйвером проекта, но если набор возможностей не подлежит обсуждению, то масштаб (объем) становится ограничением.

Любое измерение проекта, которое не является ни драйвером, ни ограничением, считается *степенью свободы*. Руководитель проекта может управлять степенями свободы в определенных пределах. Главная его задача в этом случае — так скорректировать степени свободы, чтобы достичь успеха проекта в пределах, устанавливаемых ограничениями. Например, проекты, использующие методы Agile-разработки, рассматривают масштаб (объем) как степень свободы, регулируя объем, реализуемый в каждой итерации, таким образом, чтобы уложиться во временные ограничения графика.

 У меня для вас плохие новости: проект с нулевой степенью свободы, скорее всего, потерпит неудачу. Все пять измерений не могут быть ограничениями и драйверами. Добавление новой возможности, уход члена команды, риск, который становится проблемой, слишком низкая оценка — все это рушит плановый график, поскольку из-за чрезмерных ограничений руководитель проекта не может своевременно реагировать на эти события.

 Один из студентов, слушавших мой курс по управлению проектами, сказал мне: «У нашего проекта фиксированный бюджет, мы не можем набрать больше людей, все функции важны, никакие дефекты недопустимы, и мы должны уложиться точно в срок». Я довольно пессимистично оценил шансы на успех этого сверхограниченного проекта.

Проект с нулевой степенью свободы, скорее всего, потерпит неудачу.

Соглашения о приоритетах

Важным аспектом этой модели является необходимость согласования относительных приоритетов измерений между командой, клиентами

и руководством в самом начале проекта. Например, график часто представляется как ограничение, хотя на самом деле это драйвер. Чтобы понять разницу, задайте вопрос: «Я понимаю, что вы хотите, чтобы решение было готово к 30 июня. Но что случится, если оно будет готово только к концу июля?» Если в ответ вы услышите, что тогда решение утратит свою ценность или последуют штрафные санкции, то график действительно является ограничением. Но если вам ответят: «Мы бы хотели получить решение к 30 июня, но можем потерпеть и до 31 июля, если придется», — график является драйвером.

Диаграмма гибкости

Диаграмма Кивиата, также называемая лепестковой или радиально-круговой диаграммой, помогает визуально представить степень гибкости по всем пяти измерениям. На рис. 4.6 показан пример. Диаграмма Кивиата, которую легко сгенерировать в Microsoft Excel, имеет несколько осей, исходящих из общей точки. Все оси имеют одинаковую длину и приведены к единому масштабу. В данном случае каждая ось представляет степень гибкости руководителя проекта в соответствующем измерении, поэтому я называю такие диаграммы *диаграммами гибкости* (Wiegers, 1996).

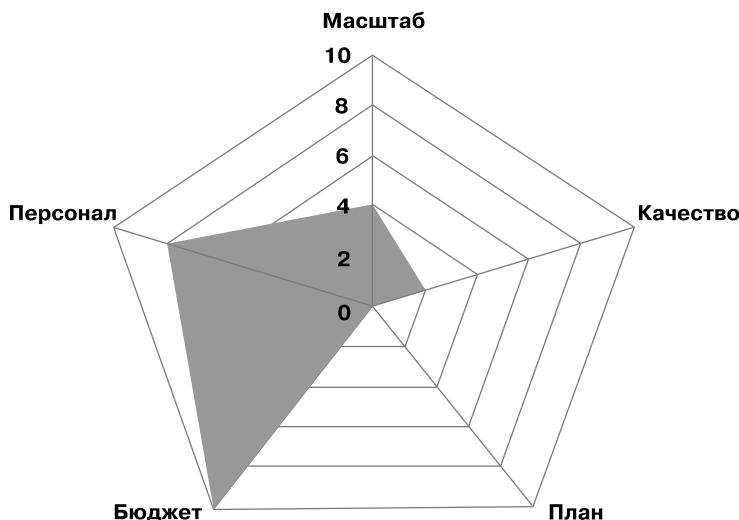


Рис. 4.6. Диаграмма гибкости для повторно используемого программного компонента показывает, что график является ограничением, качество и масштаб — драйверами, а персонал и бюджет — степенями свободы

Для измерения гибкости я использую относительную шкалу от нуля до десяти. Нулевая точка — начало координат — соответствует отсутствию гибкости, то есть данное измерение является ограничением. Точки в диапазоне от нуля до примерно четырех соответствуют драйверу, то есть в данном измерении проект имеет небольшую гибкость. Любая другая точка с более высоким значением соответствует степени свободы, когда данное измерение предлагает больше гибкости. Соединение точек, нанесенных на оси пяти измерений, дает пятиугольник неправильной формы. Диаграммы гибкости для разных проектов будут иметь разные формы.



На рис. 4.6 показана диаграмма гибкости многократно используемого программного компонента, разработанного моей командой. Время на разработку было ограничено, поскольку этот компонент должен был быть готов до того, как несколько других приложений, разрабатываемых параллельно, смогли бы его использовать. Поэтому измерение «план» получило нулевую гибкость. Надежность и правильность компонента были очень важны; соответственно, качество было важным фактором успеха. На диаграмме я оценил измерение «качество» двумя единицами гибкости. К контролльному сроку мы должны были реализовать базовый набор возможностей и затем постепенно расширять функциональность. Следовательно, масштаб проекта получил оценку гибкости — четыре. Нам предоставили значительную свободу в отношении бюджета и персонала, главное, чтобы все было сделано вовремя. Поэтому эти измерения получили высокие значения гибкости и были оценены как степени свободы.

Диаграмма гибкости не является ни точным, ни количественным инструментом. Форма пятиугольника визуально выделяет важные аспекты проекта, но мы не пытаемся вычислить площадь пятиугольника или что-то в этом роде. Однако размер пятиугольника дает приблизительное представление о том, с какой гибкостью имеет дело руководитель проекта. Маленький пятиугольник означает, что проект имеет несколько ограничений и драйверов, что усложняет путь к успеху.

Практическое применение анализа пяти измерений

Этот пятимерный анализ может помочь руководителю проекта решить, как лучше реагировать на меняющиеся условия или реалии проекта, чтобы достичь основных целей. Предположим, что персонал является ограничением. Если понадобится включить новые требования, то единственными параметры, которые потенциально могут измениться, — это



Урок 32

Если вы не контролируете риски своего проекта,
то они будут контролировать вас

Одна компания наняла меня, чтобы выяснить, почему недавний международный контрактный проект потерпел неудачу. Изучая записи проекта, я обнаружил, что команда вела список проектных рисков, следуя надежной практике управления проектами. Однако их ежемесячные отчеты о состоянии неизменно показывали два одинаковых и тех же незначительных риска, каждый из которых нес минимальную предполагаемую угрозу. Проект потерпел неудачу, а это означает, что неожиданно появились дополнительные риски, которые не были никем замечены. Руководители проекта не учли некоторые распространенные риски, характерные для сложных распределенных проектов: медленное принятие решений, проблемы с коммуникацией, изменение масштабов, неоднозначность требований, чрезмерно оптимистичные обязательства и т. д. Мы можем извлечь несколько уроков из этого опыта.



- Если вы определили всего два риска для проекта с многомиллионной стоимостью, значит, вы недостаточно тщательно изучили его.
- Недооценив потенциальную угрозу, которую может представлять риск, вы можете не уделить ему должного внимания.
- Если в течение нескольких месяцев список главных рисков остается неизменным, это значит, что либо вы недостаточно активно выявляете их, либо ваши усилия по смягчению последствий не дают должного эффекта.

Риск — это потенциальная проблема, которая еще не возникла.



Что такое управление рисками?

Риск — это условие или событие, которое может нанести вред вашему проекту (Wiegers, 1998a). Это потенциальная проблема, которая еще не возникла. Цель управления рисками — обеспечить успех проекта, несмотря на возможные негативные последствия рисков, с которыми он сталкивается. Управление рисками — важный компонент эффективного управления проектами. Говорят, что управление проектами, особенно крупномасштабными, — это управление *рисками* (Charette, 1996). Выражаясь более образно, управление рисками — это управление проектами для взрослых (DeMarco, Lister, 2003).

В управление рисками входят выявление пугающих условий и событий, оценка их влияния на проект в случае возникновения проблем, определение их приоритетности и попытки их контролировать. При формальном управлении рисками вы сосредоточиваете вашу энергию на наибольших из надвигающихся угроз. Нет смысла беспокоиться о чем-то, что не причинит большого вреда, если это произойдет или если вероятность происшествия очень мала. Никто не может предсказать будущее, но точно так же никто не хочет неожиданно оказаться в ситуации, которую можно было бы предвидеть и избежать.

Выявление рисков использования программного обеспечения

В проектах по разработке ПО может произойти удивительно много неожиданных вещей. Каждая проектная группа должна с самого начала и постоянно противостоять своим факторам риска. Если управление рисками не является чьей-либо обязанностью, то никто не будет ими управлять. В крупных проектах часто назначается отдельный сотрудник по управлению рисками, который несет главную ответственность за координацию деятельности, связанной с рисками. Ниже перечислены некоторые методы выявления потенциальных рисков для вашего проекта.

Мозговой штурм

Групповые встречи позволяют всем членам команды участвовать в выявлении рисков. Каждый из участников может поделиться своей точкой зрения и опытом, а также озвучить свои мысли о проблемах, которые не дают им спать по ночам. Для начала следует изучить любые допущения, в том числе использовавшиеся при составлении оценок, поскольку ничем не подкрепленные допущения могут содержать риск.

Я обнаружил, что многие из предполагаемых рисков, выявляемых в ходе таких встреч, действительно отражают реалии текущего проекта. Это не риски — это проблемы. А с существующими проблемами нужно бороться намного энергичнее, чем с потенциальными рисками.

Исследование списка рисков из публикаций

Еще одна стратегия — начать с изучения обширного списка рисков, полученного из книг по программному обеспечению. Книги Каперса Джонса и Стива Макконнелла содержат длинные списки рисков, свойственных проектам по разработке программного обеспечения. Эти списки не утратили актуальности и по сей день (Jones, 1994; McConnell, 1996). Списки различных рисков также можно найти в Интернете, например в Bright Hub PM (Bright Hub PM, 2009).

Просмотр длинного списка потенциальных рисков немного пугает, подобно чтению перечня всех потенциальных побочных эффектов некоторых лекарств. Не все риски применимы к вашему проекту, но их списки могут предупредить о возможностях, о которых вы даже не догадывались. Риски программных проектов можно объединить в следующие категории:

- требования и область применения;
- проектирование и реализация;
- организация и персонал;
- управление и планирование;
- заказчик;
- аутсорсинг и подрядчики;
- среда и процесс разработки;
- технология;
- юридические и нормативные требования.

Внутренняя хронология

Третья стратегия выявления рисков — изучение информации по предыдущим проектам, накопленной в вашей организации. Эти риски будут иметь для вас большее значение, чем те, которые указаны в общем списке. Изучение опыта предыдущих проектов дает возможность сорбать и записать как положительные, так и отрицательные уроки. События в проекте, удивившие команду, часто отражают риски, которых они не ожидали. Заманчиво думать, что какое-то неприятное событие было разовым, но все же добавьте его в основной список, чтобы при

разработке будущих проектов люди могли подумать и оценить, насколько это событие вероятно в их проектах.

Вместе с информацией о рисках внутри организации необходимо также записать, какие стратегии смягчения последствий были испробованы предыдущими командами для конкретных рисков и насколько хорошо они себя зарекомендовали. Участникам будущих проектов будет полезно оценить предыдущий опыт, чтобы быстро определить свои риски и решить, как их контролировать. Изучение предыдущего опыта помогает избежать необходимости преодолевать все болезненные этапы кривой обучения в каждом проекте.

Действия по управлению рисками



Управление рисками — не разовая акция, выполняемая в начале проекта. На рис. 4.7 показана блок-схема различных действий, сопутствующих управлению рисками.

Определение рисков. Просмотрите имеющиеся у вас списки рисков и отметьте все пункты, которые могут относиться к вашему проекту. В списках рисков обычно указывается условие, которое может создать проблему, например: «Неадекватные требования к отчетности со стороны внутреннего регулятора». Та-ак, и что? Мне нравится записывать риски в форме условия с возможным последствием: «Неадекватные требования к отчетности со стороны внутреннего регулятора могут привести к тому, что проект не пройдет аудит после развертывания».

Запись рисков в таком виде поможет показать, что одно условие может привести ко множеству последствий, и обеспечить эффективное управление им. Кроме того, одно и то же последствие может наступать из-за нескольких рисков. В таких случаях бывает трудно полностью избежать последствий, поэтому подумайте о разработке плана мероприятий в чрезвычайных ситуациях, этот план поможет справиться с последствиями, если это понадобится.

Оценка рисков и расстановка приоритетов. Составив список условий и их возможных последствий, подумайте, какой ущерб каждое из них может нанести вашему проекту. При оценке риска следует учитывать два аспекта.



1. Какова вероятность превращения риска в реальную проблему? Некоторые риски едва ли материализуются, а другие представляют явную и реальную опасность.

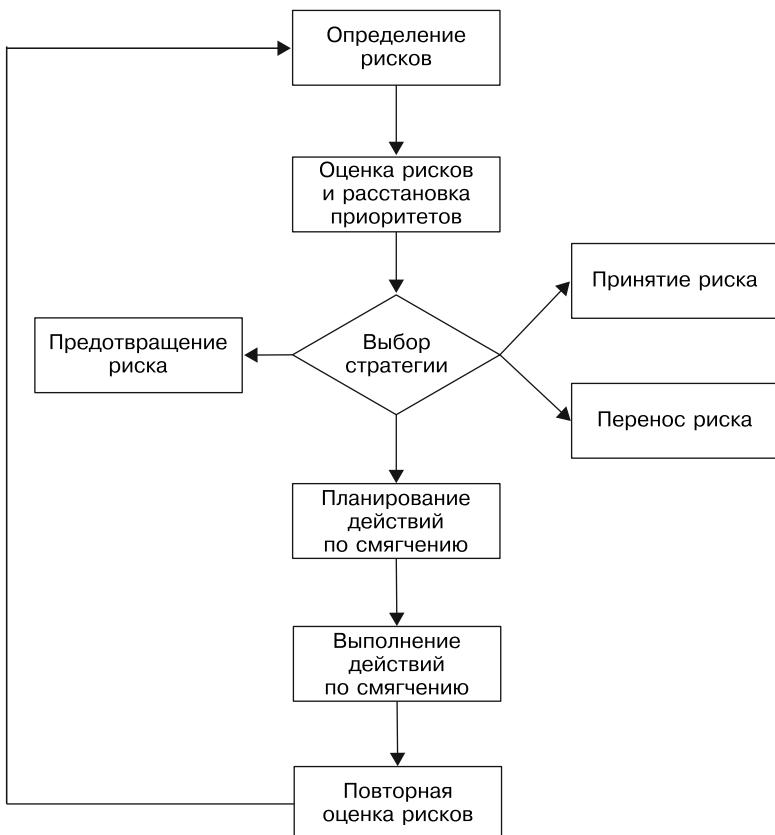


Рис. 4.7. Управление рисками связано с выполнением множества действий

2. Какое влияние проблема может оказывать на проект, если материализуется? Одни воздействия слишком незначительны, чтобы о них беспокоиться; другие могут быть разрушительными.

Я предпочитаю оценивать вероятность по шкале от 0 до 1, а разрушительность воздействия — по относительной шкале от 0 до 10. Умножение вероятности на воздействие дает оценку воздействия каждого риска.

Оценив подверженность рискам, отсортируйте список по убыванию подверженности, чтобы элементы с наибольшей угрозой располагались вверху. Этот шаг расстановки приоритетов поможет вам сосредоточить внимание на наиболее важных пунктах. Начните планирование управ-

ления рисками с начала отсортированного списка и продвигайтесь вниз. Опытные руководители проектов постоянно держат в центре своего внимания 10 (или около того) наиболее важных рисков.

Выбор стратегии. У вас есть четыре варианта реагирования на каждый риск. Первый вариант — просто принять риск. Да, он может материализоваться и иметь некоторые негативные последствия, но вы решаете не предпринимать никаких действий, а просто ждать и смотреть, что произойдет. Иногда у вас нет иного выбора. Например, если вы определили потенциальное изменение в государственном постановлении как риск, то ничего не сможете с ним поделать. Если вы вынуждены принять на себя риск, то подумайте о разработке плана мероприятий на случай непредвиденных обстоятельств. Второй вариант — предотвратить риск путем изменения направления, например, выбрав другие технологии или деловых партнеров, представляющих меньшую опасность. Третий вариант — передать то, что имеет отношение к риску, какой-либо другой стороне, чтобы больше не беспокоиться по этому поводу. Однако чаще всего вам придется выбрать четвертый вариант — попытаться снизить риск, чтобы уменьшить его воздействие.

Планирование действий по смягчению последствий. Данное планирование предполагает выбор действий, снижающих вероятность превращения риска в проблему или уменьшающих его воздействие, если это произойдет. Действия по смягчению становятся задачами проекта; кто-то должен отвечать за каждое из них. Попытайтесь также определить любые провоцирующие условия или события, которые могли бы предупредить о надвигающейся угрозе, а не только о потенциальной возможности.

Выполнение действий по смягчению. Даже великие планы бесполезны, если не выполняются. Следите за неукоснительным выполнением действий по смягчению так же, как за решением любой другой задачи в проекте, чтобы гарантировать их выполнение должным образом.

Повторная оценка рисков. Если ваши меры по смягчению последствий окажутся эффективными, то воздействие факторов риска должно уменьшиться. (В противном случае попробуйте разработать другие меры.) Как показано на рис. 4.7 (см. выше), управление рисками — итеративный, а не линейный процесс. Хорошей практикой считается еженедельная переоценка списка десяти основных рисков (McConnell, 1996). Риски, которые вы активно прорабатываете, должны переме-



щаться вниз по отсортированному списку, по мере уменьшения предполагаемой подверженности им. Просматривайте также элементы, находящиеся ниже в списке, чтобы увидеть, не изменился ли их статус. Риски могут полностью исчезнуть из списка, если соответствующее окно возможностей для нанесения ущерба закрылось. Кроме того, иногда условия проекта могут измениться так, что риск, который изначально был не слишком угрожающим, становится более вероятным или значительным. В любой момент могут появиться новые риски, поэтому всем участникам проекта следует быть настороже.

Всегда есть о чем беспокоиться

Риски существуют независимо от того, ищете вы их или нет. Неспособность управлять ими равносильна согласию с ними и принятию последствий. Я предпочитаю начинать противостоять угрозам как можно раньше. Меня радует, когда список активных рисков сокращается при успешном выполнении мероприятий по смягчению последствий.

Как вариант, вы можете просто игнорировать риски и надеяться, что ничего неприятного не произойдет. В таком случае мне остается лишь пожелать вам удачи.

Урок 33 Клиент не всегда прав

В нашем обществе модно говорить: «Клиент всегда прав». Это утверждение подразумевает, что если клиент что-то просит, то вы обязаны дать ему это. Возможно, вы видели на некоторых предприятиях плакаты, гласящие: «У нас есть только два правила. Правило № 1: клиент всегда прав. Правило № 2: если клиент не прав, см. правило № 1».

Клиент не всегда прав, но имеет свою точку зрения.

Реальность такова, что клиент *не* всегда прав. Иногда он дезинформирован, неразумен, растерян, в плохом настроении или не выполнил свою часть работы. Я предпочитаю говорить, что клиент не всегда прав, но имеет свою точку зрения, и мы должны понимать и уважать ее. Это не то же самое, что всегда выполнять все прихоти клиента.



Быть «неправым»

Помню, меня раздражал бесконечный ремонт дороги возле моего дома. Мне приходилось месяцами каждый день проезжать через перерытый участок. «Почему так долго ремонтируется этот участок?» — думал я каждый раз, проезжая через этот бардак. Тогда я понял важный момент: я ничего не понимаю в дорожном строительстве! У меня не было оснований оценивать, сколько времени нужно на то, чтобы восстановить этот участок. Ремонт раздражал, но я не мог требовать закончить его раньше, если это не было возможно. Я мог быть недовольным «покупателем» новой дороги, но это не означало, что мои ожидания были правильными. Рассмотрим несколько примеров, когда покупатель программного обеспечения может быть не прав.

Конфликт запросов

Предположим, что два клиента требуют противоречивых решений проблемы, поскольку, к примеру, по-разному представляют, как должна функционировать некая часть системы. Они не могут быть правы оба. И все же какие-то рассуждения привели их к соответствующим запросам. Мы должны понять, на чем основываются запросы, чтобы оценить, какой из них теснее связан с достижением бизнес-целей проекта. Это требование, которое мы должны удовлетворить.

Решения конфликтуют с потребностями

На этапе выявления требований представитель клиента может изложить не свои требования, а некие решения, сложившиеся у него в голове. Квалифицированный бизнес-аналитик может определить, когда заявленная потребность на самом деле является идеей решения, и задать соответствующие вопросы, чтобы выявить основную проблему. Иногда клиенты не до конца понимают смысл этого диалога. Их реакция может быть такой: «Я же сказал вам, что мне нужно. Разве вы не слышали, что клиент всегда прав? Позвоните мне, когда закончите». Такое отношение не способствует эффективному сотрудничеству в решении реальных проблем.

Заместитель представителя

Клиент может представить требования от имени класса пользователей, к которому он не принадлежит. Такое случилось со мной, когда я был ведущим бизнес-аналитиком в одном проекте. Двумя наиболее важными для нас классами пользователей были большое сообщество хи-

миков и несколько человек, работавших на складе химикатов. Женщина, управлявшая этим складом, сказала мне: «Несколько лет назад я работала химиком в лаборатории и могу предоставить вам все требования химиков к этой системе». К сожалению, она ошиблась. Ее понимание того, что сегодняшние химики ожидают от системы, было устаревшим и неполным. Было бы ошибкой полагаться исключительно на ее мнение. Поэтому мы нашли нескольких настоящих химиков, которые помогли нам понять их требования, что позволило добиться лучших результатов.



Та же проблема может возникнуть, если в роли представителя сообщества пользователей при обсуждении требований выступит руководитель. Он может не знать всех подробностей повседневной работы простых пользователей. Его опыт может устареть. В любом случае руководитель может оказаться не лучшим представителем пользователей при обсуждении требований.

Когда клиент действует в обход



Кто-то может попытаться обойти установленные правила, чтобы получить преимущество для себя. Однажды я спросил группу клиентов консалтингового сайта, как в их системе вносятся изменения в требования, после чего последовали неловкое молчание и обмен понимающими взглядами. Наконец кто-то сказал: «Если покупатель хочет внести изменения, то всегда говорит об этом Филиппу или Дебби. Он знает, что Филипп и Дебби добавят эти изменения, но это заставит остальных изрядно потрудиться».

В компании имелся механизм оценки запросов на изменение, но этот клиент попытался добиться своего другим путем. Люди определенно будут стараться обойти неэффективные и негибкие процессы, и иногда, возможно, это оправданно. Но в данном случае клиент просто не хотел утруждать себя прохождением процесса, который вполне мог закончиться отклонением его требований лицами, принимающими решения.

Поспешность



Моя коллега Таня, опытный инженер-программист и бизнес-аналитик, изначально пришла в компанию, чтобы заниматься уточнением требований и, возможно, реализацией новой информационной системы для автоматизации некоторых видов деятельности. Это был совершенно новый серьезный переход. Таня начала с изучения и докумен-

тирования текущих бизнес-процессов, выполняемых вручную. После этого она планировала разработать надлежащий набор решений. Пользователи, опасавшиеся перехода на автоматизированную систему, остались довольны работой Тани и благосклонно отнеслись к проекту.

К сожалению, руководитель по работе с пользователями взял у Тани увесистую папку с текущими материалами и сделал ошибочное заявление, что требования подготовлены, и решил купить существующий программный пакет, который по случайности продавал его друг. Этот новый пакет совершенно не подходил для данной ситуации и не отвечал потребностям пользователей. Если бы руководитель подождал, пока Таня уточнит реальные требования к продукту, он мог бы принять более правильное решение, позволяющее достичь желаемых бизнес-целей.

У кого больше полномочий, тот и устанавливает требования

Клиенты иногда настаивают, чтобы их требования получили наивысший приоритет из-за их организационного статуса или другого влиятельного положения в проекте. (См. урок 15 «Когда принимаете решение о добавлении функций, избегайте расстановки приоритетов по децибелам».) Эта настойчивость может стать проблемой, если люди запрашивают функциональность, которая не будет использоваться настолько часто, чтобы оправдать ее преимущество перед другими возможностями. Если человек обладает властью, это не значит, что он прав автоматически.



Тем не менее люди, занимающие высокое положение и имеющие гораздо более полную информацию о стратегических планах компании, чем команда разработчиков, могут расставлять приоритеты требований так, что это может показаться команде нелогичным. Понимание причин таких поступков поможет настроить всех на достижение надлежащих бизнес-целей.

Изменения не бесплатны

Типичный пример, когда клиент не всегда прав, — когда он просит добавить новые возможности или внести другие изменения, но ожидает, что цена и дата готовности останутся прежними. Клиенты относятся к этому так: «Изменения должны вноситься бесплатно; просто сделайте это». Это больше похоже на шутку, но такое действительно случается в реальной жизни.

Уважение точки зрения

Мы все клиенты в нашей повседневной жизни. Мы покупаем товары в магазинах и услуги у разных поставщиков. Мы не всегда правы, хотя нам хотелось бы думать иначе. Я мог бы обратиться к своему врачу, чтобы подтвердить правильность диагноза, к которому я пришел после поиска в Интернете по некоторым симптомам, и убедиться, что я совершенно не прав. Однажды я отвез свою машину в сервисный центр, чтобы заменить тормоза. Когда я забрал машину, механик сказал мне, что тормоза не нуждаются в замене, только в регулировке, и это было сделано бесплатно. В данном случае мне как покупателю было выгодно ошибиться.

Однако помните: клиент имеет свою точку зрения. Всегда есть причина, почему он просит или требует сделать что-то. Как производители программного обеспечения, мы не можем утверждать, что предлагаем лучшее решение, если не уважаем и не учитываем точку зрения клиента, не стремимся понять его и не удовлетворяем требование, если оно правильное. Когда клиент не прав, мы должны объяснить это со всем уважением и сопротивляться давлению сделать что-то неуместное только потому, что этого требует клиент.

Урок 34

Мы слишком часто принимаем желаемое за действительное

Реальность часто далека от идеала, который мы себе представляем; люди иногда хотят верить, что все не так, как есть на самом деле. Они могут подтасовывать факты и даже напрямую фабриковать их, но вера, о которой я здесь говорю, скорее связана с самообманом или необоснованным оптимизмом. В одних случаях это попытка убежать от реальности. В других — принятие желаемого за действительное.

Жизнь в стране фантазий

Например, мы можем сделать вид, что определили все соответствующие заинтересованные стороны проекта. Вообразить, что понимаем их цели и выявили все необходимые требования и другую информацию о проекте. Но всегда есть вероятность, что это не так. Мы могли недостаточно тщательно проанализировать длинный список тех, кто мог бы иметь

отношение к нашему проекту, или не нашли людей, способных точно описать потребности и ограничения каждой группы. Что, если бы мы не смогли найти людей, способных точно описать потребности и ограничения каждой группы? Все эти ситуации могут вызвать проблемы.

Даже пообщавшись с подходящими заинтересованными сторонами, мы воображаем (возможно, лучше сказать: «надеемся»), что получили правильные требования и довольно точно записали их, чтобы другие смогли с ними работать. Бывший президент США Рональд Рейган использовал русскую пословицу «Доверяй, но проверяй» при обсуждении договоров о сокращении вооружений с Советским Союзом. Эта идея применима и к программным проектам. Мы доверяем людям, с которыми работаем, но должны проверять их информацию и подтверждать, что работа, проделанная на основе этой информации, была выполнена правильно.

Программисты иногда воображают, что управляют проектами. Мы верим, что все продумали и наши оценки точны. Приятно думать, что объем проекта понятен и не выйдет из-под контроля, но это не всегда так. Нам кажется, что ни один из рисков и ни одно из неожиданных событий, разрушивших наши предыдущие проекты, не станут проблемой на этот раз. Возможно, так и будет, но мы наверняка столкнемся с новыми проблемами, поэтому должны предвидеть их и подумать, как реагировать на них. Всем участникам проекта нужна честная и точная информация, чтобы проект не сбился с пути. Руководители должны поощрять членов своей команды, чтобы они быстро сообщали хорошие новости, а плохие — еще быстрее.

Иrrациональное преувеличение

Люди, ожидающие, что работа над их следующим проектом пройдет быстрее и будет менее проблемной, могут быть уверены, что полученный дополнительный опыт команды принесет свои дивиденды. Они могут также просто очень верить в свою команду. Я слышал, как руководители говорят, что новые инструменты и методы позволят их командам значительно повысить продуктивность. Однако руководители не учитывают необходимость обучения и освоения материала, которые на некоторое время замедлят работу команды. Они также не задумываются о том, не повлияла ли на их ожидания маркетинговая шумиха. Они просто считают, что произойдет чудо и продуктивность повысится.

Мы можем верить, что в нашей команде собрались лучшие специалисты, хотя не каждая команда может иметь такой сильный состав. Некоторые компании действительно создают подразделения для разработки программного обеспечения, состоящие из настоящих талантов. Но это означает, что в других местах преобладают менее талантливые специалисты. Работая консультантом, я видел разные организации, причем некоторые оказывались на самых концах спектра по талантливости разработчиков.

Еще одна фантазия — вера в то, что члены команды могут посвящать работе над проектом 100 % своего времени и при этом находить время для самообучения, изобретения инноваций и расширения возможностей команды (Rothman, 2012). Думать так — большая ошибка. Как мы видели в уроке 23 «При планировании работ нужно учитывать разногласия», некоторые факторы заметно снижают эффективность рабочего времени. Руководитель, который думает, что полное использование рабочего времени над проектом осуществимо и желательно, отрицает реальность. Не имея возможности учиться, исследовать и улучшать предложения, команда не будет совершенствоваться. Ожидания получить лучшие результаты в следующий раз имеют под собой скучную основу.



Игры, в которые играют люди

Команды и организации иногда заявляют, что следуют определенному процессу или методологии, поскольку знают, что так и должно быть, или потому, что это звучит красиво. В действительности же они делают что-то другое. Они могут следовать одним правилам и игнорировать другие, которые считают неудобными, отнимающими много времени или трудными для понимания.

Однажды я консультировал правительственные учреждение, в котором сложилась интересная ситуация. Их проекты финансировались по двухгодичному графику, начинавшемуся 1 июля одного года и заканчивавшемуся 30 июня два года спустя. Они всегда завершали свои проекты в срок, к 30 июня. Если к тому времени система не была полностью готовой к внедрению, то выпускали неполную версию с ошибками, чтобы иметь возможность заявить об успешном выполнении графика. Затем платили за доработку и исправления в следующем цикле финансирования. Они свято верили, что ни разу не нарушили график, но я не думаю, что эта вера кого-то обманула.



Иногда я не в восторге от реальности, но она — все, что у меня есть, и я должен с этим жить.

Случается ли подобное в вашей организации? Если да, то каковы последствия? Можете ли вы как-то повлиять на это?

Я не большой любитель притворяться. Надежда на то, что мир не такой, какой есть на самом деле, может быть утешительной, но неконструктивной. Иногда я не в восторге от реальности, но она — все, что у меня есть, и я должен с этим жить. Да и все мы.



СЛЕДУЮЩИЕ ШАГИ: УПРАВЛЕНИЕ ПРОЕКТОМ

1. Определите, какие уроки, описанные в этой главе, имеют отношение к вашему опыту управления проектами.
2. Можете ли вы, опираясь на свой опыт, вспомнить какие-либо другие связанные с управлением проектами уроки, которыми стоит поделиться с коллегами?
3. Перечислите описанные в этой главе методы, способные помочь в решении связанных с управлением проектами проблем, которые мы определили во врезке «Первые шаги» в начале главы. Как каждый метод может улучшить управление проектами в ваших проектных группах?
4. Как вы определили, приносит ли желаемые результаты каждый метод, озвученный на шаге 3? Насколько ценные для вас эти результаты?
5. Определите любые препятствия, которые могут затруднить применение методов, перечисленных на шаге 3. Как бы вы справились с ними? Заручились бы поддержкой коллег, готовых помочь вам в реализации этих методов?
6. Внедрите описание процессов, шаблоны, руководящие документы, контрольные списки и другие инструменты, чтобы помочь будущим проектным группам эффективно применять ваш передовой опыт управления локальными проектами.

Глава 5

Культура и командная работа

ВВЕДЕНИЕ

Каждая организация, компания, подразделение и команда имеет свою культуру. Если говорить кратко, то под культурой понимается философия «как мы работаем». Здоровая культура разработки программного обеспечения характеризуется набором общих ценностей и технических практик, определяющих поведение и решения людей в организации (Wiegers, 1996; McMenamin et al., 2021). Здоровую культуру составляют обязательства на индивидуальном, командном и организационном уровнях по созданию высококачественных продуктов путем неукоснительного следования соответствующим процессам и методам. Если все идет хорошо, то члены команды получают удовольствие от работы.

Моя первая книга, опубликованная в 1996 году, называется *Creating a Software Engineering Culture*. В ней перечислены 14 общих ценностей (культурных принципов), которые приняла моя группа разработчиков программного обеспечения. Спустя годы я с той же уверенностью могу сказать, что все эти ценности важны для успешной разработки ПО. Некоторые из них представлены в этой книге в виде уроков.

- Урок 44. Высокое качество естественным образом ведет к повышению продуктивности.
- Урок 47. Никогда не поддавайтесь уговорам руководителя или клиента сделать работу наспех.
- Урок 48. Стремитесь к тому, чтобы дефект нашли коллеги, а не покупатели.
- Урок 60. Невозможно изменить все сразу.

Культура компании развивается органично, если только руководители активно не направляют ее в определенное русло. Случайный набор моделей поведения, сформировавшихся на основе прошлого опыта каждого сотрудника, вряд ли сможет спонтанно превратиться в здоровую культуру. Молодые компании, основным продуктом которых является программное обеспечение, часто устанавливают сильные культурные требования, наилучшим образом сказывающиеся на их командах и выполняемой ими работе (Pronschinske, 2017). И наоборот, ИТ-подразделение в нетехнологической корпорации наследует общие культурные особенности компании. Смена ИТ-культуры на более подходящую для современного умственного труда может оказаться контрпродуктивной. Работа в ИТ имеет свои особенности, отличающие ее от работы в других сферах, поэтому логично, что культура в них должна развиваться в другом направлении, желательно совместимом и дополняющем.



Например, я работал в крупной компании по производству потребительских товаров, где процесс принятия решений был заморожен. Казалось, что никто, кого так или иначе затрагивала какая-либо часть решения, не может его принять, не будучи на 100 % согласным с решением в целом. Казалось, что все, кто участвовал в принятии решения, имеют право вето и активно пользуются им. Единодушие — это хорошо, но бизнес должен двигаться вперед — такой подход к принятию решений нельзя использовать в быстро развивающемся программном проекте. Некоторые аспекты корпоративной культуры вошли в противоречие с требованиями современной гибкой и быстро меняющейся ИТ-сфера.

Сохраняя веру



Руководители влияют на культуру организации через свое видение, высказывания и поведение. Принимая различные меры по укреплению культуры, руководители способствуют командной работе и соответствуя позитивным культурным ценностям. Но культура — вещь хрупкая. Действия, направленные на подрыв культуры, могут разрушить фундамент, ориентированный на высокое качество работы, который постепенно создавала команда (Wiegert, 1996; McMenamin et al., 2021).



Характерный признак укоренения здоровой культуры в организации, — сохранение новых взглядов, методов и моделей поведения даже после ухода ключевых руководителей. Небольшая команда разработчиков программного обеспечения, которой я руководил в течение нескольких лет, значительно улучшила совместную работу, и мы увидели, как это отразилось на качестве поставляемых нами систем. За три года численность

команды увеличилась с 5 до 18 человек. Присоединяясь к нам, новые сотрудники впитывали наши культурные ценности и помогали поддерживать правила и методы работы, которые мы считали важными. В конце концов мы наняли нового руководителя, чтобы заменить меня, так как мне не нравилось быть руководителем. Одновременно к команде присоединились еще несколько новых сотрудников.

К сожалению, новый руководитель не полностью разделял наше стремление постоянно совершенствовать процесс разработки программного обеспечения и мою философию совершенствования, основанную на «мягком и неустанном давлении» (см. урок 54). Если руководители не сохранят курс на постоянное совершенствование, то некоторые члены команды могут вернуться в привычную для них зону комфорта. В результате некоторые процессы, которым мы следовали, постепенно стали неактуальными. Но члены команды все еще практиковали другие процессы, которые они считали лучшими подходами к работе.



Культурная конгруэнтность

Конгруэнтность — важнейший элемент здоровой культуры. Конгруэнтность означает, что руководители и рядовые сотрудники действуют согласно заявлениям организации о ценностях, а не в соответствии с негласными правилами, которые могут противоречить официальным заявлениям (McMenamin et al., 2021). Неконгруэнтность заражает культуру, подрывая заявленную ориентированность на высокие стандарты качества и этику поведения. Ответы на вопросы, подобные следующим, могут показать, является ли культура конгруэнтной.



- Следуют ли руководители убеждениям, которые они проповедуют, или могут отступать от них под давлением извне, например, выпуская продукты, которые не соответствуют высоким критериям качества?
- Следуют ли технические специалисты установленным процессам или могут экономить на качестве перед лицом надвигающихся сроков?
- Берут ли члены команды на себя реально достижимые обязательства и выполняют ли их или дают чрезмерно амбициозные обещания, которые часто остаются невыполнеными?

Если ваша организация имеет другой набор ценностей, то вы можете подобрать другие вопросы для проверки культуры на предмет конгру-

этного поведения. Независимо от того, какие принципы провозглашает команда, люди должны действовать в соответствии с ними.



Поведение, которое вознаграждается руководителями организации, может служить четким признаком их истинных ценностей. Одна компания одновременно занималась двумя крупными проектами по замене устаревших систем (подробнее об этом случае рассказывается в уроке 44 «Высокое качество естественным образом ведет к повышению производительности»). Команда А экономила на проектировании и сосредоточилась на программировании; их система каждый день давала сбои. «Отряд командос», сформированный командой А для восстановления работоспособности системы после каждого сбоя, получил похвалу от руководства за выдающееся обслуживание клиентов. Команда Б, напротив, построила свою систему, твердо придерживаясь принципов разработки качественного программного обеспечения, из-за чего задержала выпуск на несколько месяцев, но их система прекрасно работала. Эта команда не получила ни награды, ни признания.



Руководство явно отдавало предпочтение героям из команды А, которые тушили пожары, а не специалистам из команды Б, которые незаметно предотвращали пожары, следуя методам разработки, гарантирующим достижение высокого качества. Моральный дух в команде Б серьезно пострадал, когда руководство отнеслось к ним как к неудачникам из-за опоздания, восхваляя героические усилия команды А. Но, когда система команды А рухнула, моральный дух команды Б улучшился, поскольку они гордились своими достижениями. С какой группой вы бы предпочли работать? Я выбрал бы команду Б.

Кристаллизация культуры

Культура организации обычно неосознана; это негласное понимание особенностей совместной работы и важных для них принципов. Некоторые компании систематизируют важные черты своей культуры в виде руководств для сотрудников (Pronschinske, 2017). Подобная публичная коммуникация помогает держать культурные ценности на виду и облегчает их передачу новым членам команды.

Явное обозначение культурных факторов помогает всем членам команды проникнуться общими ценностями, что ведет к улучшению совместной работы.



В своей книге *Software Teamwork* Джим Броско рекомендует командам заключить своего рода контракт, соглашение, определяющее общие ценности, правила взаимодействия и нормы поведения (Brosseau, 2008). Каждая команда должна выработать свой контракт, а не принимать контракт, навязанный руководством, заимствованный у другой команды или обнаруженный в Интернете. Контракт должен отражать специфику конкретной команды, но при этом должен быть связан с другими командами и компанией в целом. Контракт должен быть рабочим, живым документом, к которому люди периодически обращаются и обновляют по мере необходимости. Он может содержать заявления, подобные перечисленным ниже (Resologics, 2021).

- Для принятия решений на основе консенсуса мы проводим дебаты, уважая при этом мнения друг друга.
- Мы вовремя приходим на встречи и другие запланированные мероприятия и соблюдаем повестку дня.
- Члены команды должны стремиться завершить к установленным срокам все задачи, которые взяли на себя.
- Мы можем расходиться во мнениях внутри команды, но вовне представляем единую позицию.
- Мы приветствуем различные точки зрения и противоположные мнения, чтобы максимизировать коллективное творчество.

Явное обозначение культурных факторов помогает всем членам команды проникнуться общими ценностями, что ведет к улучшению совместной работы. Наличие явного контракта позволяет членам команды уточнять и поддерживать его с течением времени. Контракт помогает новым членам команды адаптироваться и понять, какой вклад они могут внести в существующую культуру. Команды разработчиков программного обеспечения наиболее продуктивны, а их члены наиболее счастливы, когда удовлетворены следующие их основные потребности (Hilliard, 2018):

- безопасная и комфортная физическая среда;
- атмосфера честности и открытости для совместной работы;
- эмоциональная сплоченность команды и взаимная поддержка;
- сложная и целенаправленная, но выполнимая работа;
- правильно подобранные инструменты;
- самоопределение и независимость в работе, которую они выполняют;
- возможность внести свой вклад и профессиональный рост.



IT — необычная техническая дисциплина, поскольку привлекает людей с широким спектром знаний, характеристик и взглядов и выигрывает от их присутствия. Помимо очевидной ценности этнических, расовых, гендерных, возрастных различий и различий в способностях, люди, имеющие опыт в математике, инженерии, естественных науках, творческом дизайне, психологии, бизнесе и других областях, привносят что-то свое в проект и культуру (Mathieson, 2019). Группа разработчиков, состоящая из опытных экспертов в области технологий, только выигрывает от присутствия в ней нетехнических специалистов с сильными коммуникативными навыками и знанием предметной области.

Увеличение команды

Эффективная командная работа требует согласования общих целей и механизмов для их достижения. Новые члены команды привносят свой культурный багаж, как положительный, так и отрицательный. Проводя собеседование с кандидатами на вступление в команду, постарайтесь оценить, насколько хорошо они впишутся в вашу культуру.



Однажды я проводил собеседование с разработчицей по имени Даниэлла. Она имела солидный технический опыт, но была не согласна с нашей давней практикой ежедневной записи информации о том, куда было потрачено время. Я объяснил, что эти данные используются, только чтобы понять, как протекает работа над проектом, и помочь улучшить ее планирование. Даниэлла сказала, что просто не стала бы этого делать, но не объяснила почему. Может быть, у нее был предыдущий неудачный опыт с руководителем, который неправильно использовал эту информацию для поощрения или наказания отдельных лиц. Я оценил ее честность, но Даниэлла не получила предложения от нас. Я не хотел рисковать, нанимая кого-то, кто сразу обозначил свой культурный конфликт с командой. В ту пору я нанял еще трех разработчиков, которые быстро вписались в коллектив и внесли конструктивный вклад в непрерывное развитие нашей культуры.



Важно разобраться, почему человек возражает против определенных аспектов вашей нынешней культуры или направления, в котором она движется. На то может быть веская причина, которую вы еще не заметили. Возможно, ему знаком лучший подход, используемый в другом месте, или, может быть, он столкнулся с некоторыми недостатками, проявляющимися в долгосрочной перспективе, которых вы еще не достигли. Одно это может помочь повысить культуру команды.



Часто можно быстро определить, впишутся ли некоторые люди в вашу команду. Вскоре после того, как к нам присоединился разработчик по имени Гаутам, мы устроили совместный обед. Другой член команды, Энджи, всегда тщательно следила за своим питанием. Она никогда не заказывала десерт на наших случайных обедах, но ей нравилось пробовать чужие десерты. Когда принесли десерт Гаутама, он молча передал свою тарелку Энджи. Тогда я понял, что он отлично впишется, и так и вышло.

Несколько лет спустя я перешел в другое корпоративное подразделение, где Гаутам вырос в умелого руководителя. Он понимал, как эффективно руководить командами, и я не испытывал никакого дискомфорта, отчитываясь перед ним. В этой главе я опишу семь уроков о культуре и командной работе, которые усвоили мы с Гаутамом.

ПЕРВЫЕ ШАГИ: КУЛЬТУРА И КОМАНДНАЯ РАБОТА

Прежде чем вы перейдете к изучению уроков, связанных с культурой и командной работой, предлагаю вам потратить несколько минут на следующие действия. По мере чтения подумайте, в какой степени каждый из этих пунктов применим к вашей организации или команде.

1. Можно ли сказать, что в вашей организации здоровая культура разработки программного обеспечения? Почему да или почему нет?
2. Попробуйте перечислить предпринимаемые руководителями или членами команды модели поведения и действия, которые усиливают внимание к культуре, ориентированной на качество.
3. Наблюдали ли вы действия, убивающие культуру, негативно влияющие на отношения, мораль, поведение или работу членов команды? Воспрепятствование повторению подобных действий, убивающих культуру, — очевидное начало пути к улучшению вашей рабочей обстановки.
4. Насколько хорошо вы понимаете культуру вашей компании? Соответствует ли культура вашей команды культуре компании? Если нет, то что можно сделать, чтобы уменьшить разрыв?
5. Определите любые проблемы (болевые точки), которые можно отнести к недостаткам вашей культуры и того, как люди и команды взаимодействуют между собой. Каковы материальные и нематериальные издержки этих проблем?
6. Укажите, как каждая проблема влияет на вашу способность успешно завершать проекты. Как все они мешают достижению успеха в бизнесе?



се и организации, и ее клиентам? Недостатки в культуре могут привести к тому, что люди перестанут обмениваться информацией с другими участниками, не будут принимать или выполнять обязательства или начнут игнорировать установленные процессы. Моральные проблемы и текучесть кадров указывают на то, что культура имеет некоторые изъяны.

7. Для каждой проблемы, выявленной на шаге 5, определите основные причины, провоцирующие или усугубляющие ее. Проблемы, влияния и первопричины могут сливатся, поэтому постарайтесь разделить их и увидеть, как они связаны. Вы можете найти несколько основных причин, способствующих появлению одной и той же проблемы, или несколько проблем, обусловленных одной общей причиной.
8. Читая эту главу, перечислите любые практики, которые могут быть полезны вашей команде.

Урок 35

Передача знаний не ведет к проигрышу



Одно время я работал с разработчиком программного обеспечения по имени Стефан. Он ревностно охранял свои знания. Когда я задал ему вопрос, то почти видел, как у него в голове крутиятся колесики: «Если я дам Карлу полный ответ, то он будет знать по этой теме столько же, сколько и я. Это неприемлемо, поэтому я дам ему половину ответа и посмотрю, удовлетворится ли он этим». Вернувшись к Стефану позже в поисках более полного ответа, я мог бы получить оставшуюся половину. Так я постепенно приближался к получению полного ответа на свой вопрос.

Меня раздражала необходимость получать от Стефана информацию по крупицам. Я никогда не обращался к нему, чтобы узнать что-то конфиденциальное. Мы оба работали в одной компании, поэтому должны были быть настроены на совместный успех. Стефан, по-видимому, был не согласен со мной в том, что свободный обмен знаниями с коллегами является признаком здоровой культуры.



Знания не похожи на другие товары. Если у меня есть 3 доллара и я дам вам 1, то у меня останется только 2 доллара. Деньги — это игра с нулевой суммой в том смысле, что я должен потерять часть из них, чтобы вы что-то выиграли от этой сделки. Но со знаниями дело обстоит иначе: если я передам вам часть своих знаний, то не потеряю их. Я могу

поделиться ими с другими людьми, как и вы. Этот расширяющийся круг знаний приносит пользу каждому, кого он коснулся.

Здоровая организация способствует культуре свободного обмена знаниями и непрерывного обучения.

Пожиратель знаний

Некоторые люди неохотно передают свои знания из-за неуверенности. Они опасаются, что, поделившись с другими своими драгоценными знаниями, добытыми с таким трудом, станут менее конкурентоспособными. Есть и такие, которые просто хотят польстить вам своим обращением. Вопрошающий признает ваш опыт и владение темой. В редких случаях кто-то может запросить у вас информацию, поскольку не хочет тратить время на самостоятельные исследования. Конечно, вы не должны делать за них чужую работу, но помните, что вы и ваши товарищи по команде работаете для достижения общих целей.

Другие люди тщательно охраняют свои знания, стремясь получить гарантии своей востребованности. Если никто другой не знает того, что знают они, то компания не сможет их уволить, поскольку потеряет слишком много специфических знаний. Может быть, эти люди думают, что должны получить прибавку к зарплате, поскольку являются единственными обладателями такой важной информации.

Люди, скрывающие организационные знания, представляют опасность. Они создают информационное узкое место, которое может помешать работе других людей. Мой коллега Джим Броссо метко называет такое припрятывание знаний *техническим штрафом* (Brosseau, 2008). Скрытие информации является отличной практикой для проектирования программного обеспечения, но плохо подходит для команд разработчиков.

Исправление невежества

Здоровая организация способствует культуре свободного обмена знаниями и непрерывного обучения. Обмен знаниями повышает продуктивность каждого, поэтому руководство должно вознаграждать тех,



кто свободно передает свои знания, а не тех, кто держит их при себе. В организации, поощряющей обучение, члены команды уверены, что могут без опаски задавать вопросы (Winters et al., 2020). Никто из нас не знает всего во вселенной, поэтому давайте учиться у коллег, когда представляется такая возможность.

Опытные сотрудники могут передавать свой опыт разными путями. Самый очевидный — просто отвечать на вопросы. Но эксперты должны предлагать не только ответы, но и вопросы. Эти люди должны выглядеть доступными для коллег, особенно новичков, и быть вдумчивыми и терпеливыми, когда кто-то донимает их расспросами. Помимо простой передачи информации, эксперты могут делиться своими мыслями о том, как применять знания в конкретных ситуациях.

В некоторых организациях существуют официальные программы наставничества, помогающие ускорить адаптацию новых сотрудников (Winters et al., 2020). Закрепление новых сотрудников за опытными коллегами-наставниками значительно ускоряет обучение. В начале своей профессиональной карьеры в качестве химика-исследователя я стал первым подопытным кроликом новой программы наставничества. Мой наставник Сет был ученым из команды, к которой я присоединился, но он не входил в число моих руководителей. Я чувствовал себя комфортно, задавая Сету вопросы, которые в ином случае мне было бы неловко задавать из-за боязни показаться невеждой перед моим руководителем. Сет помог мне освоиться в новой для меня технологической области. Программа наставничества ускоряет обучение новых членов команды и помогает сразу же начать строить с ними отношения.



Расширение масштабов передачи знаний

Общение один на один эффективно, но плохо масштабируется. Опытные и талантливые специалисты пользуются большим спросом, не только из-за своей работы, но и благодаря опыту, которым они могут поделиться. Для развития культуры обмена знаниями подумайте, как эффективнее использовать информацию, получаемую от таких сотрудников, кроме как в общении один на один. Далее я перечислю несколько вариантов.

Технические беседы



Однажды моя команда разработчиков решила изучить множество отличных примеров программирования из классической книги Стива Макконнелла *Code Complete* (McConnell, 2004). Мы по очереди изучали

определенный раздел, а затем описывали его коллегам за обедом. Такая неформальная форма группового обучения позволяла эффективно распространять передовой опыт программирования в команде. Она способствовала формированию общего понимания методов и общего словарного запаса.

Презентации и обучение

Официальные технические презентации и обучающие курсы — еще один хороший способ передачи знаний в организации. Работая в Kodak, я разработал несколько учебных курсов и много раз проводил по ним занятия. Если вы создаете внутреннюю программу обучения, то подберите несколько квалифицированных инструкторов, чтобы одному человеку не приходилось все время читать одни и те же курсы.

Документация

Письменные знания охватывают весь спектр от документации по конкретным проектам или приложениям до широко применимых технических руководств, учебных пособий, списков ответов на часто задаваемые вопросы и советов. Кто-то должен написать эту документацию, но, работая над ней, этот человек не сможет участвовать в создании других компонентов проекта. Письменная документация — очень эффективный организационный актив при условии, что члены команды воспринимают его как полезный ресурс.

Я знал людей, предпочитавших делать открытия самостоятельно, а не искать знания в существующей документации. Эти люди не следовали уроку 7 «Запись знаний обходится дешевле, чем повторное их обретение». После того как кто-то потратит время на создание актуальной и полезной документации, гораздо проще прочитать ее, чем обретать знания заново. Все члены организации должны иметь возможность обновлять такие документы, чтобы они оставались цennыми источниками коллективного опыта.



Готовые шаблоны и примеры

Когда я работал в крупной организации по разработке продуктов, наша команда, занимавшаяся совершенствованием процессов, создала обширный онлайн-каталог с высококачественными шаблонами и примерами результатов многих проектов (Wiegers, 1998b). Мы тщательно изучили подразделения компании, занимавшиеся разработкой программного обеспечения, и отобрали коллекцию хороших спецификаций требований, проектной документации, планов проектов, описаний



процессов и других элементов. Эта коллекция «положительного опыта» давала ценный толчок всякий раз, когда тому или иному программисту в компании требовалось создать какой-нибудь новый артефакт проекта.

Технические обзоры

Обзоры действующего программного обеспечения тоже могут служить неформальным механизмом обмена техническими знаниями. Это отличный способ посмотреть, как работает другой человек, и позволить ему понаблюдать за вашей работой. Из каждого обзора, в котором я участвовал в роли обозревателя или автора обозреваемого компонента, я извлекал что-то полезное. Техника парного программирования, когда два человека пишут код вместе, обеспечивает форму мгновенной экспертной оценки, а также обмена знаниями между программистами. Подробнее об обзорах я расскажу в уроке 48 «Стремитесь к тому, чтобы дефект нашли коллеги, а не покупатели».



Дискуссионные группы

Вместо того, чтобы пытаться найти человека, способного ответить на ваш вопрос, можно опубликовать вопрос в группе обсуждения или групповом чате вашей компании. Признаваться в отсутствии знаний перед большим сообществом может быть неудобно. Вот почему важно развивать культуру, в которой поощряется желание задавать вопросы и вознаграждаются те, кто помогает получить на них ответ. Не стыдно не знать, стыдно не учиться.

Участники обсуждения могут быстро предложить несколько вариантов ответов на ваш вопрос. Опубликованные ответы доступны всем участникам обсуждения, что способствует дальнейшему распространению знаний. Вероятно, вы — не единственный, кто не знал ответа на конкретный вопрос, поэтому, задав его, вы могли помочь не только себе, но и другим. У меня есть друг — самый любопытный человек, которого я знаю. Он готов расспрашивать любого, кого встречает в повседневной жизни, о том, что этот человек делает и чего мой друг не знает. Так он научился многому, и люди, которых он расспрашивал, всегда были рады поделиться своими знаниями.

Здоровая информационная культура



Каждый может чему-то научить и чему-то научиться. Будучи руководителем команды разработки программного обеспечения, я нанял аспиранта информатики на время летних каникул. Признаюсь, снача-

ла я скептически отнесся к его практическим знаниям и умениям разработки ПО. Точно так же он с некоторым недоверием отнесся к процессуальному подходу, практиковавшемуся в нашей команде. Однако всего через несколько недель я проникся уважением к его знаниям в области программирования. А он понял, как разумно организованные процессы могут помочь повысить эффективность команды. Наша взаимная готовность делиться своими знаниями принесла пользу всем нам.

Необязательно быть экспертом с мировым именем в какой-то теме, чтобы быть полезным. Нужен лишь какой-то ценный блок знаний и готовность поделиться ими. В мире технологий, если вы на неделю опережаете другого человека в какой-либо области, то вы волшебник. Кто-то еще, несомненно, опередит вас в других областях, поэтому воспользуйтесь их знаниями. Люди в здоровой культуре обучения делятся тем, что они знают, а также признают, что кто-то другой может знать более хороший способ.



Урок 36

Как бы сильно на вас ни давили, не берите на себя обязательства, которые не сможете выполнить

В середине 1990-х я возглавил официальную инициативу по совершенствованию процесса разработки программного обеспечения в подразделении, где трудились 450 инженеров, создававших продукты для обработки цифровых изображений со встроенным ПО. Как и многие крупные организации того времени, мы использовали модель зрелости процессов разработки программного обеспечения (Capability Maturity Model for Software, СММ) для направления наших усилий (Paultk et al., 1995). Модель СММ — пятиуровневая структура, помогающая организациям совершенствовать методы разработки и управления ПО. Мы с моим руководителем встретились с директором подразделения Мартином, чтобы обсудить статус, цели и планы совершенствования процесса.



Мартин засомневался в реальности предложенного мной графика. Тщательно оценив текущее состояние нашей крупной организации и пробелы, которые необходимо было ликвидировать для достижения цели, наша команда пришла к выводу, что достичь желаемой цели вполне реально за 18 месяцев. Мартин, который, как мне кажется, не осознавал, какие трудности поджидают нас на пути, потребовал уложиться в шесть.

Он хотел стать первым среди директоров, достигшим следующей вехи процесса. Я объяснил, почему мы считаем этот срок нереальным. Будучи руководителем, настроенным на победу, Мартин ответил: «Не говорите мне, что вы не можете этого сделать. Скажите мне, чем я могу помочь, чтобы вы могли сделать это». Звучит обнадеживающе, но его готовность оказать любую помощь не решит проблему волшебным образом. Мартин неохотно предложил 12 месяцев, но я был уверен, что при любых условиях цели можно достичь не ранее чем через 18 месяцев.

Он продолжал давить на меня, чтобы я пообещал то, чего никак не мог выполнить, и наконец я сказал: «Извини, Мартин, но я не смогу этого сделать». Он уставился на меня.

«Ты не сможешь этого сделать, — ошеломленно произнес Мартин. — Хм-м-м». Как будто никто и никогда раньше не противился его давлению, и теперь он не знал, что ответить. Мартин неохотно принял предложенный нами срок, после того как я заверил, что мы сделаем все, чтобы уложиться в него, и придем за помощью, когда она понадобится. Позже, вернувшись в офис, я услышал, как мой руководитель, который был хорошим парнем, говорил людям: «Карл сказал Мартину, что не станет связывать себя невыполнимыми обязательствами!» Мой руководитель доверял моему мнению.



Мой пульс участился, когда Мартин обратился ко мне. Однако с моей стороны было бы неэтично и непрофессионально брать на себя обязательство, которое я не смогу выполнить. С моей стороны также неправильно было бы давать обещание, оказывающее чрезмерное давление на членов моей команды и отвергающее результаты проведенного нами анализа.

Люди редко со всей серьезностью относятся к обязательствам, которые им пришлось взять на себя под принуждением. Они также не берут на себя серьезных обязательств, которые кто-то другой берет на себя от их имени без консультаций и переговоров. Если сотруднику навязать невыполнимое обязательство, это вызовет у него стресс, а не приведет к чуду. Более того, люди могут даже уменьшить свои усилия, зная, что не могут достичь поставленной цели. Зачем убиваться, если дело все равно обречено на провал?

Обязательства накапливаются в цепочках зависимостей. Получивший обещание зависит от тех, кто взял на себя обязательства по его выполнению.

Обещания, обещания

Обязательство — это обещание выполнить некое действие или часть работы на определенном уровне в определенное время. Управление обязательствами — это компонент управления проектами, как я описал во введении к главе 4. Идея не брать на себя обязательств, которые невозможно выполнить, — вопрос личной этики. Она также отражает правильное управление проектами и людьми.



Обязательства накапливаются в цепочках зависимостей, как показано на рис. 5.1. Получивший обещание зависит от тех, кто взял на себя обязательства по его выполнению. Если все честно договариваются о достижимых обязательствах, то смогут с уверенностью положиться друг на друга (Wiegers, 2019d). В противном случае цепочка обязательств превращается в карточный домик. Любые невыполненные обязательства на нижних уровнях подорвут фундамент.

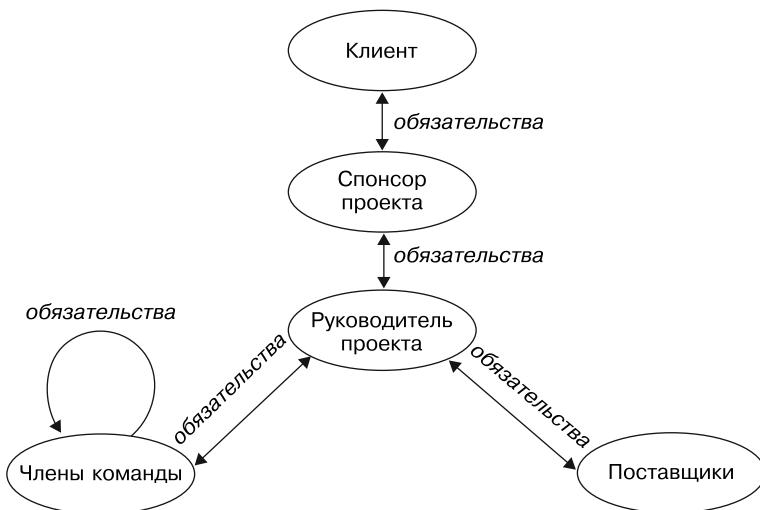


Рис. 5.1. Последовательность обязательств приводит к созданию многоуровневой цепочки зависимостей



Как консультант и писатель я стараюсь обещать меньше, а делать больше. Той же философии я следовал, когда был корпоративным разработчиком программного обеспечения. Возможно, из-за этого я выгляжу чрезмерно осторожным, но это же делает меня надежным. Один из способов не выбиться из графика — заложить в свои обязательства временной резерв на случай непредвиденных обстоятельств, чтобы

учесть неопределенность оценок, изменения в требованиях и другие неожиданности. Наличие такого резерва застрахует меня, если что-то пойдет не так, как я планировал. Но перед собой лично я ставлю более трудные цели, чем обязательства, которые даю другим. Я прикладываю все силы, чтобы достичь своей внутренней цели, зная, что если не добьюсь ее, то все равно не нарушу данного мною внешнего обязательства. Чаще всего я достигаю цели раньше, чем обязуюсь. Такой результат всем нравится.

В жизни случается всякое

Люди берут на себя обязательства из лучших побуждений. Потом что-то происходит, и все меняется. Может появиться новое задание или новая возможность, на которые приходится отвлекаться, или объем работы может превышать предполагаемый. Человек может просто потерять интерес к первоначальному обязательству и позволить ему отойти на второй план, надеясь, что никто не заметит. Но замечают.



Поняв, что по какой-то причине вы не можете выполнить взятое на себя обязательство, как можно скорее сообщите об этом тем, кого это затрагивает, чтобы они могли скорректировать свои планы. Когда кто-то дает мне обещание, я рассчитываю, что он его выполнит. Если мои первоначальные предположения не оправдались или что-то изменилось, то я предлагаю поговорить. Возможно, мы сможем достичь взаимоприемлемого соглашения. Но я также понимаю, что другой человек может не согласиться по разным причинам.



Я сталкиваюсь с ситуацией нарушения обязательств каждый раз, когда работаю над книгой. Я всегда привлекаю нескольких рецензентов-добровольцев, чтобы они помогли мне улучшить мой текст, и хотел бы получить их отзывы по конкретным главам в определенные даты. Ни один рецензент никогда не говорил, что установленные мной дедлайны нереалистичны, и тем не менее каждый раз находятся рецензенты, не присылающие отзывы. В недавней книге 9 из 26 рецензентов (напомню, добровольцев) не прислали ни своих отзывов, ни объяснений. Как будто они дали обет молчания сразу после того, как мы договорились.

Если я говорю, что собираюсь что-то сделать, то либо делаю, либо объясняю, почему не могу или не успеваю это сделать, и приношу извинения. Все мы знаем, что ситуации и приоритеты могут меняться. Если рецензент не может внести свой вклад в проект, то я лишь хотел бы,

чтобы он дал мне знать об этом, а я мог скорректировать свою работу. Это простая вежливость.

Кроме того, я не получил отзыв к одной из моих книг от другого автора, добровольно предложившего внести свой вклад. Книга была издана 25 лет назад, поэтому весьма сомнительно, что я увижу этот отзыв. Но, взяв на себя обязательство, этот человек должен был сообщить мне, что не выполнит его. Большего я не прошу.

Урок 37

Не ждите, что без обучения и освоения передовых практик продуктивность повысится как по волшебству

Некоторые из моих клиентов жаловались мне, что их руководство требует делать больше с меньшими затратами и быть более продуктивными, имея меньше людей. Когда я спрашивал: «А что делает руководство для того, чтобы вы могли делать больше с меньшими затратами?» — то всегда получал один и тот же ответ: «Ничего».

Эти требования не кажутся мне разумными. Руководители, похоже, думают, что у разработчиков программного обеспечения масса свободного времени. Может быть, они сами работают не в полную силу и имеют некий резерв, который смогут использовать, если на них надавить? Щелканье кнутом не может заставить лошадь бежать быстрее, если она уже мчится на пределе своих возможностей. Продолжая метафору, вы должны понять, почему лошадь бежит не так быстро, как вам хотелось бы, а после этого искать возможности для ускорения.

Команды можно мотивировать (или заставить) работать усерднее ради достижения краткосрочной цели, но не бесконечно. Усталые люди совершают больше ошибок, что вынуждает их переделывать сделанную работу, и в конце концов выгорают. Если продолжить оказывать давление слишком долго, то люди уйдут или вернутся к обычному темпу работы, несмотря на разглагольствования руководства. Героические усилия нигде и никогда не были устойчивой стратегией повышения производительности труда.

«Делать больше с меньшими затратами» означает быстрее реализовать больше функций меньшей командой. Если нет возможности нанять столько людей, сколько нужно для выполнения работы в сжатые сроки, то какими еще переменными можно манипулировать? В числе таких

переменных — передовые процессы и практики, лучшие инструменты и талантливые специалисты. Лучше нанять несколько талантливых специалистов, чем большую команду работников со средними возможностями (Davis, 1995). Однако вы не сможете просто поменять свою команду на группу более способных людей. Вам нужно сделать имеющихся у вас специалистов более продуктивными.

В чем проблема?



Если вы стремитесь достичь более высокой производительности, то первый вопрос, который вы должны задать: «Почему наша производительность не так высока, как хотелось бы?» Ответ на этот вопрос требует размышлений и анализа. Начинайте поиск решения с выявления причин проблемы. (См. урок 51 «Остерегайтесь «менеджмента по Businessweek», в котором обсуждается анализ первопричин.») Изучите предыдущие проекты, чтобы увидеть, где могли скрываться дополнительные возможности для повышения эффективности и результативности работы. Только поняв основные причины недостаточной производительности, вы можете приступить к поиску решений. Далее перечислены некоторые вопросы, которые следует рассмотреть.

- Выполняют ли ваши команды работу, которую не должны выполнять?
- Что делают люди в настоящее время из того, что повышает ценность проекта и, возможно, может быть использовано в дальнейшем?
- Чего не делают ваши команды из того, что ускорило бы их работу, если бы они это сделали?
- Что еще вас тормозит?



Некоторые возможные решения

Один из способов повысить производительность — перестать выполнять работу, не несущую ценности проекту, продукту или клиенту. Имеет ли процесс какие-либо ненужные издержки? Но будьте осторожны: шаги, которые не приносят немедленной выгоды, часто окупаются позже, поэтому думайте о последствиях, прежде чем прекращать что-либо делать. На бессмысленные, слишком длинные или перенасыщенные собрания впустую уходит время? Моя подруга работала в технологической компании, помешанной на собраниях. Некоторые из них проводились только для того, чтобы подготовить отчеты для передачи

следующему собранию с другой группой людей всего через час. Моей подруге было трудно продуктивно работать в такой среде.

Вторая стратегия повышения производительности — улучшить качество результатов труда команды. Незапланированные дополнительные переделки убивают продуктивность. Вместо того чтобы перейти к созданию следующего компонента, члены команды вынуждены переделывать выполненную работу и исправлять дефекты. (См. урок 44 «Высокое качество естественным образом ведет к повышению производительности».) Сократить количество дефектов можно путем внедрения дополнительных методов контроля качества. Внедрение проверенных методов, таких как статический анализ и ревью кода, отнимает время, но с лихвой окупается за счет сокращения последующих доработок. Акцент на проектировании, а не на рефакторинге уменьшает технический долг, который команда должна погасить позже. Нужно идти медленно, чтобы двигаться быстро, — урок, который мгновенно усваивает каждый ремесленник. Похожая поговорка, применимая ко многим отраслям, звучит так: «Тише едешь — дальше будешь».



Третий вариант: чтобы устраниТЬ препятствия на пути к продуктивности, важно понять, на что тратится время. Приходится ли людям подолгу ждать других, прежде чем перейти к следующему шагу? Вы можете повысить пропускную способность, ускорив действия, лежащие на критическом пути. Некоторые члены команды теряют продуктивность из-за слишком большого количества решаемых ими задач? Пересмотрите урок 23 «При планировании работ нужно учитывать разногласия», чтобы увидеть, не препятствуют ли прогрессу какие-либо из описанных там источников разногласий.



Четвертый рычаг повышения производительности — расширение возможностей отдельных членов команды. Я всегда предполагаю, что люди работают максимально эффективно для имеющегося уровня знаний и рабочего окружения. И физическое, и культурное рабочее окружение влияет на продуктивность и качество работы разработчика (Glass, 2003; DeMarco, Lister, 2013). Выбор оптимальных процессов и технических приемов может значительно улучшить качество и тем самым повысить продуктивность. Развитие здоровой культуры разработки программного обеспечения, мотивирующей и вознаграждающей желаемое поведение, способствует эффективной работе счастливой команды.

Руководитель, желающий повысить продуктивность, предоставит членам команды кабинеты, которые позволяют работать максимально



эффективно, имеют достаточную площадь, удобны и изолированы от отвлекающих факторов. Однажды я посетил рабочее место своего клиента и ужаснулся: рабочий стол главного контактного лица находился в нескольких дюймах от стола его соседа. Возле стола едва хватило места, чтобы поставить стул для меня. Клиенту пришлось неловко повернуться, чтобы разложить для меня несколько бумаг на столе. В такой стесненной обстановке просто невозможно работать эффективно. Руководитель отдела разработки программного обеспечения сообщил мне, что «в настоящее время нет возможности расширить офисное пространство и большинство его сотрудников работает в таких же ужасных условиях». Это замечание обескуражило меня.

Обучение — мощный рычаг повышения производительности при условии, что люди применяют полученные знания в работе.

Инструменты и обучение

Правильно подобранные инструменты могут повысить производительность. Индустрия программных инструментов имеет долгую историю рекламирования впечатляющего улучшения производительности, которого можно добиться, если купить самую последнюю их версию. В действительности прирост производительности, вызванный использованием одного нового инструмента, редко превышает 35 % (Glass, 2003). Производительность разработчиков ПО с годами растет благодаря накопленным преимуществам нескольких инструментов, новым языкам и методам разработки, повторному использованию ПО с открытым исходным кодом и общих библиотек и другим факторам, но не существует единого универсального инструмента или метода (Brooks, 1995). Не забывайте учитывать затраты на обучение, в ходе которого люди выясняют, как заставить новый инструмент работать эффективно. Ищите инструменты, позволяющие автоматизировать и документировать повторяющиеся задачи, такие как тестирование.

Обучение — мощный рычаг повышения производительности при условии, что люди применяют полученные знания в работе. Перегруженные работой руководители, пытающиеся сделать больше с меньшими затратами, могут не решиться оторвать членов команды от клавиатуры,



чтобы они прошли обучение, да и обучающие курсы стоят дорого. В бытность руководителем небольшой группы разработчиков программного обеспечения я регулярно превышал бюджет, выделенный на обучение, пользуясь поддержкой высшего руководства. У меня были друзья в другом отделе, не имевшем средств на покупку книг. Это казалось мне смешным. Предположим, вы тратите 40 долларов на техническую книгу и читаете ее в основном в свободное время. Сэкономив хотя бы один час на своей работе (в любое время до конца своей жизни) за счет того, что узнали из книги, вы с лихвой окупите затраченные на нее деньги. Инвестиции в обучение окупаются всегда, если новая практика, позволяющая получить более качественный результат за меньшее время, применяется в работе.



Индивидуальные особенности разработчиков

Любому хотелось бы думать, что в его команде собирались самые талантливые специалисты. Всякий хотел бы нанимать только лучших разработчиков. Однако правда в том, что половина всех программистов по своей продуктивности находится ниже медианы. Все эти люди где-то работают, и не у каждого руководителя есть возможность нанять специалистов высшей квалификации. Нелегко количественно оценить продуктивность разработчика ПО, но нередко продуктивность и качество команды зависят от того, кому будет поручена работа.

В многочисленных публикациях по программному обеспечению отмечается десятикратный и более разброс в продуктивности между лучшими и худшими исполнителями (Construx, 2010; McConnell, 2010). И это относится не только к программистам. Люди, исполняющие другие роли в проекте (бизнес-аналитики, тестировщики, владельцы продуктов), тоже могут сильно различаться своей продуктивностью. Однако в недавнем отчете Билла Николса (Bill Nichols) из Института разработки программного обеспечения (Software Engineering Institute) утверждается, что десятикратное увеличение продуктивности программиста — это миф (Nichols, 2020). Данные Николса показывают, что индивидуальные особенности могут увеличить продуктивность разработчиков максимум в два раза в любой конкретной деятельности.

Мой опыт подсказывает, что десятикратная разница — вполне правдоподобное явление. В прошлом моим коллегой был разработчик, работавший качественно и творчески. Но он делал ту же работу, что и я, всего лишь вдвое быстрее. Мне также довелось работать со старшим



разработчиком, который был как минимум в три раза продуктивнее меня. Он имел две степени магистра в области информатики и вычислительной техники, благодаря чему владел навыками эффективного решения сложных задач, с которыми я просто не мог справиться. Кроме того, за свою карьеру он накопил обширную библиотеку повторно используемых компонентов, экономивших ему массу времени. Таким образом, мы втроем охватывали как минимум шестикратный диапазон продуктивности.

Нет никаких сомнений, что квалифицированные и талантливые люди и команды работают более продуктивно. Неудивительно, что, согласно отчету The Standish Group (2015), проекты, укомплектованные «одаренными» командами, использующими методы Agile-разработки, оказались более успешными, чем проекты, реализуемые низкоквалифицированными командами. При этом интересно отметить, что небольшие проекты имели более высокий уровень успеха, чем крупные. Возможно, это объясняется тем, что небольшую команду проще укомплектовать высококвалифицированными специалистами. Не каждый может позволить себе нанимать лучших из лучших. Как отмечает Билл Николс, «найти превосходного программиста сложно, найти способного — нет» (Nichols, 2020).

Если вы не можете собрать звездную команду, то сосредоточьтесь на создании продуктивного окружения, чтобы добиться наилучших результатов от тех, кто у вас уже есть. Развивайте таланты каждого, делитесь передовым местным опытом. Постарайтесь понять, в чем секрет успеха ваших лучших работников (они известны всем), и поощряйте всех, кто учится у них (Bakker, 2020). Технические навыки важны, но не менее ценны общение, сотрудничество, наставничество и отношение к продукту как к общей собственности. Лучшие разработчики, которых я знал, уделяли большое внимание качеству. Они не были догматиками, проявляли интеллектуальную любознательность, обладали обширным опытом, постоянно занимались самообучением и всегда были готовы делиться знаниями.

Если вы должны делать больше с меньшими затратами, то не добьетесь желаемого, отдавая ничем не подкрепленные приказы, оказывая больше давления на свою команду или нанимая таланты только из 90-го перцентиля. Путь к повышению производительности неизбежно состоит из обучения, овладения передовыми практиками и совершенствования процессов.

Урок 38**Люди много говорят о своих правах, но права подразумевают ответственность**

В жизни мы имеем не только права, но и обязанности. Вы имеете право владеть автомобилем, но также должны его зарегистрировать и застраховать. Наряду с правом на покупку недвижимости возникает обязанность платить налог за нее. Как гражданин, вы имеете право голосовать, но также ответственны за правильное заполнение бюллетеня для голосования.

Сочетание прав и обязанностей определяет отношения людей с обществом. В Соединенных Штатах Америки есть Билль о правах — первые десять поправок к Конституции США. Однако нет соответствующего билля об обязанностях. Обязанности граждан закреплены в тысячах законов и постановлений разных юрисдикций. У нас также есть социальные обязательства перед нашими согражданами на планете Земля просто потому, что мы живем здесь вместе.

Люди, работающие над программными проектами, тоже имеют права и обязанности. Наши права в этом случае можно представить как наши ожидания по отношению к другим, а обязанности — как ответственность перед ними. Каждый из нас имеет такие права/обязанности перед своими коллегами, клиентами, руководителями, поставщиками и широкой общественностью. Работая с коллегами, мы должны договариваться с ними примерно так: «Для успеха в этом проекте мне нужно от вас то-то и то-то. А что вам нужно от меня?»



Помимо конкретных обязательств, которые мы берем на себя перед другими, все специалисты по программному обеспечению несут ответственность за соблюдение профессиональной этики. Две основные организации — Ассоциация по вычислительной технике и сообщество IEEE — совместно разработали Кодекс этики и профессиональной практики разработки ПО (ACM, 1999). В нем предполагается обязанность практикующего специалиста действовать в интересах общества, сохранять конфиденциальность, уважать право интеллектуальной собственности, стремиться к высокому качеству и т. д. Все специалисты по программному обеспечению должны ознакомиться с этим руководством и принять личный этический кодекс ответственной разработки ПО (Löwe, 2015).



Между правами и обязанностями специалистов по программному обеспечению существует определенная симметрия. Если члены команды А имеют право рассчитывать на некие услуги или действия членов команды Б, то члены команды А также имеют некие обязанности перед членами команды Б. Составление списка прав должно сопровождаться составлением списка соответствующих обязанностей. Или, если хотите, списки прав для обеих сторон, участвующих в отношениях, должны составляться с подразумеваемыми взаимными обязанностями.

Далее я перечислю несколько взятых из разных источников примеров прав и обязанностей команд, участвующих в разработке программного обеспечения. Но имейте в виду, что в этих источниках приводится множество других примеров (Atwood, 2006; Stack Exchange, без даты). Советую ознакомиться с источником каждого примера, чтобы получить более полное представление о профессиональных правах и обязанностях, применимых в вашем случае. Возможно, вам будет полезно перечислить представления о правах и обязанностях в своем окружении, чтобы помочь заинтересованным сторонам вашего проекта наладить более тесное сотрудничество. Если вас не устраивает формализм записи прав и обязанностей, то подумайте об идее сформулировать их в таком виде: «Для меня резонно ожидать от вас Х, а для вас резонно ожидать от меня Y».

Некоторые права и обязанности клиентов

Как клиент, вы имеете *право* ожидать, что бизнес-аналитик будет использовать информацию о вашем бизнесе исключительно для вашего блага. Вы *обязаны* дать бизнес-аналитикам и разработчикам всю интересующую их информацию о своем бизнесе (Wieggers, Beatty, 2013).

Как клиент, вы имеете *право* получить систему, отвечающую вашим потребностям и ожиданиям в отношении качества. Вы *обязаны* выделить время, необходимое для предоставления и уточнения требований.

Как клиент, вы имеете *право* изменить свои требования. Вы *обязаны* оперативно сообщать об изменениях требований команде разработчиков.

Некоторые права и обязанности разработчиков

Как разработчик, вы имеете *право* на признание и уважение вашей интеллектуальной собственности другими. Вы *обязаны* уважать право

интеллектуальной собственности других лиц, повторно используя плоды их труда только с их разрешения (St. Augustine's College, без даты).

Как разработчик, вы имеете *право* знать приоритет каждого требования. Вы *обязаны* информировать заказчика о влиянии на график новых требований и изменений в приоритетах требований.

Как разработчик, вы имеете *право* оценивать и переоценивать свою работу. Вы *обязаны* делать свои оценки как можно более точными и как можно быстрее корректировать графики работ, приближая их к реальности (Wikic2, 2006; Wikic2, 2008).

Некоторые права и обязанности руководителя или спонсора проекта

Как руководитель или спонсор проекта, вы имеете *право* ожидать от разработчиков высококачественного программного обеспечения. Вы *обязаны* предоставить среду, ресурсы и время, чтобы разработчики могли создавать высококачественное ПО.

Как руководитель или спонсор проекта, вы имеете *право* определять цели проекта и устанавливать графики их реализации. Вы *обязаны* уважать оценки разработчиков и не принуждать их выполнять работы в нереалистичные сроки.

Некоторые права и обязанности члена автономной команды

Как член самоуправляемой, автономной Scrum-команды, вы имеете *право* управлять своими возможностями, контролировать выполнение спринта и устанавливать время завершения работы. Вы *обязаны* определить цель спринта и набор требований для реализации, оценить ежедневный прогресс в достижении этой цели и принять участие в обзоре сделанного, чтобы оценить, как прошел спринт, и создать план улучшений (Ageling, 2020).

Наша главная обязанность — справедливое
и уважительное отношение к коллегам
по профессии.

Опасения перед кризисом

Вы можете думать, что вам полагаются определенные права, присущие какому-то другому сообществу, но порой они не могут или не хотят оправдать ваши ожидания. Всякий раз, сталкиваясь с подобным несоответствием, стороны должны научиться работать вместе конструктивно и без раздражения. Соавторам стоит потратить время, чтобы обсудить взаимные права и обязанности, и, может быть, даже записать их, чтобы избежать недоразумений. Этот диалог представляет собой форму управления ожиданиями, описанную во введении к главе 4.

У меня давно сложилась своя философия выстраивания близких личных отношений, помогающая справиться с беспокойством до того, как оно перерастет в кризис. Эта философия хорошо работает в профессиональной сфере. Большинство людей хотят добра, даже если иногда это не видно с первого взгляда. Достижение общего понимания прав и обязанностей помогает избежать неприятных межличностных кризисов.

Наша главная обязанность — справедливое и уважительное отношение к коллегам по профессии. Мы все должны быть в состоянии сделать это.

Урок 39

Даже небольшие физические расстояния препятствуют общению и совместной работе

В уроке 12 «Выявление требований должно помочь разработчикам услышать голос клиента» я описал свой опыт разработки программного обеспечения для ученого по имени Шон, который сидел рядом со мной. Моя высокая продуктивность была обусловлена быстрой обратной связью, которую я мог получить от него при необходимости. В середине проекта Шон переехал в офис на другом конце здания, и моя продуктивность тут же упала. Теперь я не мог быстро обратиться к Шону, чтобы получить ответ на вопрос, выбрать вариант пользовательского интерфейса или освежить информацию в памяти. Разделение увеличило время решения даже самых простых задач. Осознание, что такое скромное расстояние может снизить продуктивность, стало для меня настоящим открытием.

Барьеры пространства и времени

Выход из прямой видимости затрудняет неформальное общение людей. Как показано на рис. 5.2, работая над проектом Шона, я мог оторвать взгляд от своего монитора и посмотреть, находится ли ученый в своем кабинете. Если его не было рядом, то я мог работать над чем-то еще, что не требовало его участия. Прямая видимость позволяла мне оптимизировать свое время и не ждать ответа, чтобы продолжить работу.

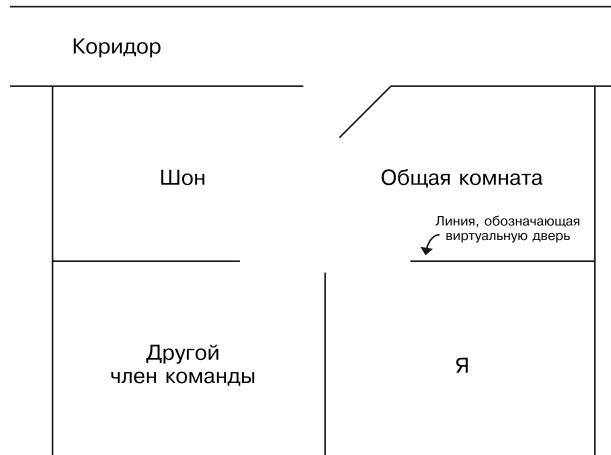


Рис. 5.2. Планировка моего офиса позволяла увидеть, находится ли Шон в своем кабинете

После того как Шон переехал в другой кабинет, я был вынужден звонить ему по телефону, отправлять электронные письма, дожидаться запланированной встречи или идти к нему в его новый кабинет, чтобы узнать, на месте ли он. Проблема усугубляется еще больше, когда сотрудники работают в разных часовых поясах. Эта же проблема наблюдается и в повседневной жизни. У меня есть друг, который часто звонит мне после ужина, но, так как он живет на один часовой пояс восточнее меня, я в момент звонка почти всегда сажусь ужинать. Друг звонит во время, удобное для него, но не для меня. Асинхронное общение возможно во времени и пространстве, но оно медленнее, чем общение в режиме реального времени.

Чем больше часовых поясов разделяют общающихся, тем серьезнее проблема со связью и тем меньше количества рабочих часов, в течение



которых люди могут взаимодействовать. В одном проекте участвовали люди из нескольких стран, находящихся в 12 часовых поясах, то есть практически из половины мира. Какое бы время они ни выбрали для видеоконференций, он было бы неудобным для некоторых членов команды. Группа постоянно меняла время встреч так, чтобы неудобства причинялись всем примерно в равной степени. Эта любезность демонстрировала уважение к каждому и давала важный культурный сигнал, указывающий, что все люди независимо от их места работы одинаково важны для проекта. Никто не должен подвергаться дискrimинации.



Организация офисных помещений таким образом, чтобы внутренняя обстановка содействовала совместной работе и при этом сохранялась приватность, позволяющая сотрудникам быть сосредоточенными на работе, — тонкое искусство. В идеале руководитель может учесть предпочтения каждого члена команды и при этом предоставить хорошие возможности для взаимодействия. Как мы видели в уроке 23 «При планировании работ нужно учитывать разногласия», работники умственного труда наиболее продуктивны, когда входят в состояние потока, глубоко погружаясь в задачу. Поддержание состояния потока требует отсутствия отвлекающих факторов. Таким образом, люди должны быть достаточно близко друг к другу, чтобы легко взаимодействовать, но им также необходимо личное пространство для сосредоточения внимания.



Близость важна, но также нужно учитывать потребность каждого выполнять свою работу, отвлекаясь минимально.



Командам разработчиков выгодно, когда клиент находится рядом, поскольку близость позволяет быстро получать ответы на вопросы. Но подумайте об этом с точки зрения клиента. Шон не был чат-ботом, терпеливо ожидающим моего следующего вопроса. Он был ученым-исследователем, пытающимся выполнить свою работу. Каждый мой вопрос или просьба посмотреть на экран или страницу отчета прерывали состояние потока Шона. Близость важна, но также нужно учитывать потребность каждого выполнять свою работу, отвлекаясь минимально.

Виртуальные команды: максимальное разделение

Все больше и больше IT-специалистов работают удаленно, выбирая такой режим работы самостоятельно или вследствие пандемии COVID-19, что, возможно, делает менее актуальными споры об устройстве офисных помещений. Однако возникновение виртуальных команд порождает новый набор сложностей с организацией культуры сотрудничества. Трудно узнать и понять коллег, которых вы никогда не встречали лично, не говоря уже о том, чтобы выстроить общую культуру ценностей и практик или интегрироваться в нее. Моя коллега Холли, бизнес-аналитик, описала свой недавний опыт:

Пока мне самой не пришлось пройти через это, я не понимала всех трудностей, связанных с созданием сильных виртуальных команд. Недавно я заняла новую должность, на 100 % удаленную. Обязанности не новы для меня, но теперь они совершенно другие. Знакомство с новыми товарищами по команде, изучение терминологии моей новой организации и стиля общения, поиск соответствующей документации, определение технологий, критически важных для поддержки моей роли, — все это знакомо и в то же время совершенно другое. Культура встреч вышла на новый уровень страданий в этом новом мире удаленной работы. Люди больше не ходят на собрания пешком; мы переключаемся с одной встречи на другую щелчком мыши. Куда делись столь необходимые перерывы?



Поиск в Интернете по запросу «сложности виртуальных команд разработчиков программного обеспечения» выдает множество статей, посвященных соответствующим проблемам. Если вы работаете удаленно, то стоит потратить время, чтобы изучить основные сложности и подумать, как лучше всего добиться успеха в мире виртуальных команд.

Дверь, дверь, королевство за дверь!

Закрытая дверь — барьер для общения. В зависимости от того, по какую сторону двери вы находитесь, она может быть преимуществом или неудобством. Мне приходилось работать в офисах с дверями и без них, в офисах с общим открытым пространством и помещениях с огромным количеством кабинок. В помещении, показанном на рис. 5.2 (см. выше), дверь имелась только на выходе в коридор, а наши три отдельных ка-

бинета не имели дверей. Посетитель, пришедший к кому-то одному из нас, отвлекал остальных.



Коллеги по работе часто заходили ко мне с вопросами по программированию. В конце концов, я повесил возле входа в свой кабинет табличку с надписью «Виртуальная дверь открыта» с одной стороны и «Виртуальная дверь закрыта» с другой. Я надеялся, что люди поймут, когда я не хочу, чтобы меня беспокоили. Каким же наивным я был! Большинство посетителей, прочитав надпись «Виртуальная дверь закрыта», находили это довольно забавным и кричали: «Эй, у меня есть вопрос». Так что эта стратегия не имела успеха. Я слышал о людях, которые для того, чтобы сигнализировать, что они сейчас заняты и не хотят, чтобы их отвлекали, использовали оранжевый конус безопасности дорожного движения или бархатную веревку, подобную тем, которые можно увидеть в дверных проемах в кинотеатрах.

В конце концов, у меня появился кабинет с настоящей дверью, что позволило мне выполнять больше работы. В некоторых компаниях эксперты, оказывающие консультативные услуги, устанавливают рабочие часы, в течение которых к ним можно прийти для обсуждения назревших вопросов (Winters et al., 2020). Это может быть не очень удобно для человека, имеющего вопрос, зато эксперт реже отвлекается.



Несколько лет спустя я переехал в другое здание с совершенно иной культурой и средой и оказался в стране кабинок. У всех были столы, окруженные полутораметровыми перегородками, за исключением руководителей, которые (разумеется) имели личные кабинеты. Моя кабинка находилась в одном из самых шумных мест в огромном здании длиной 400 метров, рядом с главным коридором, тянущимся через весь зал, туалетами и комнатами для встреч и тренировок. В метре от моего стола стояла кофеварка, где студенты собирались и болтали во время перерывов между занятиями. У моего местоположения был единственный плюс — после празднований на столе с кофеваркой оставались остатки торты. Я не пью кофе, но люблю торты.

Меня легко отвлекают посторонние звуки, которые другие даже не замечают. Мне потребовалось два месяца, чтобы научиться не реагировать на отвлекающие факторы в этом открытом пространстве и сосредоточиваться на своей работе. Конечно, подобные открытые пространства облегчают общение с окружающими людьми, но я испытывал сильный стресс, пока не адаптировался к шумной среде.

Не существует идеального решения для организации рабочих пространств, которые могли бы удовлетворить противоречивые потреб-

ности в приватности, свободе от отвлекающих факторов, близости к коллегам в целях быстрого взаимодействия и общем пространстве для совещаний по проекту. Географическое разделение вызывает проблемы, а культурные различия значительно усложняют эффективное сотрудничество. Инструменты для совместной работы помогают сотрудничать людям, разделенным большими расстояниями, но не могут полностью заменить личное взаимодействие в реальном времени. По мере возможности руководители должны позволять членам команды самим организовывать их рабочие места, чтобы наилучшим образом сбалансировать эти противоречивые цели (DeMarco, Lister, 2013). А если это невозможно, то я рекомендую наушники или беруши.

Урок 40

Неформальные подходы, используемые небольшими сплоченными командами, плохо масштабируются

Билл Гейтс (Bill Gates) и Пол Аллен (Paul Allen), Стив Джобс (Steve Jobs) и Стив Возняк (Steve Wozniak), Билл Хьюлетт (Bill Hewlett) и Дэвид Паккард (David Packard). Я полагаю, что ни у кого из них не было письменных спецификаций или инструкций для их ранних проектов. Слияние разумов между парами суперталантливых волшебников, работающих бок о бок, устраняет необходимость в большей части документации. Но такое слияние трудно масштабировать во времени и пространстве, когда этих разумов сотни и при этом они думают на разных языках и погружены в разные культуры.

На рис. 5.3 показана качественная шкала сложности проекта. В крайних точках находятся пара гениев, работающих в гараже, и тысячи людей, строящих Международную космическую станцию (самый большой проект, который я смог придумать). По мере роста организаций гении берутся за более крупные и сложные проекты, выходят из гаража, нанимают других людей и занимаются проектами вместе с одним-двумя десятками технических специалистов, сосредоточенных в одном месте. Если вы создаете устройство со встроенным программным обеспечением, то в вашу команду будут входить инженеры-механики и инженеры-электрики. В какой-то момент вы можете приобрести другую компанию в другом городе или как-то иначе начать сотрудничать с людьми издалека, запустив проект, занимающийся распределенной разработкой. Компании, создающие огромные и сложные продукты, например самолеты, могут иметь в своем составе инженеров-програм-

мистов, инженеров по оборудованию и множество других специалистов, работающих над десятками подпроектов в разных странах. Большие задачи требуют соответствующих подходов.

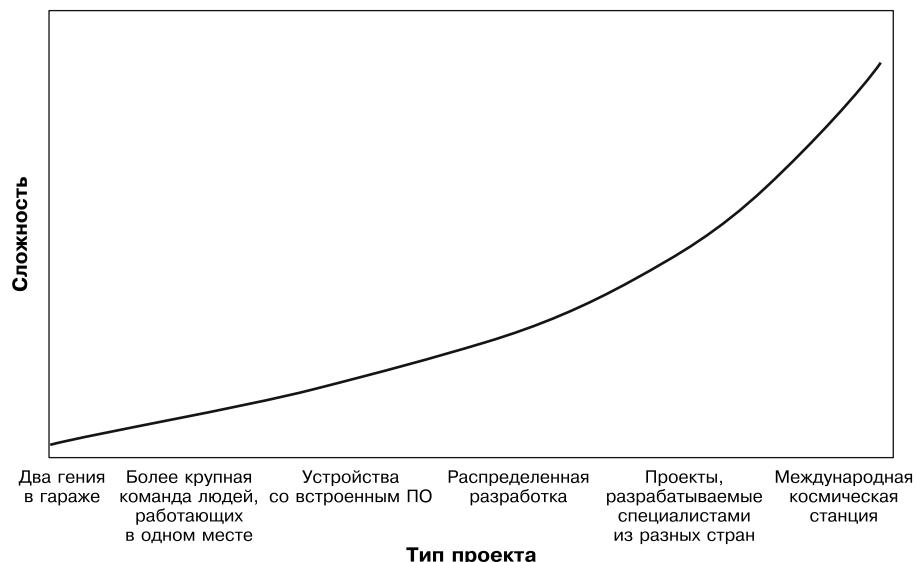


Рис. 5.3. По мере увеличения размеров и сложности проектов организации должны масштабировать свои процессы и методы

Процессы и инструменты

По мере роста команд, проектов и организаций возникает потребность в совершенствовании процессов. Люди могут осознать необходимость в этом, только столкнувшись с проблемой. Может так случиться, что две команды выполнят одну и ту же работу или какая-то задача будет пропущена, поскольку все думали, что ее сделает кто-то другой. Эти проблемы требуют более четкого планирования, координации и коммуникации между членами команды и группами. По мере вовлечения в работу все большего количества людей, особенно если они находятся в разных местах, появляются все новые источники разногласия и задержек.

Чем дальше друг от друга находятся члены команды, тем больше внимания нужно уделять организации процесса.

Более крупные проекты требуют письменного оформления большего объема информации, чтобы другие участники знали, что происходит и что ждет впереди. Сохранение требований, решений, бизнес-правил, оценок и метрик в групповом хранилище начинает приносить все большую пользу. Чем дальше друг от друга находятся члены команды, тем больше внимания нужно уделять организации процесса и тем больше люди вынуждены полагаться на инструменты удаленной совместной работы.

Чтобы работать совместно и упростить обмен информацией, командам, расположенным в разных местах, нужны инструменты управления задачами, моделирования, тестирования, непрерывной интеграции и особенно управления кодом (Senycia, 2020). Программное обеспечение с открытым исходным кодом — яркий пример распределенной разработки ПО. Участники проекта с открытым исходным кодом могут жить в разных точках мира. Люди, участвующие в постоянно развивающихся проектах, должны следовать некоторым процессам, чтобы гарантировать сочетаемость всех частей. Как отмечает Эндрю Леонард (Andrew Leonard), «ни одно нововведение не было столь же важным для разработки программного обеспечения распределенными командами, как появление систем управления версиями» (Leonard, 2020).

Необходимость специализации

Программировать я начал еще в колледже. В ту пору я писал программы для личного пользования и для развлечения. Затем я начал писать программы для других людей и создал несколько небольших коммерческих приложений. Позже я объединил усилия с другими разработчиками и занялся более крупными проектами. Работая над проектом в одиночку, я поочередно выполнял все роли участников проекта. В каждый конкретный момент я мог действовать как бизнес-аналитик, дизайнер, программист, тестировщик, составитель документации, руководитель проекта или специалист по сопровождению.

Нереально ожидать, что каждый член команды будет в полной мере владеть всеми этими ролями. По мере роста проектов и команд специализация некоторых навыков становится естественной и полезной. В какой-то момент даже Билл Гейтс и Пол Аллен решили, что им нужно нанять людей, обладающих навыками, отличными от программирования, чтобы обеспечить рост своей молодой компании: технического писателя, математика, руководителя проекта и т. д. (Weinberger, 2019). Я слышал, у них это сработало.



Коммуникационные конфликты



Как отмечалось в уроке 14, по мере увеличения группы количество коммуникационных каналов растет экспоненциально. Участникам проекта нужны механизмы, позволяющие быстро и точно обмениваться информацией. Предпочтения в общении тоже могут различаться. Люди инстинктивно выбирают способы общения, наиболее удобные для них самих, но их предпочтения не всегда совпадают с предпочтениями другой стороны.



Однажды я проводил ретроспективный обзор проекта, в котором участвовали команды веб-разработчиков и специалистов по изучению влияния человеческого фактора. В ходе обзора член команды по изучению влияния человеческого фактора сказала, что не чувствовала вовлеченности в общую работу и не всегда знала, что происходит. Удивленный член команды разработчиков запротестовал: «Я включил вас во все наши рассылки». Разбирая этот момент, мы выяснили, что эти две команды имели очень разные предпочтения в общении и выбирали разные способы получения важных сообщений.

В данном случае команды располагались недалеко друг от друга, но имели явное разделение в стилях общения. Переходя к проектам с участием нескольких команд, важно заранее заложить основу для общения, используя те способы, которые будут эффективно работать для всех. Кроме того, команды могут по-разному принимать решения. Чтобы избежать обид и ускорить принятие решений, участникам следует обсудить, как будут приниматься совместные решения, — до того как все столкнутся с первой серьезной проблемой.



Успешные организации растут и берутся за все более сложные задачи. Им не следует ожидать, что методы, эффективные для нескольких человек в гараже, подойдут и для больших распределенных команд. Они могут избежать некоторых трудностей, заранее предусмотрев процессы и инструменты, необходимые для интеграции различных команд. Гораздо проще организовать максимальный объем инфраструктуры заранее, чем переложить ее строительство на участников проекта в самый разгар работы. Если участники будут иметь возможность общаться и понимать коллег, работающих удаленно, на ранних этапах проекта, то это поможет избежать культурных конфликтов. Потратив время на то, чтобы заложить эти основы, можно предотвратить много неприятностей в будущем.

Урок 41

Не стоит недооценивать сложность изменения культуры организации по мере перехода к новым методам работы

В уроке 36 «Как бы сильно на вас ни давили, не берите на себя обязательства, которые не сможете выполнить» я упомянул, что когда-то руководил программой совершенствования процессов в крупном корпоративном подразделении. Старший менеджер этого подразделения, находящийся на три ступени выше меня в организационной иерархии, с энтузиазмом поддержал нашу работу. Но Мартин не в полной мере оценил все, что было связано с переводом нескольких сотен программистов и системных инженеров в новую парадигму.



Мартин, по всей видимости, думал, что мы просто опишем новые процедуры разработки программного обеспечения — и все получится само собой. Он утвердил новую политику управления, которую мы определили как часть изменения структуры, и на этом все. Он не объявлял и не объяснял политики, не указывал, как они должны применяться, не определял пути перехода к новым методам работы и никого не привлекал к ответственности за несоблюдение правил. Как результат, политика в целом игнорировалась, а новые процедуры изначально имели ограниченное влияние.



Этот опыт показал, что определить новый процесс проще, чем привить новую культуру. Успех инициативы по совершенствованию требует и того и другого. Вы не можете просто отправить всех на тренинг и дать толстый буклет (или тонкое руководство в зависимости от того, какова ваша цель) с описанием новых процедур, а затем ожидать, что люди начнут эффективно работать по-новому. Такой подход только сбивает с толку, вызывает напряжение и подрывает доверие к руководству даже в развитых культурах.

Ценности, модели поведения и практика

Как я уже говорил, здоровая культура разработки программного обеспечения подразумевает общие ценности, модели поведения и технические приемы. Вовлечение членов команды в процесс совершенствования повышает их заинтересованность — умелые лидеры могут заставить людей поверить в то, что эти идеи принадлежат им самим. Члены орга-

низации, в которой внедряется инициатива по совершенствованию, должны знать гораздо больше, чем просто список новых методов, которым они должны следовать. Лидеры изменений должны объяснить:

- почему изменения необходимы;
- какие проблемы они решат;
- каких результатов организация надеется достичь;
- как изменения повлияют на организационную структуру;
- любые новые роли, должности или обязанности;
- ожидания членов команды как отдельных лиц;
- сроки реализации;
- ответственность за конструктивный вклад в инициативу.

Успешное изменение культуры требует того, чтобы лидеры убедили всех, кого это касается, принять преимущества как цели, так и подхода. Убеждать легче, когда все хорошо осведомлены о недостатках. Одна моя коллега как-то сказала мне, что ни один из программных проектов, в которых она работала, не добился успеха. Это побудило ее возглавить деятельность по совершенствованию процессов, чтобы улучшить результаты проекта и добиться успеха компании в бизнесе.

Меняться тяжело. Не все хотят этого, и вы не сможете развивать культуру, если люди активно сопротивляются, не желая покидать свою зону комфорта. Однако тем, кто больше всего обеспокоен переменами, нельзя позволять сдерживать всю инициативу. Страйтесь найти разумный баланс между навязыванием нового образа жизни в приказном порядке, с одной стороны, и попытками сделать всех счастливыми — с другой.

Мартин сказал, что поддерживает инициативу по внедрению изменений в своем подразделении, но я не уверен, что он действительно так думал. Мне слово «поддержка» кажется расплывчатым и неоднозначным. Когда кто-то говорит: «Я поддерживаю это», — он часто имеет в виду: «Я не против, если это произойдет» или «Звучит неплохо». Понятие поддержки как терпимости или одобрения сильно отличается от приверженности. Преданные делу лидеры действуют и подают пример. Они ставят конкретные цели, предоставляют ресурсы и демонстрируют поведение, соответствующее намеченному результату. Они рискуют и несут ответственность за результаты своей организации. Руководители, стремящиеся к устойчивым изменениям, умеют формировать их так, чтобы они вызывали положительные эмоции и стимулировали действия (Walker, Soule, 2017). Общие фразы, предписания и директивы не помогут.



Эффективные лидеры изменений знают: направлять членов команды к новым ценностям и поведению означает, что их требуется ежедневно убеждать. У людей в организации есть три варианта: они могут помочь возглавить инициативу, последовать за ней и внести свой посильный вклад или уйти с дороги. Если масштабы или характер изменений заставляют кого-то чувствовать себя слишком некомфортно, то эти люди, скорее всего, уйдут, но это лучший выход для всех заинтересованных сторон. Я слишком часто видел, как люди, недовольные новым направлением своей организации, применяли пассивно-агрессивный подход. На словах они поддерживали развитие, но игнорировали или подрывали его при каждой возможности.

Agile-разработка и изменение культуры

Переход на Agile — серьезное изменение в работе организации, занимающейся разработкой программного обеспечения. Выше я говорил, что нельзя купить готовую культуру и установить ее, как новый инструмент или техническую практику, но это не относится к Agile. Переход, скажем, на Scrum предполагает освоение заранее подготовленной культуры с набором ценностей, моделей поведения и технических приемов. Многие аспекты культуры должны измениться при переходе на Agile, в том числе (Hatch 2019):

- организационная структура и состав команды;
- терминология;
- ценности и принципы;
- новые роли в проекте, такие как скрам-мастер и владелец продукта;
- методы составления графиков и планов;
- характер сотрудничества и общения;
- оценка прогресса;
- ответственность за результаты.

Этих изменений много, чтобы внедрить их все сразу. Традиционные стратегии совершенствования процессов разработки программного обеспечения предполагают постепенность: новые процедуры и практики добавляются в проектную деятельность небольшими порциями. Переход к методикам Agile-разработки, напротив, предполагает более резкое изменение мышления и практик. Простого обучения недостаточно, чтобы добиться приверженности новым ценностям, изменить индивидуальное поведение и принять новые методы. Людям

нужно время, чтобы забыть прошлые навыки и освоиться с новыми способами работы.



Я знаю две компании, которые провели масштабное обучение сотен своих специалистов методам Agile-разработки. Одна компания сразу же приступила к работе над масштабным стратегическим проектом, в котором участвовало более 150 человек, живущих в разных местах. После нескольких лет и значительных финансовых вливаний проект был приостановлен, поскольку так и не создал ничего пригодного для использования.



Вторая компания внедрила Agile на уровне предприятия и выбрала эту методологию для всех проектов независимо от того, насколько хорошо это соответствовало ситуации в проекте. Мой друг, работавший в этой компании, объяснил мне:

Это было глобальное преобразование. Оно повлияло на все наши методы разработки. Новые процедуры, требования аудита и агрессивный график преобразования вызвал проблему там, где для плавного перехода было выделено недостаточно времени. Переход на Agile был представлен как окончательное решение. Высшее руководство не ограничивалось решением конкретных проблем внутри компании и не пыталось уговорить нас на это. В организации выявилось множество противников. Многие разработчики были очень недовольны тем, что их нанимали для водопадной разработки, а заставляют принять методику Agile.

В идеальном мире все реагируют на глубокие изменения, размышляя: «Что это даст нам?» Но в реальности большинство людей думает: «Почему вы заставляете меня это делать?» Кто-то нашел ответ на этот вопрос, кто-то нет.

Если бы руководство уделяло больше внимания существенному культурному влиянию таких радикальных одномоментных преобразований, то изменения были бы восприняты с большей готовностью и результаты были бы более удовлетворительными. Время покажет, насколько успешными могут быть такие попытки крупномасштабных преобразований.

Интернализация



Для достижения желаемых бизнес-результатов инициатива по совершенствованию процессов и культуры преследует две цели. Одна из них — *институционализация*. Под ней понимается принятие в орга-

низации новых методов, моделей поведения и ценностей как части стандартного набора подходов к работе над проектами. Вторая цель — **интернализация**. Под ней подразумевается укоренение новых способов работы в умах отдельных членов команды.

Когда члены команды интернализируют (принимают) новые методы и модели поведения, они применяют их не потому, что этого требует руководитель, должностная инструкция или инструктор по Agile-разработке, а потому, что это лучший из известных им способов выполнения работы. Интернализация занимает больше времени, чем институционализация, но также является более явным признаком успешного изменения культуры. Освоив лучший способ работы, вы автоматически продолжите использовать его.

Фундаментальный переход от устоявшихся методов работы к чему-то совершенно иному требует изменения множества аспектов: организационных, управленческих, лидерства в проектах, практики вознаграждения, технических приемов, индивидуальных привычек и поведения команды. Все это невозможно купить или заказать. Высшее руководство должно быть привержено изменениям и убедительно доказывать их необходимость всем, кого они касаются (Agile Alliance et al., 2021). Культурный переход — более сложная задача, чем установка новых приемов и методов; не решив ее, вы почти наверняка потерпите неудачу.

Приняв новые методы и модели поведения, члены команды применяют их просто потому, что это лучший из известных им способов выполнения работы.

Урок 42

Никакие инженерные или управленческие приемы не дадут эффекта, если вы имеете дело с неразумными людьми

Иногда можно столкнуться с тем, кто действует неразумно. Это может быть коллега по команде, руководитель, клиент или кто-то еще. Мы встречали таких людей, обсуждая предыдущие уроки:

- начальник, ожидающий, что сотрудник будет непрерывно работать над проектом по восемь часов пять дней в неделю (урок 23);

- заказчик, настаивающий на том, что может выдвигать требования от имени группы пользователей, к которой он не принадлежит (урок 33);
- коллега, умалчивающий о своих знаниях вместо того, чтобы делиться ими с соратниками (урок 35);
- старший менеджер, надеющийся изменить культуру в крупной организации, утвердив ряд новых инструкций (урок 41).

Человек, действующий неразумно, — это не техническая, а людская проблема. Решить ее можно несколькими способами. Один из них — неуклонно отстаивать свою позицию в спорах. Другой — поддаваться давлению со стороны неразумного человека и делать все, что он говорит, даже если вы считаете это неправильным. Третий вариант — делать вид, что подчиняешься, но поступать по-другому или вообще ничего не делать.



Лучшая стратегия — понять, что заставляет такого человека действовать неразумно, а затем подумать, как реагировать. Эти люди могут защищать укоренившиеся привычки, а не свои законные интересы. Часто люди, кажущиеся неразумными, просто недостаточно информированы. Незнакомые приемы, жargon и самоуверенность разработчиков программного обеспечения могут заинтересовать клиентов, у которых мало опыта работы с командами разработчиков и не хватает технических знаний. Люди, имеющие отрицательный прошлый опыт проектирования, могут с подозрением относиться к нововведениям.



Однажды, работая ведущим бизнес-аналитиком в одном проекте, я общался с клиентом, который уже дважды обжегся на неудачных проектах. Он опасался работать еще с одной командой разработчиков программного обеспечения. Узнав его историю, я понял, почему он с самого начала не доверял мне — человеку, которого только что встретил. Мы преодолели его недоверие, проявляя терпение и подробно объясняя наши действия, и проект был весьма успешным.

Попробуйте поделиться знаниями

Если необоснованная реакция вызвана недостатком знаний, то попробуйте передать человеку часть своих знаний. Он должен научиться понимать терминологию, которую вы используете, методы, которые вы применяете, почему они способствуют успеху проекта и что произойдет, если отказаться от них. Люди, с которыми вы работаете,

должны понимать, чего вы от них ждете и чего они могут ожидать от вас. Если бизнес-аналитик начинает разговор с новым клиентом со слов: «Давайте поговорим о ваших пользовательских историях», — то это пугает и вызывает неприятие. Клиент понятия не имеет, что такое «пользовательские истории» и какова их роль в процессе. Иногда, чтобы разрешить ситуацию, достаточно поделиться небольшим объемом информации.

Я столкнулся с подобной нехваткой знаний вскоре после того, как занял пост руководителя группы, поддерживавшей программное обеспечение в исследовательских лабораториях крупной компании. Я рассказал о работе моей группы на ежегодном собрании старших научных руководителей и заявил, что отныне наша группа не будет разрабатывать ПО без письменных спецификаций. Старший менеджер по имени Скотт запротестовал: «Постойте-ка, у меня есть подозрение, что мы по-разному понимаем слово “спецификация”». На что я ответил: «Тогда мы можем воспользоваться моим пониманием». Чтобы эти слова не показались слишком легкомысленными (каковыми они и были, конечно же), я объяснил, что имел в виду под спецификациями. Поняв, что я не прошу ничего сверхъестественного, Скотт согласился с тем, что мое заявление было справедливым.



Кто здесь вне очереди?

Мой разговор со Скоттом проиллюстрировал важный момент: убедитесь, что *вы* сами ведете себя разумно. Бизнес-аналитики, руководители проектов или разработчики могут показаться неразумными, если не соглашаются делать все, что от них требуют. Бизнес-аналитик, который просит, чтобы пользователь или руководитель уделяли время работе с командой разработчиков программного обеспечения, может показаться неразумным тому, кто не ожидал, что от него потребуется такое активное участие в проекте.



Иногда люди думают, что они знают больше, чем на самом деле, и оказываются давление, требуя использовать неверный подход. Моя подруга Андреа, очень опытный разработчик и проектировщик баз данных, заключила контракт с компанией, для которой она уже построила несколько успешных систем. Старший разработчик клиента настаивал на том, чтобы база для новой системы использовала созданную им модель данных. Андреа указала на серьезные недостатки этой модели, но разработчик клиента пригрозил: «Или вы используете мою модель



данных, или я отзываю проект». Андреа неохотно согласилась. Конечно, у продукта было много проблем, решение которых потребовало больших усилий. Когда Андреа правильно заметила, что эти проблемы возникают из-за несовершенной модели данных, разработчик клиента обвинил ее в том, что она испортила реализацию его великолепной модели. Поведение этого клиента было неразумным по любым меркам, и от последствий пострадали все.

Чтобы никто не выглядел неразумным, попробуйте понять цели, приоритеты, движущие силы, страхи и ограничения друг друга.

В пользу гибкости



Категоричность — еще одна черта, кажущаяся неразумной. Однажды, когда я работал в группе совершенствования процессов разработки программного обеспечения на основе модели зрелости процессов разработки (Capability Maturity Model, CMM), я поспорил с менее опытной коллегой Сильвией. Она утверждала: «CMM — это модель. Мы должны поступать в соответствии с ее требованиями!» Это показалось мне чрезмерно ограничивающим и неправильным толкованием. Я ответил: «CMM — это модель. Мы должны адаптировать ее к нашей ситуации!» Сильвия была категоричной и неуступчивой; она думала, что я нарушаю цель CMM, не следя в точности ее требованиям. В процессе последующего обсуждения наша группа пришла к общему пониманию, как использовать CMM в нашей среде. Тем не менее меня поразила большая разница в том, как мы с Сильвией интерпретировали понятие «модель».



Чтобы никто не выглядел неразумным, попробуйте понять цели, приоритеты, движущие силы, страхи и ограничения друг друга. Выясните, какие действия и результаты приносят плоды в обоих ваших мирах, а какие — чреваты проблемами. Заключите соглашение, чтобы все стороны знали, чего ожидать друг от друга. Мы будем лучше ладить, ценя и уважая точку зрения друг друга.

СЛЕДУЮЩИЕ ШАГИ: КУЛЬТУРА И КОМАНДНАЯ РАБОТА



1. Определите, какие уроки, описанные в этой главе, имеют отношение к вашему опыту с точки зрения культуры и командной работы.
2. Можете ли вы, опираясь на свой опыт, вспомнить какие-либо другие связанные с культурой и командной работой уроки, которыми стоит поделиться с коллегами?
3. Перечислите описанные в этой главе методы, способные помочь в решении связанных с культурой и командной работой проблем, которые мы определили во врезке «Первые шаги» в начале главы. Как каждый метод может улучшить совместную работу членов команды? Как вы могли бы исправить действия, разрушающие культуру, и усилить влияние действий, направленных на ее укрепление?
4. Как бы вы определили, приносит ли желаемые результаты каждый метод, озвученный на шаге 3? Насколько ценные для вас эти результаты?
5. Определите любые препятствия, которые могут затруднить применение методов, перечисленных на шаге 3. Как бы вы справились с ними? Заручились бы поддержкой коллег, готовых помочь вам в реализации этих методов?
6. Подумайте, какие положительные аспекты сотрудничества имеются в ваших командах и как распространить этот опыт на другие проектные команды. Как бы вы прививали культурные обычай и взгляды, чтобы сохранить их с течением времени и помочь будущим командам добиться успеха?

Глава 6

Качество

ВВЕДЕНИЕ



Однажды я написал идеальную программу на языке ассемблер, представляемое? Она была небольшой (обучающая игра по химии), но в ней не было ни одного дефекта, и она делала правильно все, что должна была делать. Но мне доводилось писать намного больше кода, который, несмотря на все мои усилия, содержал ошибки, и мне пришлось исправлять их позже. Качественное ПО важно для меня и для всех, кто создает или использует программные системы. Мы все должны стремиться качественно выполнять свою работу, но что такое *качество*?

Определения качества

Люди веками пытались дать определение понятию «качество», но безуспешно. Я видел много попыток сделать это, но так и не встретил исчерпывающего и сжатого определения, применимого к программному обеспечению. Американское общество качества (American Society for Quality, 2021a) признает это в первой части своего определения: «Субъективный термин, который определяется каждым человеком или отраслью по-своему». Это верно, хотя и почти бесполезно. Разные наблюдатели всегда будут иметь разные представления о том, какие черты данного продукта или услуги определяют качество или его отсутствие. Ниже перечислены несколько примеров определений качества; каждое имеет свои достоинства, но ни одно из них не совершенно.

- «1) Характеристики продукта или услуги, связанные с их способностью удовлетворять заявленные или подразумеваемые потребности; 2) продукт или услуга без недостатков» — определение Американского общества качества (American Society for Quality, 2021a).
- «Степень, в которой программный продукт удовлетворяет заявленным и подразумеваемым потребностям при использовании в определенных условиях» — определение Международной организации по стандартизации и Международной электротехнической комиссии (ISO/IEC, 2011).
- «Пригодность для использования, в том смысле, что продукт должен удовлетворять реальные потребности клиента и вызывать у него чувство удовлетворения» — определение пионера качества Джозефа М. Джурана (Joseph M. Juran) (American Society for Quality, 2021b).
- «Соответствие требованиям» — определение Филипа Б. Кросби (Philip B. Crosby) (Crosby, 1979).
- «Отсутствие дефектов» — также определение Кросби (Crosby, 1979).
- «Ценность для некоторых людей» — определение Джеральда Вайнберга (Gerald Weinberg) (Weinberg, 2012).

Наличие столь разных определений позволяет сделать два вывода: качество имеет множество аспектов и понятие качества зависит от ситуации. Вероятно, все мы можем согласиться с тем, что в контексте программного обеспечения понятие качества описывает, насколько хорошо продукт выполняет все, что от него требуется, и дать более строгое определение, вероятно, не получится. И все же каждая проектная группа должна изучить, что ее клиенты понимают под качеством, как его оценить и достичь, а затем в ясной форме передать эти знания всем участникам проекта (Davis, 1995).



В идеальном мире каждый проект должен предоставлять продукт, содержащий все функции, которые когда-либо потребуются любому пользователю, не имеющий дефектов и идеально удобный, произведенный в рекордно короткие сроки и с минимальными затратами. Но это фантазия; ожидания в отношении качества должны быть реалистичными. Лица, принимающие решения по каждому проекту, должны определить, какие аспекты наиболее важны для успеха проекта и на какие компромиссы они могут пойти, преследуя свои бизнес-цели.

Планирование качества

Недостатки качества программного обеспечения оказывают ошеломляющее совокупное влияние на организации, страны и планету. В ходе подробного анализа общие потери из-за низкого качества программного обеспечения в США в 2018 году были оценены в *2,26 трлн долларов* без учета технического долга и в *2,84 трлн долларов* вместе с ним (Krasner, 2018). Только представьте, какие экономические выгоды на каждом уровне может дать более качественное ПО!

Как показано на рис. 6.1, в классическом (железном) треугольнике ограничений управления проектами качество обычно не указывается явно как регулируемый параметр наряду с масштабом, стоимостью и временем. Его отсутствие можно интерпретировать как непреложность ожидания высокого качества или, что более вероятно, как получение качества определенного уровня, которого команда может достичь в рамках ограничений, налагаемых другими параметрами. То есть качество является зависимой, а не независимой переменной (Nagappan, 2020b).



Рис. 6.1. Классический треугольник управления проектами не показывает качество в явном виде

Тем не менее команды разработчиков и руководители иногда решают пойти на компромисс в отношении качества, чтобы уложиться в срок, или включить более богатый, хотя и несовершенный, набор функций, делающих их продукт более привлекательным для клиентов. Вот почему моя расширенная пятимерная модель из урока 31 в главе 4 содержит качество как явный параметр проекта, наряду с масштабом, планом, персоналом и бюджетом. Люди, принимающие решения о выпуске, могут смириться с существованием неких известных дефектов, если, по их мнению, эти дефекты не будут оказывать большого влияния на клиентов или бизнес. Однако пользователи могут не согласиться с этим компромиссом (Weinberg, 2012). Если любимая функция пользователя будет иметь дефект, то пользователь, скорее всего, сочтет весь продукт проблемным и расскажет об этом всем, кого знает.





Программная система необязательно должна иметь множество проблем, чтобы произвести впечатление низкокачественной. У меня есть хобби: писать и записывать песни. Я купил приложение, которое могу использовать для записи музыкальных партитур. Написание музыки — сложная задача; соответственно, и приложение, которое я использую, очень сложное. У него есть некоторые недостатки, в частности в нем неудобно вводить ноты. Хуже того, я столкнулся с многочисленными сбоями, пытаясь создать или изменить партитуру. Очень неприятно, когда вводишь самую обычную музыку, а программа сходит с ума, отображая что-то несусветное. Это приложение имеет чрезмерно богатый набор функций, многие из которых я никогда не буду использовать, а некоторые вообще работают плохо, и я предпочел бы иметь меньше функций, но чтобы все они работали правильно и удовлетворяли потребности большинства пользователей.

Команды разработчиков выиграют от создания плана управления качеством в начале проекта. В плане должны быть определены реалистичные ожидания в отношении качества продукта, включая классификацию серьезности дефектов (серьезные, умеренные, незначительные, косметические). Это поможет создать у всех участников проекта согласованное представление о понятии качества. Определение общей терминологии и ожиданий в отношении различных видов тестирования дополнительно поможет сблизить заинтересованные стороны, позволяя достичь общей цели — создания высококачественного решения.

Несколько взглядов на качество

Качество программного обеспечения состоит из множества слагаемых. Это больше, чем простое соответствие заданным требованиям (при условии, что эти требования верны), и больше, чем отсутствие дефектов. Говоря о качестве данного продукта, необходимо учитывать многочисленные характеристики: функциональные возможности, эстетику, производительность, надежность, удобство использования, стоимость, своевременность доставки и т. д. (Juran, 2019).



Как мы видели в уроке 20 «Невозможно оптимизировать все желаемые атрибуты качества», команды, занимающиеся разработкой программного обеспечения, должны изучить широкий набор требований к атрибутам качества. Поскольку невозможно создать продукт, идеальный по всем параметрам, часто приходится идти на компромиссы в отношении

качества. Разработчики должны принимать решения, отдавая предпочтение определенным атрибутам. Атрибуты качества должны быть точно определены и расставлены по приоритетам, чтобы те, кто принимает решения, могли сделать правильный выбор.

Кроме того, заинтересованные стороны могут по-разному воспринимать качество. Разработчик может считать, что высококачественный код должен быть написан элегантно и выполняться эффективно и правильно. Однако специалист по сопровождению может высоко ценить более простой и понятный код. Пользователь может рассматривать высококачественный код (в той мере, в какой он вообще думает о коде) как все, что необходимо, чтобы он мог использовать продукт легко и без сбоев. В этом примере разработчик и специалист по сопровождению сосредоточены на внутреннем качестве продукта, а пользователю интереснее его внешнее качество.

Последовательное обеспечение качества



Повышение качества, не считая косметических и эстетических улучшений, — сложная задача. Нельзя просто написать несколько целей доступности в пользовательской истории и добавить ее в требования продукта для реализации в одной из будущих итераций разработки. Качество должно повышаться с самого начала последовательно и неуклонно путем применения соответствующих процессов, постановки целей и формирования отношения членов вашей команды. Некоторые атрибуты качества накладывают ограничения, влияющие на весь процесс разработки, а не только на определенные элементы функциональности (Scaled Agile, 2021c). Удовлетворение конкретных критериев качества влечет архитектурные последствия, которые команда должна учитывать с самого начала проекта.

Повышение качества, не считая косметических и эстетических улучшений, — сложная задача.

Трудно обеспечить высокое качество продукта, построенного на шатком фундаменте. Пытаясь уложиться в срок за счет экономии времени на реализации некоторых возможностей, разработчики будут накапливать технический долг, что еще больше усложнит модификацию и расширение кодовой базы. *Технический долг* определено относит-

ся к накопленным недостаткам качества реализованного программного обеспечения. Он может быть обусловлен множеством причин и сам являться основной причиной срыва сроков сдачи проекта (Pearls of Wisdom, 2014a). Этот долг рано или поздно приходится оплачивать. (См. урок 50 «Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения».)



Клиенты, страдающие из-за проблем с качеством продукта, проявляют недовольство. Почти каждый день я сталкиваюсь с каким-либо сайтом или другим продуктом, разработанным бездумно, сложным в использовании, тратящим мое время или просто работающим неправильно (Wiegers, 2021). Это раздражает, поскольку зачастую создать более качественный продукт ненамного сложнее. Мой коллега-программист, который периодически рассказывает мне о некачественных продуктах, заканчивает свои истории аббревиатурой «НРВН» — его сокращение от «Ничего не работает, и всем наплевать». К сожалению, он слишком часто оказывается прав.

В общие затраты, связанные с качеством, входит стоимость всех работ, выполняемых для предотвращения, обнаружения и исправления дефектов. (Подробнее о стоимости качества рассказывается в уроке 44 «Высокое качество естественным образом ведет к повышению производительности».) Бизнес-аналитики, разработчики и другие участники проекта будут совершать ошибки — это, увы, неизбежно. Поэтому вам необходимо принять на вооружение технические методы, которые помогут свести к минимуму количество создаваемых дефектов. Вам также необходимо развивать личную этику и организационную культуру, которые способствуют предотвращению дефектов и их раннему обнаружению. Страйтесь находить дефекты раньше, чем они нанесут слишком много вреда, то есть до того, как они повлекут за собой слишком много переделок.



Не всякий продукт должен быть идеальным, но каждый должен *положительно оцениваться* пользователями и другими заинтересованными сторонами. Первые пользователи инновационных продуктов относятся терпимо к дефектам, пока продукт позволяет им делать что-то новое. В других областях, таких как медицина, авиация и программные компоненты многократного использования, к качеству программных продуктов предъявляются гораздо более строгие требования. Первый шаг для любой проектной группы — решить, что означает качество для их продукта во всех его видах. После этого им пригодятся восемь уроков о качестве ПО, представленные в этой главе.





ПЕРВЫЕ ШАГИ: КАЧЕСТВО

Прежде чем вы перейдете к изучению уроков, связанных с качеством, предлагаю вам потратить несколько минут на следующие действия. По мере чтения подумайте, в какой степени каждый из этих пунктов применим к вашей организации или команде.

1. Как в вашей организации определяется внутреннее (с точки зрения разработчиков и специалистов по сопровождению) и внешнее (с точки зрения конечных пользователей) качество продуктов?
2. Документируют ли ваши проектные группы, что подразумевает понятие качества для каждого из их проектов? Ставят ли они измеримые цели в области качества?
3. Как в вашей организации оценивается соответствие каждого продукта ожиданиям в отношении качества со стороны команды и клиентов?
4. Перечислите методы поддержания качества программного обеспечения, в которых ваша организация особенно преуспела. Задокументирована ли информация об этих методах? Доступна ли она другим членам команды, чтобы они могли ознакомиться с этими методами и применять на практике?
5. Определите любые проблемы (болевые точки), которые можно отнести к недостаткам в том, как ваши команды подходят к вопросам качества программного обеспечения.
6. Укажите, как каждая проблема влияет на вашу способность успешно завершать проекты. Как все они мешают достижению успеха в бизнесе и организации, и ее клиентам? Проблемы с качеством приводят как к материальным, так и к нематериальным затратам, таким как незапланированные доработки, задержки, расходы на поддержку и техническое обслуживание, неудовлетворенность клиентов и нелестные отзывы о продуктах.
7. Для каждой проблемы, выявленной на шаге 5, определите основные причины, провоцирующие или усугубляющие ее. Проблемы, влияния и первопричины могут сливатся, поэтому постарайтесь разделить их и увидеть, как они связаны. Вы можете найти несколько основных причин, способствующих появлению одной и той же проблемы, или несколько проблем, обусловленных одной общей причиной.
8. Читая эту главу, перечислите любые практики, которые могут быть полезны вашей команде.

Урок 43

Решая вопрос о качестве программного обеспечения, вы можете выбирать: платить немало сейчас или позже, но еще больше

Предположим, я как бизнес-аналитик поговорил с клиентом, чтобы конкретизировать некоторые детали требований. Закончив беседу и вернувшись в свой офис, я записал все, что узнал о требованиях к моему проекту. На следующий день клиент написал мне такое электронное письмо: «Я только что поговорил с коллегой и понял, что неточно обозначил требования, о которых мы говорили вчера». Какой объем работы я должен сделать, чтобы исправить эту ошибку? Совсем небольшой — мне достаточно лишь обновить требование и привести его в соответствие с текущим запросом клиента. Предположим, что время, затраченное на внесение этой поправки, стоило компании 10 долларов.

Теперь рассмотрим другой случай: клиент связывается со мной через месяц или полгода после нашего разговора, чтобы сообщить о той же проблеме. Сколько теперь будет стоить исправление ошибки? Это зависит от того, какой объем работы успела проделать команда, опираясь на исходное неверное требование. Теперь компания должна будет заплатить не только за исправление требования 10 долларов, но еще и за переделку части дизайна, которая может стоить еще 30 или 40 долларов. А если разработчики уже реализовали исходное требование, то им придется изменить или полностью переписать код, обновить тесты, проверить новую реализацию и запустить регрессионные тесты, чтобы увидеть, не нарушили ли что-нибудь изменения в коде. Все это может стоить, пожалуй, еще 100 долларов. Возможно, кто-то также должен пересмотреть веб-страницу или файл со справкой, что еще больше увеличивает счет.

Пластиность программного обеспечения позволяет вносить изменения и исправления всякий раз, когда это оправданно. Но у каждого изменения есть цена. Даже обсуждение возможности добавления какого-то функционала или исправления ошибки, а затем решение не делать этого требуют времени. Чем дольше дефект в требованиях остается незамеченным и чем больше переделок придется вносить, чтобы исправить его, тем выше цена.



Кривая роста стоимости исправлений

Стоимость исправления дефекта зависит от того, когда он был допущен в продукте и когда кто-то его исправил. Кривая на рис. 6.2 показывает



ет, что стоимость исправления ошибок тем больше, чем позднее они обнаруживаются. Я опустил числовую шкалу по оси Y, поскольку разные источники приводят разные данные, а специалисты по программному обеспечению спорят о точных цифрах. Соотношение затрат и крутизны кривой зависит от типа продукта, жизненного цикла разработки и других факторов.

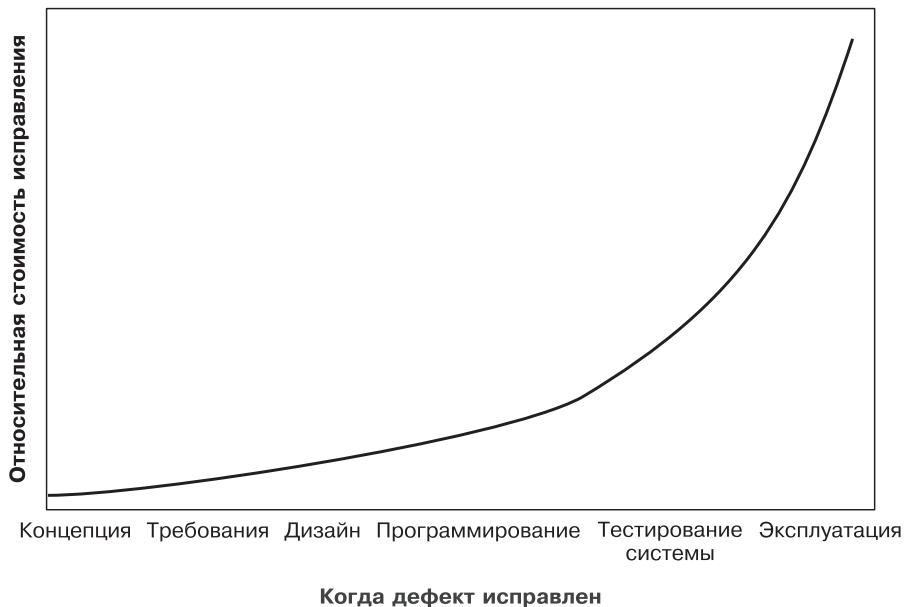


Рис. 6.2. Стоимость исправления дефекта быстро возрастает со временем

Например, согласно данным компании Hewlett-Packard, соотношение затрат может достигать 110:1 между обнаружением дефекта в процессе эксплуатации и обнаружением его во время разработки требований (Grady, 1999). По другим данным, соотношение затрат между обнаружением дефекта после выпуска и обнаружением его на этапе проектирования архитектуры составляет 30:1 (NIST, 2002). Для очень сложных программно-аппаратных систем коэффициент увеличения стоимости исправления на этапе эксплуатации по сравнению с исправлением на этапе разработки требований может варьироваться от 29 до более чем 1500 (Haskins et al., 2004).

Стоимость исправления дефекта зависит от того, когда он был допущен в продукте и когда его кто-то исправил.

Независимо от точных цифр все согласны, что чем раньше исправляется дефект, тем дешевле это обходится (Sanket, 2019; Winters et al., 2020). Это напоминает оплату счета по кредитной карте. Вы можете погасить кредит вовремя или заплатить меньшую сумму сейчас, а оставшуюся сумму вместе со значительными процентами и штрафами за просрочку — в будущем. Джоанна Ротман сравнила на примере трех гипотетических компаний использование разных стратегий устранения дефектов, эти стратегии характеризуются разными затратами (Rothman, 2000). Однако во всех трех сценариях видно, что чем позже команда исправляет дефект, тем дороже это обходится.

Некоторые утверждают, что методики Agile-разработки значительно сглаживают кривую стоимости изменений (Beck, Andres, 2005), но я не нашел никаких фактических данных, подтверждающих это. Однако данный урок не о стоимости внесения изменений, таких как добавление новых функций, а о цене, которую приходится платить за исправление дефектов. Дефект в требованиях, обнаруженный до реализации пользовательской истории в коде, исправить дешевле, чем тот же дефект, обнаруженный во время приемочного тестирования. Скотт Эмблер (Scott Ambler) предполагает, что относительная стоимость исправления дефектов ниже в Agile-проектах благодаря быстрой обратной связи, сокращающей время между выполнением некой работы и оценкой ее качества (Ambler, 2006). Это звучит правдоподобно, но лишь отчасти решает фундаментальную проблему стоимости исправления дефектов.

Проблема стоимости исправления заключается не только во времени между моментом появления дефекта в продукте и моментом его обнаружения. Коэффициент увеличения затрат во многом зависит от того, какой объем работы был выполнен для реализации ошибочного требования и как много теперь необходимо переделать. Исправить ошибку в коде почти ничего не стоит, если ваш партнер по парному программированию обнаружит ее через несколько мгновений после того, как



вы напечатали что-то не то и в вашем мозгу еще свежи воспоминания о проделанной работе. Однако если клиент звонит, чтобы сообщить об ошибке уже после передачи программного обеспечения в эксплуатацию, то ее исправление обойдется намного дороже. В качестве примера приведу рассказ моего друга-разработчика:



На той неделе я пропустил одну запятую (буквально) в сценарии на ColdFusion для сайта клиента. Это привело к аварии, которая вызвала задержку и разбор претензий пользователей. Кроме того, мы с ним вступили в переписку по электронной почте, а затем я открыл и исследовал исходный код, нашел ошибку, добавил запятую, повторно протестировал изменения и т. д. Столько труда и нервов из-за одной треклятой запятой.

В своем рассказе, помимо вопроса стоимости исправлений, мой друг упомянул еще один важный аспект позднего обнаружения дефектов, о котором следует помнить: негативное влияние на пользователей.

Сложнее найти

Диагностика сбоя занимает больше времени, если обусловившая его ошибка возникла давно. Обнаружив ошибку при просмотре некоторых требований, вы почти сразу поймете причину проблемы. Однако если клиент сообщит о неисправности через месяц или через пять лет после того, как кто-то написал требование, то поиск причины существенно усложняется. Является ли сбой следствием ошибочного требования, неправильного проектирования, ошибки в коде или стороннем компоненте? В этом заключается аргумент Эмблера в пользу более низких затрат на исправление дефектов в Agile-проектах: если дефекты обнаруживаются вскоре после их внесения, то найти ошибку, вызвавшую сбой, легче.

Выяснив первопричину системного сбоя, о котором сообщил клиент, вы должны выявить все рабочие продукты, на которых эта проблема сказалась негативно, исправить их, повторно протестировать систему, написать примечания к выпуску, повторно развернуть исправленный продукт и сообщить клиенту, что проблема устранена. Все это обходится очень дорого. Кроме того, чем позднее обнаружится проблема, тем более широкий круг заинтересованных сторон она затронет.

Ранние действия по улучшению качества

Серьезные дефекты, обнаруженные в ходе тестирования системы, могут увеличить объем работы по их исправлению. Дефекты, обнаруженные после выпуска продукта, могут воспрепятствовать работе пользователей и потребовать экстренного исправления и отвлечения членов команды от разработки нового продукта. Эта реальность наводит на несколько мыслей об оптимизации затрат на разработку качественного программного обеспечения.

Предотвращайте дефекты, а не исправляйте их

Действия, связанные с контролем качества, такие как тестирование, статический анализ и ревью кода, направлены на поиск дефектов. Деятельность по обеспечению качества в первую очередь направлена на предотвращение дефектов. Усовершенствованные процессы, лучшие технические приемы, более опытные специалисты и немного больше времени на тщательное выполнение работы — все это помогает предотвратить ошибки и избежать затрат на их исправление.



Выполняйте контроль качества раньше

Независимо от жизненного цикла разработки проекта, чем раньше обнаружится дефект, тем дешевле будет стоить его устранение. На каждый этапе работы над программным обеспечением совершается микропоследовательность таких действий, как определение требований, разработка дизайна и программирование, сменяющих друг друга слева направо на оси времени. Мы убедились в том, что устранение ошибок в требованиях максимально экономит время в будущем. Поэтому мы должны использовать все имеющиеся инструменты для поиска ошибок в требованиях и проектах до того, как они превратятся в ошибочный код.

Совместные обзоры и прототипирование — эффективные способы обнаружения ошибок в требованиях. Смещение тестирования с его традиционного места в конце последовательности разработки (то есть справа на временной шкале) влево дает особенно сильный эффект. В числе возможных стратегий можно назвать разработку через тестирование (Beck, 2003), создание приемочных тестов для конкретизации деталей требований (Agile Alliance, 2021b) и мое любимое — одновременное написание функциональных требований и соответствующих им тестов (Wiegers, Beatty, 2013).

Каждый раз, разрабатывая тесты вскоре после определения требований, я обнаруживаю ошибки как в требованиях, так и в тестах. Мыслительные процессы, связанные с написанием требований и тестов, дополняют друг друга, поэтому я считаю, что сочетание этих двух видов деятельности позволяет получить результат высочайшего качества. Написание тестов бизнес-аналитиком в сотрудничестве с тестировщиком дает положительный эффект, поскольку идея максимально раннего выявления проблем сочетается с участием нескольких людей, смотрящих на одно и то же явление с разных точек зрения. Написание тестов в начале цикла разработки не увеличивает время разработки проекта, а просто перераспределяет это время, перенося его в точку, где оно обеспечивает наибольшее влияние на качество. Эти концептуальные тесты могут быть преобразованы в подробные тестовые сценарии и процедуры по мере разработки.

В ходе реализации разработчики могут использовать инструменты статического и динамического анализа кода, помогающие выявить многие проблемы гораздо быстрее, чем при просмотре вручную. Эти инструменты могут находить ошибки, которые проявляются во время выполнения и с трудом обнаруживаются рецензентами кода, такие как повреждение и утечки памяти (Briski et al., 2008). В то же время рецензенты-люди могут выявлять логические ошибки и упущения в коде, которые не обнаруживаются автоматическими инструментами.



Большое значение также имеет время выполнения действий по контролю качества. Однажды я работал с женщиной-разработчицей, не позволявшей никому проверять свой код, пока тот не будет полностью реализован, протестирован, отформатирован и задокументирован. Таким способом она психологически защищалась от высказываний, что код еще не закончен. Каждая проблема, поднятая кем-то в ревью кода, вызывала защитную реакцию и попытку объяснить, что на самом деле все в порядке. Гораздо лучше начать с предварительных ревью части решаемой в данный момент задачи (будь то требования, проект, код или тесты), чтобы узнать мнение других и определиться, как лучше выполнить оставшуюся часть. Сдвиньте контроль качества влево, устраивая ревью раньше и чаще.

Отслеживайте дефекты, чтобы понять их

Самый эффективный способ контролировать дефекты — ограничить их рамками жизненного цикла разработки требований, проектирования, кода, в котором они возникли. Записывайте информацию о своих ошибках вместо того, чтобы просто исправлять их мимоходом. Опре-

деляйте причины происхождения каждого дефекта, чтобы понять, какие типы ошибок возникают чаще всего. Эта проблема возникла из-за того, что я не понял, чего хочет клиент? Я верно понял, что ему необходимо, но сделал неправильное предположение о других системных компонентах или интерфейсах? Я просто ошибся при программировании? Не случилось ли так, что запрос клиента на изменение не был доведен до сведения всех, кто должен был об этом знать?

Обратите внимание на действия в жизненном цикле (не обязательно отдельную фазу проекта), при выполнении которых возник каждый дефект, и на то, как он был обнаружен. Имея такую информацию, можно рассчитать, сколько дефектов переходят с этапа, на котором они появились, на более поздние этапы разработки, тем самым увеличивая стоимость их исправления. Эта информация покажет вам, какие практики обнаружения дефектов наиболее эффективны и где еще можно было бы улучшить контроль.

Минимизация возможностей появления дефектов и их обнаружение на ранних стадиях снижают общую стоимость разработки. Используйте все возможности своего арсенала инструментов контроля качества, начиная с самых первых этапов проекта.

Урок 44

Высокое качество естественным образом ведет к повышению производительности

Организации и люди, разрабатывающие программное обеспечение, хотели бы работать более продуктивно. Но осуществлению этого желания могут мешать многие факторы, и проблемы с качеством — самый большой из них. Команды планируют выполнить установленный объем работы за определенное время, но затем им приходится устранять проблемы, обнаруженные в завершенном проекте, или выделять время на исправление промышленной системы. Один из способов повысить продуктивность — создавать высококачественное ПО с самого начала, чтобы команды тратили меньше времени на доработку как в ходе разработки, так и после развертывания (Wiegers, 1996).

Терпеть не могу доделывать и переделывать то, что уже было сделано. Избегать этого я научился еще на уроках труда в 9-м классе. Наш первый проект заключался в том, чтобы взять короткий отрезок деревянного бруса 5×10 см и придать ему определенную форму, чтобы



попрактиковаться на нем в применении различных инструментов. Если мы просверливали отверстие не там, где надо, или состругивали лишнее, то нам приходилось начинать все сначала. Мне потребовалось девять попыток, чтобы сделать работу правильно.



Я заметил, что мой одноклассник работал медленнее, чем я, но закончил свой проект уже после второй попытки. Ему пришлось гораздо реже переделывать работу из-за ошибок и не нужно было покупать девять брусков. Как результат, качество его работы и продуктивность превзошли мои. Я усвоил важный урок: тише едешь — дальше будешь. С тех пор я стараюсь не делать одну и ту же работу дважды. Внимание к качеству с самого начала высвобождает время, которое можно посвятить новой, более полезной работе. Это верно в отношении создания изделий из дерева и еще более справедливо в отношении разработки программного обеспечения.

История двух проектов



Для иллюстрации того, как низкое качество ухудшает продуктивность, сравним два реальных проекта одной и той же компании, описанные консультантом Мэйлиром Пейдж-Джонсом, работавшим в команде Б. IT-отдел этой компании одновременно разработал два новых высокодоступных приложения для замены устаревших двадцатилетних систем. У нас есть два проекта, две команды, два подхода и два совершенно разных результата. (Подробнее об этом случае рассказывается во введении к главе 5.)



Подходы

Руководители команд А и Б составили сметы и графики, но все они были урезаны и сокращены высшим руководством. Команда А создала довольно длинную и скучную текстовую спецификацию требований, получила одобрение и вскоре приступила к разработке кода. Они относились к своей работе так: «Если не начать программировать сейчас, то мы не уложимся в срок». Команда А разработала проект своей базы данных, опираясь на процедурный код.

Руководитель команды Б твердо верил в программную инженерию. Команда оформила свои требования в основном в виде визуальных моделей, дополненных текстовыми описаниями вариантов использования, данных и их взаимосвязей, макетов страниц и т. д. Они разработали проект своей базы данных на основе диаграммы связей

классов и, опираясь на модели программного обеспечения, создали тестовые сценарии еще на ранних этапах разработки.

Результаты

Команда А с самого начала была немаленькой и к концу разработки проекта стала еще больше. Они уложились в срок, приняв несколько разработчиков и тестировщиков и много работая сверхурочно. Они превысили бюджет на 50 %, большую часть которого потратили на отладку, проводившуюся в течение нескольких месяцев перед развертыванием. После него команда ежедневно получала не менее одного сообщения от пользователей о том, что их система вышла из строя или «сделала что-то непонятное». Они создали «Отряд коммандос» для реагирования на постоянный поток проблем.

Команда Б изначально была небольшой и тоже выросла, хотя и не так сильно, как команда А. К намеченней дате завершения проекта команда Б имела работающую, но неполную систему, а превышение бюджета составило всего 10 %. Система работала хорошо, и пользователи внесли лишь несколько малозначительных предложений по улучшению.

Несколько месяцев спустя аудит обнаружил, что загадочные проблемы системы А были вызваны обширными повреждениями в базе данных, которая месяцами накапливала недостоверную информацию. Ручная чистка оказалась бесполезной. База данных вскоре снова была повреждена, и никто не знал почему. Команда А вернулась к устаревшей системе, которую они пытались заменить, на время, необходимое для полной переработки новой системы. Однако через несколько месяцев их система вообще не перезапустилась, и компания отказалась от нее как от неисправимой. Они запустили новый проект по перестройке системы А под управлением руководителя команды Б!

Анализ

Команда А быстро запустила в производство плохо спроектированную и наспех построенную систему, не используя надежные методы разработки программного обеспечения. Она потратила месяцы на доработку до и после развертывания, прежде чем компания наконец отказалась от дальнейшего финансирования проекта. Руководство ожидало, что после развертывания люди из команды А освободятся для работы над следующим проектом, но их «Отряд коммандос» был занят поиском проблем и внесением исправлений. Поскольку от их системы отказались, окончательная производительность команды А оказалась

нулевой. Слабый контроль качества на протяжении всего проекта стоил компании больших временных и финансовых затрат.

Команде Б потребовалось чуть больше времени, чтобы построить высококачественную систему, и немного дополнительных усилий по доработке, после чего большинство участников высвободились для работы над следующим проектом. Я всегда буду выбирать команду, похожую на команду Б.

Бич переделок

Существуют два основных класса доработок программного обеспечения: исправление дефектов и погашение технического долга. В предыдущем уроке было описано, как со временем растут затраты на исправление дефектов. Точно так же чем дольше в коде сохраняются недостатки, тем больше накапливается технического долга и тем больше работы требуется для улучшения проектного решения. (См. урок 50 «Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения».) Реорганизация упрощает поддержку и расширение кода, но иногда код необходимо реорганизовать лишь потому, что он был создан в спешке на основе далеко не идеального проекта. Объемная непредвиденная доработка — пустая трата времени, которая отвлекает разработчиков от создания дополнительной ценности для клиентов.

Слишком часто организации безоговорочно воспринимают доработку как неотъемлемую часть разработки программного обеспечения. Конечно, определенная доработка ПО неизбежна. Это обусловлено природой умственного труда, несовершенством человеческого общения и нашей неспособностью ясно видеть будущее. Лучше переделать проектное решение, чтобы добавить неожиданные новые функции, чем перепроектировать систему ради потенциального роста, которого никогда не будет. Тем не менее каждая команда должна стараться минимизировать переделки, которых можно избежать, улучшая качество своей работы с самого начала.

Планируйте доработку в виде отдельных задач в проекте, а не прячьте ее внутри задач по обнаружению дефектов.





Проектные группы не всегда учитывают вероятность переделки при планировании. Даже если их оценки точны в отношении разработки, они окажутся весьма далекими от реальности, как только переделки явят себя во всей красе. Я видел, как эта проблема возникала в проектах, где не выделялось времени на исправление ошибок, обнаруженных во время мероприятий по контролю качества, таких как тестирование или рецензирование. Я предлагаю планировать доработку в виде отдельных задач проекта, а не прятать ее внутри задач по обнаружению дефектов. Обеспечив видимость трудозатрат на доработку, вы сделаете первый шаг к их сокращению.

Организации, подсчитывающие, сколько сил и средств уходит на доработку программного обеспечения, могут получить пугающие цифры. Один из банков подсчитал, что тратит от 1 до 1,5 млн долларов *в месяц* на автоматическое повторное тестирование (McAllister, 2017). Различные исследования показывают, что команды разработчиков ПО могут тратить от 40 до 50 % своего времени на доработку, которой можно было бы избежать (Charette, 2005). Только подумайте, как подскочила бы продуктивность вашей команды, если бы они могли дополнительно тратить треть своего времени или более для разработки чего-то нового!

Если вы фиксируете некие показатели, характеризующие затраты на разработку программного обеспечения, то постарайтесь отделить затраты на поиск дефектов от затрат на их устранение. Узнайте, сколько времени вы тратите на доработку, когда и почему. Эти данные раскрывают широкие возможности для повышения продуктивности. Подсказка: до 85 % затрат на доработку приходится на дефекты в требованиях (Marasco, 2007). Наличие некоторых исходных данных о затратах на доработку позволит вам наметить цели для улучшения и увидеть, снижают ли более совершенные процессы и методы разработки необходимость в доработке (Hossain, 2018; Nussbaum, 2020). Сделав это в нашей команде разработчиков, мы смогли сократить затраты на исправление дефектов с 13,5 % от общего объема до стабильного уровня примерно в 2 % (Wiegers, 1996).



Стоимость качества

Возможно, вы слышали, что качество бесплатно. Так называлась классическая книга Филипа Б. Кросби *Quality is free*, вышедшая в 1979 году. Слова «качество бесплатно» означают, что дополнительные усилия, необходимые для надлежащего выполнения работы с первого раза, являются разумным вложением. На решение проблемы уходит

больше времени и денег, чем на ее предотвращение. Плохо выполненная работа доставляет хлопоты всем, кто находится далее в цепочке рабочего процесса, и имеет неприятные побочные последствия, такие как:

- накопление технического долга, из-за которого становится все труднее улучшать продукт;
- упущеные возможности и задержки в других проектах из-за того, что разработчики отвлекаются на переделки;
- перебои в обслуживании клиентов и последующие отчеты о проблемах, потеря доверия и, возможно, судебные иски;
- гарантийные претензии, возвраты и недовольные клиенты.



Под стоимостью качества понимается общая цена, которую компания платит за поставку продукции и услуг приемлемого качества. Стоимость качества складывается из четырех слагаемых (Crosby, 1979; American Society for Quality, 2021c):

- предотвращение дефектов — планирование качества, обучение, мероприятия по совершенствованию процессов, анализ первопричин;
- оценка качества — оценка рабочих продуктов и процессов на предмет проблем с качеством;
- внутренний отказ — анализ отказов и доработка для устранения проблем перед выпуском продукта;
- внешний отказ — анализ отказов и доработка для устранения проблем после доставки; работа с претензиями клиентов, исправление и замена продукции.

Экономия на затратах по предотвращению дефектов и оценке качества приводит к стремительному росту стоимости отказов. Помимо затрат времени и средств на доработку, внешние отказы могут привести к негативным последствиям для бизнеса, таким как снижение эффективности (как в случае с командой А в примере выше) и уход клиентов к конкурентам. Существует множество страшных историй о компаниях, понесших огромные убытки и потерявших доверие общественности в результате сбоев в программном обеспечении (McPeak, 2017; Krasner, 2018).

Для организаций, занимающихся разработкой программного обеспечения, полезно подсчитать свои общие затраты на качество и понять,

как эти суммы распределяются между разными видами деятельности по обеспечению качества. Для этого необходимо собрать и проанализировать данные, но они покажут, как расходуются средства на качество. Такие данные позволяют организации *выбирать*, на что именно потратить свои деньги.

Я построил табличную модель стоимости качества для одного из моих клиентов. Она позволяла им рассчитать, во сколько в среднем обходятся ошибки в требованиях или проектном решении. Узнав, какой процент их бюджета уходит на разработку нового программного обеспечения, а какой — на предотвращение дефектов, оценку качества, внутренние и внешние отказы, клиент смог уделить больше сил обеспечению качества с максимальной для себя выгодой. Такой анализ показывает, что вложения организации в предотвращение и раннее обнаружение дефектов окупаются.



Обычные человеческие ошибки и некоторые доработки неизбежны. Доработка может увеличить ценность продукта, но только если делает его более функциональным, эффективным, надежным или удобным в использовании. Руководители компаний могут согласиться на некоторые доработки как приемлемый компромисс между скоростью и небольшими первоначальными затратами. Это деловое решение хорошо выглядит в бухгалтерских книгах, но может привести к более дорогостоящим проблемам в будущем. Методы, описанные в конце урока 43, тоже сокращают затраты на доработку и тем самым снижают общие затраты организации на обеспечение качества и повышение производительности.



Я уже говорил, что терпеть не могу что-то переделывать?

Урок 45

У организаций никогда нет времени, чтобы правильно создать программное обеспечение, но они находят ресурсы, чтобы исправить его позже

В предыдущем уроке были описаны два проекта по замене устаревших систем, которые компания реализовала одновременно. Один проект завершился успешно, хотя и с небольшим превышением графика и бюджета. Другой существенно превысил бюджет и вовремя поставил систему, имеющую серьезные недостатки, и от нее в конечном итоге отказались. Но компания не просто забраковала неудачную систему:

«Ну, не получилось. Забудем и перейдем к следующему проекту». Но все еще нужно было заменить устаревшую систему, поэтому была предпринята еще одна попытка, на этот раз с использованием надежных подходов.



Я долго восхищался этой великой тайной бизнеса программного обеспечения. Многие проектные группы работают в условиях нереалистичного графика и ограниченного бюджета, что вынуждает их экономить на качестве. В результате часто появляется продукт, который необходимо долго и дорого приводить в порядок или даже отказываться от него. Однако каким-то образом организация находит время, деньги и людей для доработки или замены.

В моей школе в кабинете химии на стене висел плакат с вопросом: «Если у вас нет времени сделать правильно, то где вы возьмете время сделать заново?»



Почему не сразу?

Очевидно, если система настолько необходима и актуальна, что руководство оказывает сильное давление на работников, требуя ускорить ее развертывание, то стоит создавать ее должным образом. В моей школе в кабинете химии на стене висел плакат с вопросом: «Если у вас нет времени сделать правильно, то где вы возьмете время сделать заново?» Я запомнил это послание. Когда у команды разработчиков нет времени, квалифицированного персонала, надлежащих процессов или инструментов для правильного выполнения работы, им неизбежно придется переделывать хотя бы часть работы. Как мы видели в предыдущем уроке, такая доработка влечет снижение производительности.

К сожалению, многие не понимают, насколько важно потратить дополнительное время на разработку изначально правильного программного обеспечения, а не на его доработку позже. Время, необходимое для применения эффективных методов обеспечения качества, таких как технические экспертные оценки, часто не закладывается в график. В результате люди начинают проводить такие оценки, только осознав их важность. Даже если оценки запланированы как часть процесса разработки, проекты со слишком жесткими графиками могут их не выполнять, поскольку оказывается, что у людей нет времени на участие

в них. Отказ от экспертных оценок и других методов обеспечения качества означает не отсутствие дефектов, а то, что кто-то найдет их позже, когда последствия будут более серьезными.

Масштабные неудачи чаще всего являются результатом плохого управления, а не технических проблем. Недооценка объемов работ в сочетании с нереалистичной надеждой на то, что разработчики смогут работать быстрее, чем в прошлом, приводит к отставанию от графика и снижению качества. И простые разработчики, и руководители должны предусматривать время и действия, необходимые для достижения успеха, чтобы избежать потенциально огромных затрат времени и денег.

Синдром 100 миллионов долларов

Похоже, что только государство может позволить себе отказаться от несостоявшегося проекта стоимостью более 100 млн долларов. Корпорациям нужны новые системы для ведения бизнеса, поэтому они снова и снова будут браться за них. Правительства же иногда поднимают руки вверх или переключаются на план Б. В качестве одного из множества примеров можно привести программу расширенной автоматизации Федерального авиационного управления, которая была запущена в 1982 году в рамках широкомасштабной программы модернизации системы управления воздушным движением (УВД). Центральным элементом проекта была усовершенствованная система автоматизации, стоимость которой к запланированному моменту завершения в 1996 году оценивалась в 2,5 млрд долларов.

В проекте имели место множество задержек и перерасход средств, отчасти из-за изменений требований, которые вызвали значительную доработку. Проект был кардинально реструктурирован в 1994 году после того, как предполагаемая окончательная стоимость выросла примерно до 7 млрд долларов. По оценкам, некоторые основные компоненты отставали от графика на целых восемь лет (Barlas, 1996). Отдельные работы по проекту были отложены на будущее, для выполнения в рамках модернизации УВД, и все равно федеральное правительство понесло чистые убытки в размере около 1,5 млрд долларов (DOT, 1998).

Недавно один масштабный проект провалился в месте, где я живу. После того как в 2010 году Конгресс США принял Закон о доступности медицинского обслуживания, также известный как Obamacare, штаты создали биржи медицинских услуг, где граждане могли бы приобретать

медицинские страховки. Одни штаты создали собственные биржи, другие установили партнерские отношения между штатами и федеральным правительством, а третьи полагались на федеральную биржу HealthCare.gov. Мой штат Орегон попытался создать свою чрезвычайно сложную биржу медицинского страхования под названием Cover Oregon. Для реализации проекта штат привлек крупного подрядчика по разработке программного обеспечения. Потратив около 305 млн долларов за три года, штат отказался от проекта и переключился на HealthCare.gov (Wright, 2016). Проект Cover Oregon потерпел сокрушительное фиаско, повлекшее за собой гигантские судебные иски.

Достижение баланса

Почти все технические специалисты хотят добросовестно трудиться и предоставлять высококачественные продукты и услуги. Иногда это желание вступает в противоречие с внешними факторами, такими как смехотворно короткие сроки, продиктованные руководством, или правила, установленные руководящими органами. Специалисты-практики не всегда знают о бизнес-мотивах или причинах такого давления. Качество и целостность тоже должны быть частью обсуждения, когда команда обдумывает, что можно сделать, чтобы уложиться в сроки, достичь бизнес-целей и реализовать правильную и надежную функциональность.



Как и у многих, у меня тоже есть личная профессиональная философия: «Стремиться к совершенству; добиваться превосходства». Не все получается идеально, но я стараюсь сразу сделать свою работу хорошо, чтобы избежать финансовых потерь, затрат времени, позора и потенциальных юридических последствий, связанных с необходимостью переделывать все заново. Если для этого потребуется больше времени, пусть будет так. Выигрыш в долгосрочной перспективе стоит первоначальных инвестиций.

Урок 46

Остерегайтесь малозаметных разрывов между плохим и хорошим



Разница между качественным и некачественным продуктом порой бывает удивительно маленькой. Сведите большой и указательный пальцы так, чтобы они оказались на расстоянии примерно сантиметра друг от друга. Вот такой разрыв часто отделяет хорошее от плохого (Wiegers,

2019e). Во многих случаях разницу между качественным продуктом и некачественным определяют небольшой дополнительный анализ, опрос, проверка или тестирование. Говоря о малозаметных разрывах между плохим и хорошим, я имею в виду не обычные ошибки, которые время от времени совершают все, а проблемы, возникающие из-за поспешности, небрежности или невнимания к деталям.

Иллюстрация малозаметного разрыва между плохим и хорошим

Приведу один из примеров малозаметного разрыва между плохим и хорошим, которые встречаются в повседневной жизни (Wiegers, 2021). В прошлом году я купил кое-что из крупной бытовой техники. У меня возник вопрос, и я решил задать его производителю, заполнив форму на их сайте. Форма требовала, чтобы я выбрал тему, а затем подтему. Однако подтемы не отображались. Какую бы тему я ни выбрал, в списке подтем доступным был единственный вариант — подсказка по умолчанию: «Пожалуйста, выберите тему». Когда после безуспешной попытки выбрать подтему я попытался отправить форму, то получил сообщение об ошибке, что требуется указать подтему. Поскольку это было невозможно, я не мог отправить форму, и мне пришлось звонить производителю со всеми вытекающими отсюда трудностями поиска полезного человека из службы поддержки.



Никто не заметил эту проблему при тестировании сайта? Возможно, функция отлично работала во время разработки, но в промышленной версии не были заполнены соответствующие таблицы вариантов для выбора. А может, тестирование выявило проблему, но кто-то решил отложить ее исправление. Спустя много месяцев после того, как я сообщил о проблеме, веб-страница наконец была исправлена и в ней появилась возможность выбора подтем для каждой основной темы. Вероятно, это исправление обошлось компании ненамного дороже, чем если бы это было сделано сразу. Но сколько времени клиентов было потрачено впустую, прежде чем компания наконец исправила ошибку? Компании не должны считать время своих клиентов бесплатным.

Руководство должно формировать культуру, в которой члены команды ожидают и получают возможность хорошо выполнять свою работу с первого раза.



Как я уже упоминал, я не люблю передельывать то, что уже было сделано. Руководители организации устанавливают стандарты, избегая малозаметных разрывов между плохим и хорошим в своей работе, и не допускают их появления в работе других. Руководство должно формировать культуру, в которой члены команды ожидают и получают возможность хорошо выполнять свою работу с первого раза.

Малозаметные разрывы между плохим и хорошим в программном обеспечении



Чтобы избежать малозаметных разрывов между плохим и хорошим, часто нужно лишь немного подумать, прежде чем продолжить. Мне встречалось слишком много программных продуктов с ошибками, которые должны были быть обнаружены во время тестирования, или с плохим дизайном, по которому было видно, что пользовательскому опыту не уделили должного внимания. Например, когда я захожу на популярный сайт финансовых услуг, он сообщает, что у меня есть одно непрочитанное уведомление. Но когда я щелкаю на значке уведомления, то появляется сообщение: «У вас нет уведомлений». Другой пример: недавно я видел печатный отчет, на последней странице которого было написано «Страница 5 из 4». Подобные дефекты очень озадачивают меня.

Ниже представлены некоторые категории проблем, ухудшающих качество программного обеспечения, которых разработчики могут избежать.

- **Предположения.** Бизнес-аналитик может ошибиться в своих предположениях или записать предположение, сделанное клиентами, но затем забыть проверить, насколько оно верно.
- **Идеи решения.** Клиенты часто предлагают бизнес-аналитикам свои идеи решения вместо требований. Если бизнес-аналитик не заглянет за пределы предложенного решения, чтобы понять реальную потребность, то может взяться за решение неправильной задачи или выбрать неадекватное решение, которое придется исправлять позже.
- **Регрессионное тестирование.** Если не выполнить регрессионное тестирование после быстрого изменения кода, то можно пропустить ошибку в измененном коде. Даже небольшое изменение может неожиданно сломать что-то еще.

- **Обработка исключений.** Реализации могут настолько сосредоточиться на «счастливом пути» в ожидаемом поведении системы, что могут не справиться с распространенными ошибками. Отсутствующие, ошибочные или неправильно отформатированные данные могут привести к неожиданным результатам или даже сбою системы.
- **Изменение воздействия.** Иногда люди внедряют изменения, не задумываясь о том, как они повлияют на другие части системы или сопутствующие продукты. Изменение одного аспекта поведения системы может привести к нарушению взаимодействия с пользователем, если аналогичная функциональность, имеющаяся в других местах, не будет изменена соответствующим образом.

В уроке 44 «Высокое качество естественным образом ведет к повышению производительности» описываются стоимость качества и представление о том, что качество бесплатно. Качество не бесплатно в смысле стоимости. Предотвращение, обнаружение и исправление дефектов требуют дополнительных ресурсов. Тем не менее устранение малозаметных разрывов между плохим и хорошим обязательно окупится, так как вам не придется тратить еще больше ресурсов на устранение проблем.



Урок 47

Никогда не поддавайтесь уговорам руководителя или клиента сделать работу наспех



Однажды Чизуко, разработчик программного обеспечения, рассказала мне, что ее руководитель проекта потребовал: «Ради экономии времени вы должны отказаться от проведения любого модульного тестирования». Она была потрясена этим приказом. Будучи опытным разработчиком, Чизуко знала, насколько важно модульное тестирование, помогающее убедиться в правильности реализации программы. Чизуко чувствовала, что ее руководитель требует, чтобы она пошла по более короткому пути, в слабой надежде, что это каким-то образом ускорит работу. Возможно, это сэкономит время, но пропуск модульного тестирования, несомненно, приведет к пропуску дефектов, которые будут обнаружены позднее, чем следовало бы. К ее чести, вопреки приказу Чизуко решила продолжить модульное тестирование.

Все мы должны взять на себя обязательство следовать лучшим профессиональным практикам, адаптируя их так, чтобы получать наибольший положительный эффект в каждой ситуации.



Я давно считаю, что мы не должны позволять нашим руководителям, клиентам или коллегам утешать нас делать свою работу плохо (Wiegers, 1996). Следование принципам — вопрос личной и профессиональной честности. Все мы должны взять на себя обязательство следовать лучшим профессиональным практикам, адаптируя их так, чтобы получать наибольший положительный эффект в каждой ситуации. Оказавшись в ситуации, вызывающей дискомфорт в профессиональном плане, постараитесь описать, что вам нужно для того, чтобы сделать что-то, что не будет считаться плохой работой. Как и многое другое, эту философию можно довести до бесполезной крайности. Ищите баланс в достижении профессионального мастерства, не впадая в чрезмерный догматизм и жесткость.

Умение противостоять силе

Люди, наделенные властью, могут пытаться повлиять на вас разными способами, чтобы заставить сделать то, что вы считаете плохой работой. Предположим, человеку, которому вы представляете оценку стоимости предстоящих работ, не нравятся ваши цифры. Он пытается надавить на вас, чтобы уменьшить оценку и помочь ему сбалансировать бюджет или достичь его собственных целей. Мотивация понятна, но это не повод менять оценку.

Тот, кто требует от вас изменить оценку, сам может испытывать давление, о котором вы не подозреваете. Он имеет право знать, как вы получили вашу оценку, и обсудить возможность ее корректировки. (См. урок 28 «Не меняйте оценку в зависимости от того, что хочет услышать получатель».) Однако менять оценку только потому, что она кому-то не нравится, означает отказываться от вашей интерпретации реальности. Это не отменяет вероятного результата проекта.



Спешка в программировании

Предположим, вы работаете в IT-отделе и у вас появляется новый проект. Ваши партнеры по бизнесу могут попытаться потребовать от

команды разработчиков, чтобы она немедленно приступила к программированию, не имея экономического обоснования и четких требований. Возможно, у партнеров выделены финансы на проект, которые они хотят потратить как можно быстрее, прежде чем потеряют их. ИТ-персонал тоже может испытывать желание как можно быстрее приступить к работе. Разработчики могут не хотеть тратить время на обсуждение требований, поскольку те, скорее всего, все равно изменятся.

Как следствие, в таких случаях пишется много бесполезного кода, тогда как результат неясен. Слишком часто никто не несет ответственности за отсутствие цели, поскольку она все равно не была четко определена. Не лучше ли ИТ-отделу попытаться противостоять давлению со стороны бизнеса и не начинать путешествие, пока не будет определен пункт назначения?

Нехватка знаний

Люди, которые требуют от вас сделать что-то, что вы считаете неуместным, могут не понимать подходов к разработке программного обеспечения, которые вы пропагандируете. Например, кто-то может посчитать ненужным проведение совместных реview кода. Люди могут думать, что не стоит тратить время на обсуждение требований или их запись. Руководители или клиенты могут настаивать на доставке продукта, даже если он не соответствует всем критериям выпуска. Клиенты не всегда ценят ускоренную поставку продукта, полноценное использование которого может потребовать масштабных исправлений.

Однажды коллега предложил своему руководителю проекта технический подход к конкретной программе. Руководитель, который сам был опытным разработчиком, не оценил достоинств подхода и отверг идею, и мой коллега оказался перед выбором из трех вариантов.

1. Объяснить подход так, чтобы его механика и преимущества стали более очевидными.
2. Все равно использовать предложенную им стратегию, несмотря на отказ руководителя.
3. Следовать указаниям менее информированного руководителя и применять неоптимальный подход.

Как вы думаете, какой выбор будет лучшим? Я предлагаю сначала попробовать вариант 1 (информирование). Если эта попытка не удастся,



то выбрать вариант 2 и поступить правильно. В зависимости от мелочности руководителя этот выбор влечет за собой некоторые риски, но я думаю, что это лучшее решение для проекта.



Однажды я работал под началом руководителя, который не понимал, как можно написать пользовательскую документацию к новому приложению до завершения его разработки. Он был ученым, имевшим навыки программирования, и потому считал, что разбирается в разработке ПО. Я объяснил, что знаю, что будет делать система, благодаря зафиксированным требованиям и проработанному проектному решению. Поэтому я не терял времени и написал систему справки и руководство пользователя до того, как мы реализовали последнюю строчку кода.



Заказчик сказал мне, что не понимает, почему проект займет столько времени, сколько ожидает моя команда. Основываясь на своем ограниченном опыте работы с компьютерами, он заявил, что основная работа — это ПНК, или простое написание кода. Я раньше не слышал этого выражения, но оно точно не относилось к тому проекту, что я и постарался ему объяснить. Люди, которые не зарабатывают этим на жизнь, не понимают разницы между написанием кода и разработкой программного обеспечения.

Негласная этика



Независимые консультанты и подрядчики могут подвергаться различного рода давлению. Однажды мой потенциальный клиент попросил меня прийти к нему в компанию под вымышленным предлогом. Он хотел, чтобы в нашем контракте было указано, что я буду выполнять определенную работу, а на самом деле — заниматься чем-то другим. Клиент не мог получить финансирование на выполнение необходимых ему работ, но у него были средства для оплаты других услуг. Я посчитал его просьбу неэтичной и отказался от сделки. Работа на таких условиях стала бы моей профессиональной ошибкой и могла бы привести меня к юридическим проблемам, если бы руководители этого клиента узнали, что происходит.

В обход процессов



Процессы разрабатываются и устанавливаются не просто так. В бытность работы в Kodak, когда пользователи предлагали внести изменения

в какое-либо приложение, я направлял их к нашему очень простому инструменту регистрации запросов на изменение. Информация, предоставленная пользователем, позволяла соответствующим исполнителям принять правильные решения о запрошенных изменениях. Некоторые пользователи не хотели утруждать себя отправкой запроса и спрашивали: «А нельзя ли просто изменить то-то и то-то?» На что я неизменно отвечал: «Нет, извините». На мой взгляд, действие в обход установленных практических процессов ради удобства — плохая работа.

Возможно, вам придется объяснить, почему необходимо следовать подходу, который вы отстаиваете. Укажите, насколько это повышает качество и ценность проекта. Эта информация поможет другому человеку понять, почему вы сопротивляетесь его просьбам. Однако иногда встречаются просто неразумные люди. Даже если вы примените все свое красноречие, чтобы убедить их в обратном, они могут настаивать на своем предложении срезать углы или последовать неразумному подходу.

Предположим, вы сопротивляетесь выполнению просьб, которые считаете непрофессиональными или неэтичными. Вторая сторона может пожаловаться вашему руководителю, что вы тратите время на ненужные действия или отказываетесь сотрудничать. Руководитель может поддержать вас иликазать на вас дополнительное давление. Во втором случае вам придется выбирать: уступить давлению, согласившись с потенциальными негативными последствиями для проекта и вашей психики, или продолжить использовать лучшие профессиональные подходы, известные вам. Несмотря на возможные риски, я выбираю последнее.

Урок 48

Стремитесь к тому, чтобы дефект нашли коллеги, а не покупатели

Я допустил серьезную ошибку в рукописи недавней книги, и смысл моего высказывания получился полностью противоположным тому, что я вкладывал в него. К счастью, один из моих зорких рецензентов заметил ошибку. Я был очень благодарен. Было бы неловко, если бы книга вышла с этой ошибкой.



Даже самые опытные писатели, бизнес-аналитики, программисты и другие специалисты допускают ошибки. Независимо от того, насколько

хороша ваша работа, когда другие просматривают ее результаты, они улучшаются. Много лет назад я приучил себя просить коллег проверить мой код и все остальное, что создаю для программного проекта.

Я всегда чувствую себя неловко, когда рецензент замечает мою ошибку, но в голове сразу всплывает выражение «молодец, что заметил!».

Показать свое творение другим людям и просить их подсказать вам, что с ним не так, — это не инстинктивное, а приобретаемое поведение. Человеку свойственно смущаться или даже обижаться, когда другие находят недостатки в его работе. Я всегда чувствую себя неловко, когда рецензент замечает мою ошибку, но в голове сразу всплывает выражение «молодец, что заметил!». Когда я говорю рецензенту: «Спасибо, что заметил», — тон разговора становится менее напряженным, поскольку я выражая благодарность за найденную ошибку вместо того, чтобы изображать жертву или защищаться. Я бы предпочел, чтобы мои ошибки находили друзья или коллеги до выпуска, а не клиенты после выпуска.

Некоторые думают, что их работа не нуждается в рецензировании, но лучший разработчик программного обеспечения, которого я когда-либо знал, испытывал неуверенность, если никто другой не просмотрел его код. Этот человек знал, насколько важен вклад других умных разработчиков. Разные рецензенты поднимают разные вопросы и дают отзывы разного уровня — от поверхностных и очевидных до глубоких и хорошо продуманных. Это верно независимо от того, что рецензируется: рукопись книги, спецификация требований или программный код. Все точки зрения полезны.



Рецензирование коллегами — самая передовая практика разработки программного обеспечения. Убедившись в ее преимуществах за многие десятилетия, я бы не хотел работать в организации, где ревью и рецензирование кода не являются неотъемлемой частью культуры.

Преимущества ревью

Техническое рецензирование (ревью) — проверенный метод повышения качества и продуктивности. Он улучшает качество, позволяя обнару-

живать дефекты раньше, чем они могли бы быть выявлены в противном случае. Как мы уже знаем, раннее обнаружение дефектов повышает продуктивность, поскольку члены команды тратят меньше времени на исправление дефектов на более поздних этапах разработки или даже после поставки.

Люди часто откладывают свое обращение к рецензентам до завершения работы. Однако обзор разрабатываемого продукта до его завершения позволяет потребителям оценить, насколько хорошо этот продукт удовлетворит их потребности. Очень неприятно получить какой-либо документ, например с требованиями, и обнаружить, что он не содержит всей необходимой информации, включает нечто совершенно ненужное или организован не так, как вам нужно. Отзыв о документе до того, как тот будет завершен, позволяет автору внести корректины и сделать его более полезным для целевой аудитории.

За исключением парного программирования, мы редко видим чужую работу изнутри, разве что нам нужно исправить ошибку или добавить улучшение. Поскольку часто люди, которым в будущем придется вносить изменения в код, не являются его первоначальными авторами, им будет полезно познакомиться с кодом через механизм ревью. Если вы пригласите рецензентов, не входящих в проектную команду, то они смогут узнать о некоторых аспектах вашего продукта, а также увидеть, как работает другая команда. Это взаимное обогащение помогает распространять эффективные практики по всей организации.

В последнее время я часто встречаю в литературе по программному обеспечению дискуссии о ревью кода. Меня всегда радует, когда люди серьезно относятся к ревью. Однако разработчики программного обеспечения создают множество других артефактов, которые тоже являются достойными кандидатами на ревью. Вот почему я предпочитаю использовать более общий термин «партерское ревью» (peer review). Он означает не проверку коллег с нашей стороны, а приглашение профессионалов для проверки нашей работы. Помимо программного кода, проектная группа может создавать планы, требования в различных формах, несколько видов дизайна, планы тестирования и сценарии, экраны справки, документацию и многое другое. Все, что создает человек, может содержать ошибки, поэтому будет очень полезно, если кто-то еще просмотрит результаты его труда.



Разновидности ревью программного обеспечения

Рецензирование можно выполнять различными способами: онлайн или на личной встрече и с разной степенью строгости. Оба подхода имеют свои преимущества и недостатки. Личные встречи могут дать синергетический эффект, когда комментарий одного человека помогает другому обнаружить проблему, которую никто не замечал до этого. Но рецензирование на личной встрече стоит дороже, и его сложнее запланировать, чем рецензирование онлайн. Ниже представлено несколько способов, которые можно использовать для изучения результатов работы коллеги (Wiegers, 2002a).

- **Проверка коллегой.** Попросите одного из коллег просмотреть ваш код и внести предложения по улучшению или исправлению. Здесь самое важное — объединиться с кем-то, кто обладает зоркостью и имеет время, чтобы помочь. Предложите в случае чего оказать такую же услугу — это справедливо.
- **Круговое обсуждение.** Передайте фрагмент своей работы нескольким коллегам и попросите каждого написать отзыв. Для этого можно использовать инструменты рецензирования, позволяющие рецензентам видеть и обсуждать комментарии друг друга. Передача по кругу — хороший способ асинхронного или распределенного рецензирования, когда личная встреча участников неудобна или не нужна.
- **Пошаговое обсуждение.** Автор начинает обсуждение, объясняет, как работает его продукт, просит дать обратную связь. Пошаговые обсуждения часто используются для проверки проектного решения, когда требуется мозговой штурм с коллегами.
- **Командный обзор.** Автор заранее передает продукт и вспомогательные материалы нескольким рецензентам, чтобы у них было время изучить его и обозначить любые проблемы. Во время встречи рецензенты высказывают свои замечания. Ведущий собрание (модератор) следит за ходом дискуссии и за тем, чтобы группа обсуждала рабочий продукт в разумном темпе. Если взять слишком высокий темп, то можно упустить из виду какие-то дефекты; если слишком низкий, то продолжительная дискуссия может утомить участников. Секретарь собрания может записать поднятые вопросы в виде стандартных форм.
- **Инспекция.** Наиболее формальный тип рецензирования подразумевает участие нескольких персонажей, роли которых играют участ-

ники собрания: автор, ведущий (модератор), секретарь, инспектор и иногда читатель (Gilb, Graham, 1993; Radice, 2002). Инспекция — самый дорогостоящий метод рецензирования, но и самый эффективный, как показывают многочисленные исследования. Она лучше всего подходит для рецензирования продуктов с повышенным риском.

Даже если вы не практикуете ни один из перечисленных методов рецензирования, просто попросите коллегу посмотреть на вашу работу и помочь найти ошибку или немного улучшить ваш дизайн. Любой отзыв лучше, чем его отсутствие. Разработчики программного обеспечения в Google предложили несколько правил ревью кода, в том числе: будьте вежливы и профессиональны, вносите изменения небольшими порциями, давайте комментарии, хорошо описывающие изменения, и минимизируйте количество рецензентов (Winters et al., 2020).

Положительная сторона: культурные последствия ревью

Рецензирование предполагает и техническую деятельность, и межличностное взаимодействие. Порядок проведения рецензирования (или его отсутствие) в организации показывает ее отношение к качеству и командной работе. Если члены команды не решаются поделиться результатами своего труда, опасаясь критики, — это тревожный сигнал. Еще одним таким сигналом является критика рецензентами автора за ошибки или просто за то, что он выполнил работу не так, как сделали бы они. Плохо продуманные процедуры рецензирования могут нанести ущерб культуре команды разработчиков (Wiegers, 2002b).

В здоровой культуре разработки программного обеспечения члены команды не только предлагают, но и принимают конструктивную критику. Они не стремятся скрыть от посторонних глаз свою работу и охотно тратят часть своего времени на просмотр чужого кода, поскольку признают преимущества рецензирования. Это пример взаимовыручки: ты помогаешь мне, я помогаю тебе — и все выигрывают.

Один из моих клиентов использовал процедуру обязательного рецензирования. Участники назвали такие сеансы «погружением в аквари-



ум с акулами». Это негативный образ. Кому захочется оказаться в аквариуме с акулами, будучи незащищенным и держа наживку в руке? По завершении сеанса авторы чувствовали себя оскорблёнными или подвергшимися нападкам и никогда больше по своей воле не приглашали других оценить их работу. Эти шрамы могут остаться на долгие годы.

Рецензирование может улучшить сотрудничество в команде, если выполняется должным образом в нужное время и знающими людьми. Но оно также может быть вредным, если участники не задумываются о том, в какой форме дают обратную связь. Ниже перечислены рекомендации, помогающие рецензентам дать конструктивный отзыв, который люди посчитают полезным.

- При комментировании сосредоточьтесь на продукте, а не на авторе. Рецензентов приглашают не для того, чтобы они показали, насколько умны, а для того, чтобы они улучшили результат труда команды.
- Формулируйте комментарии как замечания, а не обвинения. Говорите «я» чаще, чем «ты». Фраза «Я не увидел, где инициализируются эти переменные» воспринимается более легко, чем «Ты не инициализировал эти переменные».
- Выискивая недостатки, сосредоточьтесь на содержании, а не на стиле. Автор может помочь, устранив легко обнаруживаемые проблемы, такие как типографские ошибки, перед рецензированием. Следуйте стандартным шаблонам документов и правилам форматирования кода (например, используйте красивый шрифт), чтобы стилистические вопросы не отвлекали от обсуждения сути.
- Если вы — автор, то отбросьте свое это и будьте более восприимчивы к предложениям по улучшению. В конечном итоге именно вы несете ответственность за качество своей работы, но принимайте во внимание вклад ваших коллег.



Не ждите, когда ваша организация определит процедуру и культуру рецензирования, — просто просите помощи у своих друзей и коллег. Основной фактор успеха любого рецензирования — ваша установка на то, что лучше пусть ваши коллеги обнаружат дефекты, чем пребывать в неведении относительно их наличия или отсутствия. Если вы не разделяете эту философию, то просто передайте свою работу на следующий этап разработки или пользователю, а затем ждите, когда вам позвонят.

Урок 49

Разработчики программного обеспечения любят инструменты, но дурак с инструментами — это вооруженный дурак

Мой друг Норм — опытный столяр. Он спроектировал и построил свою столярную мастерскую, включая само здание. В его мастерской бесчисленное множество ручных и электроинструментов, и он знает, как и когда использовать каждый из них правильно и безопасно. Опытные инженеры-программисты тоже имеют множество инструментов и знают, как их эффективно применять.

Возможно, вы слышали поговорку «Дурак с инструментом остается дураком», которую иногда приписывают инженеру-программисту Грэди Бучу (Grady Booch). Это слишком великодушно. Тому, кто не совсем понимает, что он делает, инструменты дают возможность сделать что-то быстрее и порой более опасным образом. Инструмент только усиливает их неэффективность. Все инструменты имеют преимущества и недостатки. Чтобы воспользоваться преимуществами, специалистам-практикам необходимо знать суть инструментов и методы их применения, чтобы правильно решать соответствующие задачи. В данном случае под инструментами я имею в виду не только пакеты программного обеспечения, облегчающие или автоматизирующие некоторые операции с проектом (оценка, моделирование, тестирование, совместная работа), но и специализированные средства разработки ПО, такие как сценарии использования.

Если люди не понимают того или иного приема и не знают, когда можно, а когда нельзя его использовать, то инструмент, позволяющий выполнять работу быстрее и эффективнее, не поможет.

Инструменты могут повысить продуктивность квалифицированных членов команды, но не улучшат работу неподготовленных людей. Предоставление инструмента менее способным разработчикам может ухудшить их продуктивность, если они используют его наобум. Если люди не понимают того или иного приема и не знают, когда можно,



а когда нельзя его использовать, то инструмент, позволяющий выполнить работу быстрее и эффективнее, не поможет.

Инструмент должен добавлять ценность



Инструменты могут помочь программистам правильно построить продукт, сэкономить время или повысить качество, но я видел множество примеров их неэффективного использования. Моя команда разработки программного обеспечения однажды выбрала Microsoft Project для планирования проектов. Большинство из нас нашли Project полезным для записи и определения последовательности задач, оценки их продолжительности и отслеживания хода выполнения. Однако одна из участниц команды слишком увлеклась. Она была единственной разработчицей в проекте с трехнедельными итерациями. В начале каждой итерации она тратила пару дней на создание подробного плана итерации в Microsoft Project, вплоть до того, что расписывала работы по часам. Я обеими руками за планирование, но ее подход к использованию этого инструмента был лишней тратой времени.



Я знаю одно государственное учреждение, которое приобрело высококлассный инструмент управления требованиями, но почти не выиграло от его использования. Участники команды записали сотни требований для своего проекта в традиционный документ спецификации. Затем импортировали эти требования в инструмент управления, но документ остался в репозитории. Всякий раз, когда требования менялись, бизнес-аналитику приходилось обновлять и документ, и содержимое, хранящееся в базе данных инструмента управления требованиями. Единственной важной функцией инструмента, которую использовала команда, было определение сложной сети связей между требованиями. Это полезная функция, но позже обнаружилось, что никто и никогда не использовал масштабные отчеты, созданные с ее помощью! Неэффективное использование инструментов этим учреждением отняло много времени и денег, но принесло мало пользы.



Инструменты моделирования легко использовать не по назначению. Аналитики и дизайнеры иногда уделяют чересчур много времени совершенствованию моделей. Я большой поклонник визуального моделирования, облегчающего итеративное мышление и выявление ошибок, но люди должны создавать модели выборочно. Моделирование хорошо изученных частей системы и детализация до мельчайших подробностей не делают проект пропорционально более ценным.

Однако неправильно могут использоваться не только автоматизированные инструменты, но и специализированные методы проектирования. Например, варианты использования помогают мне понять, что пользователи должны делать в системе, после чего я могу определить перечень функций для реализации. Но я знал некоторых людей, которые пытались принудительно впихнуть каждую известную часть функциональности в вариант использования просто потому, что таков метод работы с требованиями. Если вам уже известна какая-то необходимая функциональность, то я не вижу большого смысла в ее переупаковке только ради того, чтобы сказать, что у вас имеется полный набор вариантов использования.

Инструменты должны использоваться разумно

Я присутствовал в офисе моего клиента в тот же день, когда один из членов его команды настраивал инструмент запроса на изменение, который они только что приобрели. Я одобряю разумное использование механизмов контроля изменений, в том числе инструмента для сбора запросов и отслеживания их статуса с течением времени. Однако участник группы настроил **более 20** возможных статусов запросов: отправлено, оценено, одобрено, отложено и т. д. Даже притом что они логически разумны, никто не будет использовать 20 статусов — вполне достаточно семи. Из-за такой сложности на пользователей возлагается чрезмерное и нереалистичное бремя. Это может даже отбить у них желание вообще использовать данный инструмент, заставляющий их думать, что он приносит больше проблем, чем пользы.



Однажды, читая лекцию по передовым методам разработки программного обеспечения, я спросил студентов, используют ли они какие-либо инструменты статического анализа кода, такие как lint. Руководитель проекта сказал: «Да, у меня на столе десять копий PC-lint». Моей первой мыслью было: «Возможно, вы захотите раздать их разработчикам, так как они не приносят никакой пользы на вашем столе». Программные инструменты часто пылятся на полках и не используются. Если люди не понимают, как эффективно применять инструмент, то он для них бесполезен.



Тот же вопрос о применении инструментов статического анализа кода я задал в другой компании. Один из студентов ответил, что когда его команда запустила инструмент lint для проверки кода их системы, то он выдал 10 000 ошибок и предупреждений, поэтому они больше не использовали его. Если крупная программа никогда не проходила ав-



томатическую проверку, то при ее проверке наверняка будет выдано множество предупреждений. Многие из них будут ложными или несущественными, и команда решит их игнорировать. Но среди них, скорее всего, обнаружатся настоящие проблемы. Если есть такая возможность, то настройте инструменты так, чтобы можно было сосредоточиться на действительно важных вопросах и не отвлекаться на мелкие проблемы.



Я столкнулся с той же проблемой ложных срабатываний в коммерческом средстве проверки грамматики, которое недавно начал использовать для проверки своих статей и книг. Я игнорирую более половины проблем, о которых оно сообщает, поскольку они неактуальны для моего стиля письма или просто неверны. Чтобы найти действительно ценные замечания, нужно потратить массу времени на просмотр всех сообщений. К сожалению, в этом инструменте отсутствуют полезные настройки, которые позволили бы улучшить соотношение «сигнал/шум».

Инструмент — это не процесс



Иногда люди думают, что использование хорошего инструмента решает проблему. Однако инструмент не заменяет процесс, он лишь поддерживает его. Когда один из моих клиентов сказал мне, что использует инструмент отслеживания проблем, я задал несколько вопросов о процессе, который поддерживает этот инструмент. В результате я узнал, что у них нет определенного процесса получения и обработки отчетов о проблемах — только инструмент. Без сопутствующего практического процесса инструмент может внести дополнительную неразбериху, если не использовать его должным образом.



Инструменты могут заставить людей думать, что они делают свою работу лучше, чем есть на самом деле. Инструменты автоматизированного тестирования ничем не лучше запускаемых ими тестов. Возможность быстро запускать автоматические регрессионные тесты не означает, что эти тесты эффективно отыскивают ошибки. Инструмент, вычисляющий охват кода тестированием, может сообщить о большом проценте охвата, но это не гарантирует, что тестированию подвергся весь важный код. Высокий процент охвата не говорит о том, что проверен весь код, все логические ветви и все варианты выполнения со всеми возможными входными данными. Инструменты не заменяют людей. Люди, тестирующие программное обеспечение, могут обнаруживать проблемы, помимо тех, поиск которых запограммирован в инструментах тестирования.

Мне доводилось общаться с людьми, утверждавшими, что их проект отлично обрабатывает требования, поскольку они хранят их в инструменте управления требованиями. Такие инструменты предлагают множество ценных возможностей. Однако возможность генерировать хорошие отчеты не означает, что сами требования, хранящиеся в базе данных, достаточно хороши. Инструменты управления требованиями — яркая иллюстрация старого выражения в компьютерном мире: *GIGO — garbage in, garbage out* (мусор на входе, мусор на выходе). Инструмент не знает, являются ли требования точными, ясными или полными. Он не способен обнаруживать отсутствующие требования.



Вы должны знать не только о возможностях, но и об ограничениях каждого инструмента. Некоторые инструменты могут анализировать набор требований и отыскивать конфликты, дубликаты и неоднозначности, но эта оценка ничего не говорит о логической правильности или необходимости требований. Команде, которая использует инструмент управления требованиями, сначала нужно научиться выявлять, анализировать и определять требования. Покупка инструмента управления требованиями не сделает из вас опытного бизнес-аналитика. Вы должны освоить ручную процедуру и доказать себе, что в полной мере владеете ею, прежде чем автоматизировать ее (Davis, 1995).

Правильное применение инструментов и методов может повысить ценность проекта, качество работы и продуктивность, поднять планирование и сотрудничество на новый уровень, а также навести порядок в хаосе. Но даже лучшие инструменты не помогут преодолеть слабость процессов, неподготовленность членов команды, сложность инициатив по преобразованию или культурные проблемы (Costello, 2019). И всегда помните один из законов информатики Вигерса: «Искусственный интеллект не заменит настоящий» (Wiegers, 1989).

Урок 50

Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения

После выпуска системы в промышленную эксплуатацию начинается этап ее поддержки. Существуют четыре основные категории поддержки программного обеспечения (Merrill, 2019):

- адаптация — модификация системы в целях работы в изменившейся операционной среде;
- корректировка — диагностика и устранение дефектов;
- совершенствование — внесение изменений в целях повышения потребительской ценности, например добавление новых функций, повышение производительности и удобства использования;
- профилактика — оптимизация и реструктуризация кода в целях достижения большей эффективности, простоты, удобства сопровождения и надежности.



Для систем, разрабатываемых поэтапно, добавление новых запланированных и расширение существующих возможностей не считается совершенствованием. Это просто часть цикла разработки. Однако добавляемые улучшения могут по-прежнему требовать корректирующей поддержки в целях устранения дефектов.

Предыдущие уроки в этой главе показали, почему корректирующая поддержка со временем становится дороже и как проблемы с качеством подрывают продуктивность команды. Помимо недостатков требований и кода, недостатки проектирования программного обеспечения будут продолжать поглощать ресурсы, поскольку разработчики и сопровождающие со временем улучшают кодовую базу, осуществляя профилактическую поддержку.

Технический долг и профилактическая поддержка

Чтобы сохранить высокую скорость работы, команды разработчиков иногда допускают снижение качества, что приводит к образованию технического долга. Они могут не использовать методику защитного программирования, такую как проверка входных данных и обработка исключений. Быстро написанный код может быть недостаточно продуманным, работать только в данный момент, но не быть структурированным на долгую перспективу. Он может выполняться неэффективно или быть малопонятным для тех, кто будет сопровождать его в будущем. Вероятно, разработчики не предусмотрели возможность адаптации программного обеспечения или базы данных к будущим расширениям. Новая функциональность, наспех добавленная в надежную демонстрационную версию или прототип, может попасть в промышленную версию, увеличивая технический долг.

Быстрые исправления кода могут давать неожиданные побочные эффекты. Хрупкий код разрушается, когда кто-то его меняет, вызывая

каскад модификаций, необходимых для восстановления его работоспособности. Будущий разработчик может решить полностью перестроить проблемный модуль вместо того, чтобы попытаться включить в него новые возможности или заставить его работать в изменившейся среде.

Как и любой другой, технический долг рано или поздно должен быть погашен, причем с процентами. Чем дольше долг сохраняется в системе, тем больше процентов начисляется по нему. Выплата технического долга в программном обеспечении включает рефакторинг, реструктуризацию или переписывание кода (опять эта приставка *re-/nere-*). Уорд Каннингем (Ward Cunningham, 1992):



Небольшой долг ускоряет развитие, если он быстро погашается при переписывании кода... Опасность возникает, когда долг не выплачивается. Каждая минута, потраченная на не совсем правильный код, считается процентом по этому долгу. Целые инженерные организации могут затормозить работу, оказываясь под давлением непогашенного технического долга в реализации.

Значительная доля работ при осуществлении профилактической поддержки связана с устранением технического долга. Независимо от того, проводите ли вы рефакторинг кода, созданного на предыдущей итерации в текущем проекте, или работаете с хрупкой устаревшей системой, ваша цель должна заключаться в том, чтобы оставить код в лучшем состоянии, чем когда он попал к вам. Это более конструктивный подход, чем просто проклинивать предыдущих разработчиков, создавших беспорядок, с которым вы столкнулись.

Решение сводится к сознательному принятию допустимости некоего объема технического долга с учетом того, что позже вам придется потратить больше времени на профилактическое сопровождение.

Осознанный технический долг

Иногда разумно накопить некий объем технического долга, если команда полностью осознает, что переделка несовершенного проекта

в будущем обойдется дороже. Если ожидается, что код будет жить недолго, то, возможно, не стоит тратить время на его детальную проработку. Однако слишком часто это ожидание не оправдывается. Так называемый временный код, встраиваемый в прототип, слишком часто попадает в промышленную версию программного обеспечения и в дальнейшем сбивает с толку тех, кто занимается его сопровождением.

Если вы осознаете, что поступаете рационально и предусматриваете время в будущих итерациях на устранение недостатков, а не просто надеетесь, что они не вызовут проблем, то, возможно, имеет смысл отложить тщательную проработку проекта. Или если вы разрабатываете что-то новое и неопределенное либо проводите исследования и нашли проектное решение, работающее более или менее устойчиво в данный момент, то этого пока может быть достаточно. Однако в итоге вам все равно придется улучшить проект, поэтому убедитесь в том, что понимаете это.



Решение сводится к *сознательному* принятию допустимости некоего объема технического долга с учетом того, что позже вам придется потратить больше времени на профилактическое сопровождение. То есть имеются осознанный технический долг и случайный технический долг (Soni, 2020). Неспособность исправить недостатки проектного решения и кода усложняет работу с системой в последующих итерациях или во время эксплуатации. Эти накопленные проблемы замедляют дальнейшую разработку сейчас и потребуют чрезмерных усилий по поддержке позже.

Погашение технического долга добавляет в проект свои риски. Может показаться, что вы просто улучшаете то, что уже работает, но эти улучшения тоже требуют проверки и утверждения, как и любой другой код в проекте. Регрессионное тестирование и другие методы контроля качества, направленные на обнаружение плохих исправлений, требуют больше времени, чем написание самого кода. Чем масштабнее код и чем глубже перерабатывается проект, тем выше риск непреднамеренно нарушить нормальную работу чего-то еще.

Качественное проектирование — сейчас или позже

Всегда есть что-то более срочное, чем исправление существующего кода. Руководителю порой трудно выделить ресурсы на погашение техниче-

ского долга, когда клиенты требуют *незамедлительно* расширить возможности ПО. Руководители должны стиснуть зубы. Десятилетиями существуют многие программные приложения, кодовая база которых постоянно расширяется и становится все более разрозненной. Дэвид Райс (David Rice) отмечает (Rice, 2016):

Основная проблема при работе с унаследованным кодом — на внесение изменений необходимо много времени. Поэтому если вы хотите, чтобы ваш код был долговечным, то убедитесь, что будущим разработчикам будет приятно вносить в него изменения.

Возможно, «приятно» — чересчур смелое заявление. Тем не менее страйтесь создавать программное обеспечение, которое не вызовет муки у будущих разработчиков.

Сделайте профилактическую поддержку частью вашей повседневной деятельности по разработке, улучшайте проекты и код всякий раз, когда взаимодействуете с ними. Не закрывайте глаза на встречающиеся недостатки качества — минимизируйте их.

Постепенное профилактическое сопровождение похоже на ежедневную чистку зубов; доработка в целях уменьшения накопленного технического долга похожа на периодическое посещение стоматолога. Каждый день я трачу несколько минут на чистку зубов щеткой и зубной нитью, чтобы доставлять как можно меньше хлопот моему стоматологу-гигиенисту. И работа стоматолога, и разработка программного обеспечения будут менее болезненными, если решать проблемы по ходу дела и не позволять им накапливаться.

СЛЕДУЮЩИЕ ШАГИ: КАЧЕСТВО

1. Вернитесь к определению понятия «качество» во врезке «Первые шаги». Сейчас вы изменили бы это определение? Если да, то вы изменили свою оценку качества вашего продукта?
2. Определите, какие уроки, описанные в этой главе, имеют отношение к вашему опыту с точки зрения качества программного обеспечения.
3. Можете ли вы, опираясь на свой опыт, вспомнить какие-либо другие связанные с качеством уроки, которыми стоит поделиться с коллегами?



4. Перечислите описанные в этой главе методы, способные помочь в решении связанных с качеством проблем, которые мы определили во врезке «Первые шаги» в начале главы. Как каждый метод может улучшить качество ваших продуктов?
5. Как вы определили, приносит ли желаемые результаты каждый метод, озвученный на шаге 4? Насколько ценные для вас эти результаты?
6. Определите любые препятствия, которые могут затруднить применение методов, перечисленных на шаге 4. Как бы вы справились с ними? Заручились бы поддержкой коллег, готовых помочь вам в реализации этих методов?
7. Внедрите описания процессов, шаблоны, руководящие документы и другие инструменты, чтобы помочь будущим проектным командам эффективно применять ваши передовые методы обеспечения качества.

Глава 7

Совершенствование процессов

ВВЕДЕНИЕ

Возможно, вы помните первое предложение в этой книге: «Я не знаю никого, кто мог бы, положа руку на сердце, сказать: “Сегодня я пишу программное обеспечение лучше, чем когда-либо”». Если вы не можете с уверенностью сказать то же самое, то вам следует поискать более эффективные приемы реализации программных проектов. Вот что такое совершенствование процесса разработки ПО (Software Process Improvement, SPI).

Совершенствование процесса разработки: что и зачем

Цель SPI — снизить стоимость разработки и сопровождения программного обеспечения. Это не создание арсенала процессов и процедур и не соблюдение требований самой модной в настоящее время модели совершенствования процессов или системы управления проектами. Совершенствование процессов — это средство достижения превосходных бизнес-результатов, что бы ни подразумевалось под ними. Ваша цель может заключаться в ускорении доставки продуктов, уменьшении количества переделок, лучшем удовлетворении потребностей клиентов, снижении затрат на поддержку или в достижении всего перечисленного. Что-то должно измениться в работе вашей команды, чтобы данная цель стала достижимой. Это изменение и есть



совершенствование процесса разработки. Всякий раз, когда ваша команда проводит ретроспективный обзор, чтобы извлечь уроки и улучшить свою работу в следующий раз, она закладывает основу для совершенствования процессов. Каждый раз, применяя новую технику, чтобы сделать проект более эффективным и действенным, вы совершенствуете процесс.

Совершенствование процессов — это средство достижения превосходных бизнес-результатов, что бы ни подразумевалось под ними.

Начиная с конца 1980-х многие организации прилагали систематические усилия по совершенствованию процессов в своих отделах и проектных группах, добиваясь разной степени успеха. Эти подходы часто следовали выбранной модели SPI, такой как модель зрелости процессов разработки программного обеспечения (Capability Maturity Model for Software, CMM) (Paulk et al., 1995), модель зрелости процессов интеграции (Capability Maturity Model Integration, CMMI) (Chrissis et al., 2003). Тысячи организаций по всему миру, особенно государственных в США, по-прежнему ежегодно проводят официальную оценку процессов, следуя модели CMMI (CMMI Institute, 2017). Многие компании выяснили, что систематический подход к совершенствованию процессов помог им добиться превосходных результатов.



В частности, появление методик Agile-разработки было обусловлено частыми попытками совершенствования процессов, основанных на модели зрелости. Фактически некоторые первые методики Agile назывались облегченными. Двенадцатый принцип Манифеста Agile-разработки программного обеспечения гласит: «Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы» (Agile Alliance, 2021c). В этом суть совершенствования процессов.

Не бойтесь процессов

В некоторых кругах слово «процесс» имеет негативный оттенок. Иногда люди не осознают, что у них уже есть процесс разработки программного обеспечения, даже если он плохо определен или не задокументирован. Некоторые разработчики опасаются, что необходимость сле-

довать определенным процедурам будет ограничивать их стиль или подавлять творческий потенциал. Руководители могут опасаться, что соблюдение определенного процесса замедлит проект. Конечно, можно догматически применять неподходящие процессы, не добавляя ценности и не допуская изменений в проектах и людях. Но это не является обязательным требованием! Когда все работает правильно, организации добиваются успеха благодаря своим процессам, а не вопреки им.

Разумные и подходящие процессы помогают организациям, занимающимся разработкой программного обеспечения, добиваться успеха постоянно, а не только тогда, когда нужные люди объединяются, чтобы реализовать сложный проект, приложив титанические усилия. Процесс и творчество совместимы. Я следую процессу написания книг, но он нисколько не ограничивает слова, которые я использую. Мой процесс — это структура, которая экономит мое время, поддерживает организованность и позволяет постоянно следить за продвижением к моей цели — закончить хорошую книгу в срок. Имея возможность опереться на существующий процесс, я могу сконцентрировать свои умственные ресурсы на текущей задаче, а не на управлении усилиями.



Несмотря на концептуальную простоту, SPI — сложная задача. Нелегко заставить людей признать наличие недостатков в их нынешних методах работы. Любая проектная работа сложна, и как же уговорить команды тратить время на выявление и устранение недостатков? Убедить руководителей инвестировать в будущие стратегические преимущества, когда они сталкиваются с надвигающимися сроками, — значит сразиться в тяжелой битве. Изменить культуру организации непросто, однако SPI предполагает изменение культуры наряду с изменениями в технических и управленческих методах.

Как возвести SPI в привычку

Многие программы SPI не дают эффективных и устойчивых результатов. Большие блестящие инициативы по изменениям вводятся с помпой, но потом тихо исчезают без объявления и анализа причин. Организация отказывается от приложенных усилий и позже пробует что-то другое. Я думаю, что вы можете совершить только две неудачные попытки стратегического совершенствования, прежде чем люди решат, что организация несерьезно относится к изменениям. После двух неудач мало кто всерьез отнесется к следующей инициативе по изменению.



Чтобы достичь успеха в совершенствовании процессов, нужно время. Организации должны достаточно долго прикладывать усилия на этом поприще, чтобы получить первые плоды. Почти любой систематический подход к совершенствованию может улучшить результаты. Однако если вы остановитесь на полпути после вложения средств в оценку и обучение, но до того, как изменения оккупятся, то потеряете свои вложения. Крупномасштабные изменения процессов происходят не быстро, поэтому учитесь получать удовольствие от маленьких побед. Постарайтесь определить улучшения, которые можно быстро внедрить, чтобы решить известные проблемы, а также долгосрочные системные изменения.



Последовательность решений руководства тоже имеет значение. Один из моих клиентов поделился своими разочарованиями. Его организация в течение некоторого времени смогла добиться значительных успехов в реализации стратегии совершенствования на основе СММ. Однако старший руководитель решил пойти в другом направлении. Он отказался от СММ и переключился на систему управления качеством, основанную на стандарте ISO 9001. Все, кто усердно работал над внедрением стратегии СММ, были разочарованы, когда плоды их нелегкого труда оказались не нужны. Пренебрежение деятельностью по совершенствованию процессов может разочаровать тех, кто искренне заинтересован в том, чтобы работать лучше. Если в организации нет настоятельной необходимости соблюдать определенный стандарт, например в целях сертификации, то приемлема любая система разработки высококачественных процессов.



Приступив к новой работе и приняв руководство программой SPI в крупном корпоративном подразделении, я познакомился с женщиной, которая в течение года занимала похожую должность в аналогичном подразделении компании. Я спросил ее, как разработчики отнеслись к этой программе в ее подразделении. Она на мгновение задумалась, а затем ответила: «Они просто ждут, когда все закончится». Если рассматривать SPI просто как новомодную причуду руководства, то большинство практиков постараются просто переждать ее, пытаясь выполнить свою настоящую работу, несмотря на отвлекающие факторы. Это не способствует успеху изменений.

В этой главе представлены девять уроков, которые я усвоил за многие годы совершенствования процессов разработки программного обеспечения в моих организациях и организациях моих клиентов. Возможно, они помогут вам добиться успеха.



ПЕРВЫЕ ШАГИ: СОВЕРШЕНСТВОВАНИЕ ПРОЦЕССА РАЗРАБОТКИ

Прежде чем вы перейдете к изучению уроков, связанных с совершенствованием процессов, предлагаю вам потратить несколько минут на следующие действия. По мере чтения подумайте, в какой степени каждый из этих пунктов применим к вашей организации или команде.

1. Каких бизнес-результатов вы еще не достигли, что могло бы указывать на необходимость совершенствования процессов разработки программного обеспечения или управления?
2. Увенчались ли успехом прошлые инициативы SPI в вашей организации? Если вы добились некоторого успеха, то какие действия и взгляды окупились? Вы добились улучшения, применяя устоявшуюся модель совершенствования или доморощенные подходы?
3. Определите любые недостатки или проблемы в работе вашей организации, устранив которые можно улучшить процессы разработки программного обеспечения и управления.
4. Укажите, как каждая проблема влияет на вашу способность успешно выявлять, разрабатывать и внедрять передовые процессы и практики. Как эти проблемы мешают вам неуклонно совершенствовать подходы к созданию программного обеспечения или добиваться успеха в выпуске продуктов?
5. Для каждой проблемы, выявленной на шаге 3, определите основные причины, провоцирующие или усугубляющие ее. Проблемы, влияния и первопричины могут сливаться, поэтому постарайтесь разделить их и увидеть, как они связаны. Вы можете найти несколько основных причин, способствующих появлению одной и той же проблемы, или несколько проблем, обусловленных одной общей причиной.
6. Читая эту главу, перечислите любые практики, которые могут быть полезны вашей команде.

Разочарование неутешительными результатами — мощная мотивация попробовать другой подход. Однако при этом важно иметь уверенность, что любая новая стратегия, которую вы примете, имеет хорошие шан-

сы на успех в решении вашей проблемы. Организации иногда выбирают модные решения и горячие новинки в области разработки программного обеспечения, считая их волшебным эликсиром, способным решить их задачи.

Руководитель может прочитать о многообещающей, но, возможно, чесноке разрекламированной методологии и настаивать на немедленном принятии ее в организации. Я слышал об этом феномене, имеющем название «менеджмент по Businessweek». Возможно, разработчик пришел в восторг, посетив презентацию новой методики работы, и хочет опробовать ее. Стремление к совершенствованию похвально, но вы должны направить эту энергию в правильное русло и оценить, насколько хорошо потенциальное решение соответствует вашей культуре, прежде чем принять его.

За прошедшие годы люди изобрели бесчисленное множество парадигм, методологий и систем управления разработкой программного обеспечения, среди которых:

- анализ и проектирование структурированных систем;
- объектно-ориентированное программирование;
- информационная инженерия;
- быстрая разработка приложений;
- спиральная модель;
- разработка через тестирование;
- рациональный унифицированный процесс;
- DevOps.

Методология Agile-разработки во многих ее вариациях (экстремальное программирование, адаптивная разработка, разработка на основе функциональности, Scrum, Lean, Kanban, Scaled Agile Framework и др.) служит примером стремления к идеальным решениям.

Увы, как красноречиво сообщает нам Фредерик П. Брукс-младший, серебряной пули не существует: «Не бывает единой методологии ни в технологии, ни в способах управления, которая обещала бы за десятилетие увеличить продуктивность, надежность, простоту хотя бы на один порядок» (Brooks Jr., 1995). Все подходы из вышеупомянутого перечня имеют свои достоинства и недостатки; все они должны применяться к соответствующим задачам правильно подготовленными командами и руководителями. В качестве примера в дальнейшем

обсуждении я буду использовать гипотетический новый подход к разработке программного обеспечения под названием «Метод-9».

Прежде чем выбрать какой-либо новый подход к разработке, спросите себя: «Что мешает нам добиться таких же результатов, которые он обещает, уже сегодня?»

Сначала проблема, потом решение

В статьях и книгах, написанных изобретателями и первыми последователями «Метода-9», восхвалялись его преимущества. Некоторые компании выбрали «Метод-9», желая создавать продукты, которые лучше удовлетворяют потребности клиентов. Хотите быстрее доставлять полезное программное обеспечение? (А кто не хочет?) «Метод-9» поможет вам в этом. Хотите уменьшить количество дефектов, раздражающих клиентов и отнимающих у команды время на доработку? (Опять же, кто этого не хочет?) «Метод-9» придет вам на помощь! В этом суть совершенствования процессов: постановка целей, выявление препятствий и выбор методов, которые, по вашему мнению, могут помочь их устранить.

Однако прежде, чем выбрать какой-либо новый подход к разработке, спросите себя: «Что мешает нам добиться таких же результатов, которые он обещает, уже сегодня?» (Wiegers, 2019f). Если вы хотите быстрее доставлять полезные продукты, то что вам мешает? Если ваша цель — уменьшить количество дефектов и переделок, то почему сегодня ваши продукты содержат слишком много ошибок? Если вы стремитесь быстрее реагировать на изменяющиеся потребности, то что стоит на вашем пути?

Другими словами, если «Метод-9» является решением проблем (по крайней мере, согласно той статье, которую вы читали), то в чем заключалась их причина?

Я подозреваю, что не все организации тщательно анализируют первоначальные причины, прежде чем хвататься за решения, выглядящие многообещающими. Постановка целей совершенствования — отличное начало, но, помимо этого, важно понимать, какие препятствия стоят на пути к этим



целям. Лечить нужно причины, а не симптомы. Если вы не понимаете причины проблем, то выбор любого нового подхода — всего лишь выстрел в пустоту.

Пример основной причины

Предположим, вы хотите поставлять программные продукты, хорошо удовлетворяющие потребности клиентов. Вы прочитали, что в командах, применяющих «Метод-9», есть роль под названием «всевидящий гуру», который отвечает за то, чтобы продукт достиг желаемого результата. «Отлично! — думаете вы. — Всевидящий гуру позаботится о том, чтобы мы создали правильный продукт. Клиенты будут счастливы». Проблема решена, верно? Может быть, но, прежде чем изменять процессы, ваша команда должна понять, почему ваши продукты не вызывают восторга у клиентов уже сейчас.



Анализ первопричин — это процесс размышлений в обратном направлении, когда несколько раз задается вопрос «почему?», пока вы не доберетесь до проблем, на которые можно воздействовать с помощью тщательно подобранных действий по улучшению. Первая найденная причина может не оказывать прямого влияния и не быть конечной первопричиной. Следовательно, устранение этой начальной причины не решит проблему. Вам нужно спросить «почему?» еще раз или два, чтобы убедиться, что вы добрались до основания дерева анализа.

На рис. 7.1 показан фрагмент диаграммы «Рыбий скелет», также называемой диаграммой Исиакавы или причинно-следственной диаграммой, — удобного способа анализа первопричин. Все, что для этого нужно, — несколько заинтересованных лиц, белая доска и маркеры. Пройдемся по этой диаграмме.

Ваша цель — выпускать продукты, лучше удовлетворяющие потребности клиентов. Напишите эту цель вдоль длинной горизонтальной линии. В качестве альтернативы можно сформулировать ее как проблему: «Выпускаемый продукт не соответствует потребностям клиентов». В любом случае эта длинная горизонтальная линия — основа диаграммы «Рыбий скелет» — представляет вашу основную проблему.

Затем спросите свою команду: «Почему мы не удовлетворяем потребности наших клиентов?» С этого начинается анализ. Один из возможных ответов: команда не получает адекватной информации о требованиях от конечных пользователей — обычная ситуация. Запишите эту при-

чину вдоль диагональной линии, отходящей от формулировки цели. Это хорошее начало, но решение проблемы требует более глубокого понимания. Поэтому далее вы спрашиваете: «Почему мы не получаем такой информации?»



Рис. 7.1. Анализ первопричин часто изображается в виде
диаграммы «Рыбий скелет»

Один из членов группы говорит: «Мы пытались поговорить с реальными пользователями, но их руководители говорят, что они слишком заняты, чтобы работать с командой разработчиков». Кто-то еще жалуется, что представители клиента, работающие с командой, не имеют полного представления о реальных потребностях конечных пользователей. Напишите эти причины второго уровня вдоль горизонтальных линий, отходящих от диагональной линии родительской проблемы.

Третий участник отмечает, что разработчики, на которых возложена обязанность по выявлению требований, задают представителям клиента неправильные вопросы. Затем следует естественный вопрос: «Почему задаются неправильные вопросы?» Причин может быть несколько, в том числе отсутствие образования или интереса к требованиям со стороны разработчиков. Возможно, бизнес-анализ не является основным навыком команды или в команде нет подготовленного бизнес-аналитика. Каждая причина записывается вдоль новой диагональной линии, соединяющейся с родительской.

Теперь вы приближаетесь к фактическим препятствиям, стоящим на пути к желаемой цели. Продолжайте этот многоуровневый анализ до тех пор, пока участники не придут к полному пониманию, почему до

сих пор не достигнуты желаемые результаты. Я обнаружил, что эта техника очень эффективно концентрирует внимание участников и позволяет быстро достичь ясного понимания ситуации. Такая диаграмма может стать неразборчивой, поэтому попробуйте записать причины на стикерах, чтобы их можно было перетасовывать по мере продолжения исследования.

Постановка диагноза ведет к излечению

В ходе последующих мозговых штурмов члены команды могут найти практические решения для устранения этих первопричин, и тогда вы окажетесь на правильном пути к достижению высоких результатов. Возможно, вы придете к выводу, что добавление опытных бизнес-аналитиков в ваши команды может быть более ценным действием, чем принятие «Метода-9» с его «всевидящим гуру». Или, может быть, комбинация этих двух методов окажется ноу-хау. Вы не узнаете этого, пока не продумаете все до конца.



Рассматривая вопрос о применении нового метода разработки, не обращайте внимания на шумиху. Изучите предпосылки и риски, связанные с новым подходом, а затем сопоставьте их с реалистичной оценкой потенциальной выгоды. Ниже перечислено несколько хороших вопросов, которые следует задать себе.

- Потребуется ли команде обучение, инструменты или консультации, чтобы начать работу и двигаться вперед?
- Принесет ли решение выгоду по сравнению с вложениями?
- Какое культурное влияние окажет переход к новым методам на вашу команду, клиентов и их организаций?
- Насколько крутой может быть кривая обучения?

Результаты анализа первопричин могут помочь вам выявить более эффективные методы решения каждой обнаруженной проблемы. Если вы не изучите препятствия на пути к поставленным целям, то не удивляйтесь, что проблемы никуда не исчезнут и после перехода на другую стратегию развития. Попробуйте провести анализ первопричин, а не гнаться за самой горячей новинкой, название которой кто-то прочитал в заголовке.

Анализ первопричин занимает меньше времени, чем вы могли бы ожидать. Это разумное вложение средств в концентрацию ваших усилий

по совершенствованию процессов. Любой врач скажет вам, что прежде, чем назначать лечение, нужно поставить диагноз.

Урок 52

Не спрашивайте: «Что это даст мне?» Спрашивайте:
«Что это даст нам?»

Когда людям предлагают использовать новый подход к разработке, следовать другой процедуре или взяться за неожиданное задание, они инстинктивно задаются вопросом: «Что это даст мне?» Это естественная реакция, но не совсем правильный вопрос. Правильный вопрос: «Что это даст нам?» Местоимение «нам» в этом вопросе может относиться к остальным членам команды, ИТ-подразделению, компании или даже к человечеству в целом — к кому угодно, только не к отдельному человеку. Инициативы по внедрению изменений должны учитывать коллективные результаты работы, а не только влияние на продуктивность, эффективность или уровень комфорта каждого человека. Люди, руководящие совершенствованием процессов, должны иметь убедительный ответ на вопрос «Что это даст нам?» Если в этом есть какая-то ценность для нас, значит, есть ценность и для меня, учитывая, что я — часть нас.

Может показаться, что просьба к занятому члену проектной группы выполнить дополнительное задание, например проверить работу коллеги, не принесет ему никакой выгоды. Однако все вместе такие усилия позволяют команде сэкономить больше времени, чем потратил отдельный человек, и тем самым внести чистый положительный вклад в проект. Коллеги, проверяющие некоторые требования или код на наличие ошибок, тратят свое время. Ревью кода может занять два-три часа времени каждого участника. Эти часы рецензенты не смогут потратить на выполнение своих обязанностей. Однако эффективная проверка выявит дефекты, а, как мы уже знаем, чем раньше они обнаруживаются, тем дешевле обходится их исправление.

Люди, руководящие совершенствованием процессов, должны иметь убедительный ответ на вопрос:
«Что это нам даст?»

Выгода для команды

Чтобы увидеть соотношение командной и индивидуальной выгоды, рассмотрим гипотетический пример, когда рецензирование может принести существенную выгоду всей команде. Предположим, что Ари (бизнес-аналитик в моей команде) написала несколько страниц требований, сопроводив их некоторыми визуальными аналитическими моделями и таблицей состояний, и затем попросила меня и еще двух коллег просмотреть ее требования. Каждый из нас четырех потратил по часу на изучение материала перед встречей команды, которая тоже длилась час:

$$\begin{aligned} \text{затраты на подготовку} &= 1 \text{ час/рецензент} \times 4 \text{ рецензента} = 4 \text{ часа;} \\ \text{затраты на встречу} &= 1 \text{ час/рецензент} \times 4 \text{ рецензента} = 4 \text{ часа;} \\ \text{общие затраты на рецензирование} &= 4 \text{ часа} + 4 \text{ часа} = 8 \text{ часов.} \end{aligned}$$

Предположим, что в процессе рецензирования обнаружены 24 дефекта разной степени серьезности и на исправление каждого из них у Ари ушло в среднем 5 минут:

$$\text{фактические затраты на доработку} = 24 \text{ дефекта} \times 0,0833 \text{ часа/дефект} = 2 \text{ часа.}$$

А теперь представьте, что Ари не запросила рецензирование. Дефекты останутся в наборе требований и будут обнаружены позже, уже в цикле разработки. Ари все так же придется исправить их, а другим членам команды — переделывать проектное решение, код, тесты и документацию после исправления дефектов. На все эти работы легко может понадобиться в десять раз больше времени, чем на простое и быстрое исправление дефектов в одних только требованиях. Стоимость переделки может возрасти еще больше, если эти дефекты попадут в конечный продукт и будут обнаружены покупателями. Этот десятикратный рост усилий позволяет оценить потенциальную доработку, если бы Ари и компания не провели рецензирование:

$$\text{потенциальные затраты на доработку} = 24 \text{ дефекта} \times 0,833 \text{ часа/дефект} = 20 \text{ часов.}$$

То есть это гипотетическое рецензирование требований предотвратило потенциальные затраты 18 часов на доработку на последних этапах реализации проекта:

сэкономленное время = 20 часов потенциальных затрат – 2 часа фактических затрат = 18 часов.

Этот простой анализ показывает, что минимальная окупаемость затрат на рецензирование составляет 225 %:

окупаемость рецензирования = 18 часов сэкономленного времени ÷ 8 часов на рецензирование = 2,25 = 225 %.

Это ощутимая выгода для команды в целом. Это то, что получим мы все, даже если никто конкретно не получит ничего.

Многочисленные компании, измерившие преимущества введения обязательной процедуры рецензирования, называемой *инспектированием*, сообщили о более впечатляющих результатах, чем в этом примере. Например, Hewlett-Packard оценила рентабельность вложений в свою программу инспектирования как десятикратную (Grady, Van Slack, 1994). В IBM подсчитали, что час, потраченный на рецензирование, в среднем экономит 82 часа, потраченных на доработку, по сравнению с выявлением дефектов в выпущенном продукте (Holland, 1999). Как обычно, у вас могут получиться другие цифры, но лишь немногие методы разработки программного обеспечения могут десятикратно окупить вложения.



Личная выгода

Предположим, вы объясняете этот анализ члену команды, который не понимает, какую выгоду он получит, если потратит свое драгоценное время на рецензирование вашей работы. Он может согласиться с вашими доводами и признать, что два часа его времени, потраченные на анализ, могут принести значительную пользу команде, но по-прежнему не видит в этом личной выгоды. Как убедить его? Я рецензирую часто, поэтому нашел много преимуществ в том, чтобы тратить время на изучение чужой работы. Далее я перечислю некоторые из них.

- Каждый раз, наблюдая за работой коллеги, я узнаю что-то новое. Например, он может использовать метод программирования, с которым я не знаком, или, возможно, он нашел способ сообщить о требованиях, который лучше моего.
- Я начинаю лучше понимать некоторые аспекты проекта, и это может помочь мне лучше выполнять мою часть работы. Кроме того, пони-

мание позволит мне сохранить важные знания, если конкретный разработчик покинет организацию.

- Рецензирование способствует распространению знаний среди членов проектной команды, что повышает общую продуктивность. Весьма желательно, чтобы все члены команды делились своими знаниями друг с другом, и рецензирование — один из способов сделать это.

Стоят ли эти преимущества тех часов, которые я потратил на рецензирование чужой работы? Возможно, нет. Но есть еще кое-что. Я, в свою очередь, тоже могу попросить коллег проверить мою работу. Как разработчик программного обеспечения и автор книг и статей, я осознал огромную ценность исследования моей работы группой коллег. Ошибки, которые они находят, и их предложения по улучшению неизменно помогают мне создавать превосходные продукты.



Рецензии других коллег помогают понять, какие виды ошибок я совершаю. Это знание, в свою очередь, помогает мне с самого начала не допускать их в будущих проектах. Суть в том, что мое участие в коллективной деятельности по улучшению качества всегда окупается как для меня, так и для моих коллег.

Вносите свой вклад в общее дело

В следующий раз, когда коллега или руководитель попросит вас сделать в проекте что-то, что не принесет вам личной выгоды, подумайте не только о своих интересах. Сотрудники несут ответственность за соблюдение установленных правил и приемов разработки. Справедливо спросить: «Что нам это даст, если я это сделаю?» На просителе лежит бремя объяснения того, какую пользу всей команде принесет ваш вклад. А вы старайтесь внести свой вклад в успех команды.

Урок 53

Боль — лучшая мотивация для изменения методов работы



В декабре 2000 года, направляясь к своему клиенту для проведения консультации, я поскользнулся на обледенелых ступеньках, упал и сильно повредил правое плечо. Это была самая сильная боль, которую я когда-либо испытывал. Через три дня я вернулся домой и встретился

с врачом, который сообщил мне, что у меня разрыв вращательной манжеты плечевого сустава. Физиотерапевт порекомендовал мне комплекс упражнений, которые я мог выполнять дома. Боль сильно мотивировала меня выполнять упражнения и стараться быстрее восстановиться, поскольку я — правша и моя левая рука предназначена в основном для создания визуальной симметрии.

Для команд и организаций, как и для отдельных людей, боль является мощным мотиватором, подталкивающим к изменениям. Я говорю не о боли, искусственно вызванной извне, когда руководители или клиенты требуют невозможного, а о самой настоящей боли, которую команда испытывает от используемых методов работы. Один из способов побудить людей меняться — объяснить, какой удивительно зеленой станет трава, когда они достигнут уровня CMMI Level 5 или освоят методику Scrum-Lean-Kanban. Люди легко убеждаются в необходимости двигаться, если показать им, что трава позади них горит. Осуществляя совершенствование процессов, необходимо делать акцент на снижении сложности проекта сначала путем тушения, а затем предотвращения пожаров.



Чтобы мотивировать людей участвовать в изменении, обещанное уменьшение боли должно перевешивать дискомфорт, вызываемый самой процедурой совершенствования.

Мероприятия по совершенствованию процессов не особенно приятны. Они отвлекают от проектной работы, которая более интересна членам команды и приносит пользу бизнесу. Усилия по изменению могут казаться сизифовым трудом, поскольку много факторов препятствуют проведению устойчивых организационных изменений. Чтобы мотивировать людей участвовать в изменении, обещанное уменьшение боли должно перевешивать дискомфорт, вызываемый самой процедурой совершенствования. И в какой-то момент участники должны почувствовать, что боль уменьшилась, иначе откажутся участвовать в подобной работе в следующий раз. Найдите влиятельных лиц в вашей организации и их болевые точки, свяжите их с целями изменений — и у вас появится прочная основа, на которой можно строить усилия по трансформации методов работы.

Боль причиняет неудобства!



Однажды я руководил деятельностью SPI в быстро развивающейся группе, которая создавала сайты для крупной корпорации. Они были перегружены просьбами о разработке новых проектов и совершенствовании существующих сайтов. У них также была проблема с управлением конфигурацией: они использовали два веб-сервера с почти, но не совсем одинаковым содержимым. Все члены группы согласились с моими рекомендациями по внедрению практической системы запросов на изменение и более строгих методов управления конфигурацией. Они видели, насколько запутанной была текущая практика и скольких напрасных усилий она требовала. Внедренные нами процессы помогли значительно снизить уровень сложности управления конфигурацией.

Как бы вы определили «боль» в вашей организации? Какие хронические проблемы возникают в ваших проектах? Определив их, вы сможете сконцентрировать усилия по совершенствованию там, где они принесут наибольшую пользу. Приведу несколько типичных примеров болей в проектах:

- несоблюдение запланированных сроков поставки;
- выпуск продуктов с чрезмерным количеством дефектов или функциональными недостатками;
- неспособность поспевать за запросами на изменение;
- создание систем, которые трудно расширять без значительных доработок;
- поставка продуктов, не отвечающих в должной мере потребностям клиентов;
- частые сбои системы, вынуждающие дежурного специалиста по поддержке работать по ночам;
- взаимодействие с руководителями, недостаточно хорошо разбирающимися в текущих технологических проблемах и подходах к разработке программного обеспечения;
- сложности из-за рисков, которые не были выявлены или уменьшены.



Цель любой деятельности по оценке процесса (группового обсуждения, ретроспективного обзора проекта или оценки сторонним консультантом) — выявить эти проблемные области. После этого появляется возможность определить основные причины проблем и предпринять шаги

по их устранению. Как консультант, я редко делаюсь с клиентами наблюдениями, удивляющими меня, но об одном из них не могу не сказать: никто из них не нашел времени, чтобы воспротивиться боли и устраниТЬ ее. Сторонние наблюдатели могут помочь более ясно увидеть проблемы и мотивировать на выполнение действий.

Незаметная боль

Когда-то давно я выступал в роли заказчика и работал с разработчиком Джин над созданием базы данных и простого интерфейса запросов. У нас не было письменных требований, зато мы часто обсуждали возможности системы. В какой-то момент мне позвонил руководитель Джин и отчитал меня: «Перестаньте так часто менять требования», — сказал он. — Джин топчется на месте, поскольку вы слишком часто меняете их».



Это было очень неожиданно для меня. Джин никогда не говорила, что неуверенность в требованиях была проблемой. Трудности, которые мой подход вызвал у Джин, не были мне видны. Если бы она или ее руководитель с самого начала объяснили, какому процессу они предпочли бы следовать, я был бы рад принять его. Мы с Джин договорились о более структурированном подходе к определению моих требований и после этого начали двигаться вперед намного быстрее.

Данный опыт научил меня тому, что проблемы, затрагивающие одних участников проекта, могут быть незаметны другим. Это подчеркивает необходимость четкого информирования всех заинтересованных сторон об ожиданиях и проблемах. Кроме того, из данного урока я извлек важный вывод: трудно продать хорошую мышеловку тому, кто не знает, что у него есть мыши.



Если вы не замечаете негативных последствий использования текущего подхода, то, скорее всего, не будете восприимчивы к предложениям об изменениях. Любое предлагаемое изменение будет выглядеть как решение проблемы. Поэтому важным аспектом SPI выступает выявление причин и затрат, обусловленных проблемами, которые порождает процесс, и последующее информирование тех, кого это касается. Такая осведомленность может побудить всех участников начать поступать иным образом.

Иногда вы можете стимулировать это осознание. На своих занятиях я делю студентов на небольшие группы и обсуждаю проблемы, с кото-



рыми сталкиваются их проектные команды. Эти обсуждения особенно плодотворны, когда члены группы представляют разные точки зрения на проект: бизнес-аналитика, руководителя проекта, разработчика, заказчика, тестировщика, маркетолога и т. д. После одного из таких обсуждений представитель заказчика поделился откровением: «Теперь я больше сочувствую разработчикам». Это сочувствие — хорошая отправная точка для совершенствования подходов к работе, приносящего пользу всем.

Урок 54

Внедряя новые методы работы, делайте это мягко, но непрерывно

Когда несколько лет назад я занимался бизнесом в сфере SPI, в нашем сообществе ходил анекдот:

- Сколько руководителей по совершенствованию процессов требуется, чтобы заменить лампочку?
- Только один, но лампочка должна быть готова измениться.

Я слышал, что у терапевтов есть похожая шутка.



В этой шутке есть доля правды. Никто не может по-настоящему изменить что-то, что, по мнению других, работает и дает положительный результат. Но вы можете использовать механизмы, мотивирующие заинтересованные стороны действовать по-другому. Вы можете объяснить, почему изменения выгодны им и другим людям, надеяться, что они примут ваши доводы, и вознаграждать тех, кто решится на изменения. Вы можете угрожать или наказывать людей, если они не согласятся с вами, но это не лучший способ мотивации к совершенствованию процессов. В конечном счете каждый сам решает, готов ли он в будущем действовать иначе.

Руководство

Для того чтобы новые способы работы в организации, занимающейся разработкой программного обеспечения (будь то небольшая группа или целая компания), внедрялись эффективно, руководители должны оказывать постоянное мягкое давление в нужном направлении (The

Mann Group, 2019). Чтобы задать желаемое направление, нужно четко определить цели инициативы и довести их до сведения всех участников. Кроме того, они должны нести пользу бизнесу. Как организациям, так и отдельным людям с трудом даются быстрые и радикальные изменения. Постепенные изменения менее разрушительны, и к ним легче адаптироваться. Людям нужно время, чтобы усвоить новые практики и новый образ мышления.

Для руководителей также желательно заручиться моральной поддержкой. Все участники должны понимать, какой вклад они могут внести в успех проекта. Ищите первых последователей, более восприимчивых к нововведениям, которые могут выступать в качестве пропагандистов изменений среди коллег. Люди, которые внедряют изменения и подвергаются их влиянию, должны чувствовать, что их голоса услышаны и изменения не навязываются им.

Людям, которые руководят внедрением изменений, я могу дать несколько советов, как мягко и непрестанно оказывать давление, чтобы организация двигалась к цели.

- Определите цели инициативы по изменению, ее мотивацию и основные желаемые результаты. Расплывчатые цели, такие как «достичь мирового уровня» или «стать лидерами по продуктивности», бесполезны.
- Выберите метрики (ключевые показатели эффективности), которые будут показывать прогресс в достижении целей и влияние этих изменений на ускорение разработки проекта, но имейте в виду, что последние показатели являются запаздывающими. Деятельность по изменению инициируется, поскольку мы уверены, что новые подходы помогут добиться лучших результатов. Однако для того, чтобы эти новые подходы повлияли на проекты, нужно время. Информирование о положительных тенденциях может поддерживать энтузиазм участников.
- Ставьте реалистичные и осмыслиенные цели и ожидания. Члены команды будут сопротивляться попыткам быстро и радикально изменить подходы к работе. Они могут выполнять предписанные вами меры, но это не значит, что они искренне будут следовать инициативе или признают ее ценность.
- Относитесь к инициативе изменений как к проекту с конкретными действиями, результатами, вехами и обязанностями. И не забывайте о ресурсах. Предусмотрите дополнительное время, чтобы люди

могли привыкать к новым подходам параллельно с выполнением своих обязанностей по разработке проекта.

Цели инициативы по изменению должны быть четко определены, доведены до сведения всех участников и нести пользу бизнесу.

- Информируйте о продвижении инициативы изменений. Сделайте это информирование регулярной частью планерок и совещаний. Следите за прогрессом в достижении своих целей наряду с обычными действиями по отслеживанию проектов.
- Призывайте людей ответственно относиться к их обязательствам по выполнению определенных мероприятий в рамках SPI. Если работа над проектом всегда будет иметь исключительный приоритет, то мероприятия по совершенствованию будут игнорироваться.
- Стремитесь добиваться небольших успехов как можно раньше, чтобы показать, что инициатива по изменению уже начинает приносить результаты. Объясняйте, какие преимущества эти успехи приносят организации. Команды, успешно осваивающие новые практики, прокладывают путь другим.
- Публично говорите даже о небольших успехах как доказательство того, что внедрение новых подходов оправдывает себя. Поддерживайте информированность участников об этих успехах и оказывайте мягкое положительное давление до тех пор, пока новые приемы не укоренятся в умах членов команды.
- Обеспечьте обучение новым методам работы. Наблюдайте, как члены команды применяют новые знания, но учитывайте сложность обучения и не требуйте слишком много. Людям нужно время, чтобы научиться применять полученные знания на практике.



Обучение — это инвестиции в будущие результаты. На обзорной встрече за год, в которой приняли участие почти 2000 человек, директор корпоративной исследовательской лаборатории назвал достижением тот факт, что большое количество ученых прошло обучение планированию экспериментов. Это не достижение — это инвестиции. Я бы посоветовал руководству спросить ученых несколько месяцев спустя, что они стали делать лучше благодаря такому обучению. К сожалению, руководство не проверило, какую выгоду принесли инвестиции в обучение.

Управление вышестоящими руководителями

Совершенствование процессов оказывает влияние и на руководство, и на членов технических групп. Управление вышестоящими руководителями — ценный навык для любого, кто осуществляет внедрение изменений. Если этим занимаетесь вы, то научите своих руководителей говорить публично, какие модели поведения и результаты следует искаать, а также какие результаты должны вознаграждаться или корректироваться.

Линда — одна из моих клиенток — имела богатый опыт управления вышестоящими руководителями. Она придумала, как представить ценность изменений каждому руководителю так, чтобы найти у них отклик и получить поддержку. Она знала, как научить своих руководителей публично подчеркивать важность инициативы по изменению, которую возглавляла. Линда хорошо ориентировалась в организационной политике и умела объединять ключевых лидеров, стоящих за проектом изменений, не запутываясь в самой политике. Она могла подобрать ключик к каждому.



Одна организация, успешно продвигавшая программу SPI, задокументировала отношения и поведение руководителей, ожидаемые от них в будущем. Некоторые из них перечислены ниже.



- Я согласен, что требования являются фундаментальным продуктом работы.
- Я обязуюсь не определять стоимость и не составлять график реализации запросов клиента, пока не получу подтверждение от проектной группы.
- Я обязуюсь следовать процессу планирования проекта, даже если это противоречит требованиям заказчика.
- Я помогаю разрешать связанные с приоритетами конфликты, взаимодействуя с руководителями проектов и заказчиками.
- Я призываю членов команды учитывать обязательства, взятые во время планирования проекта, и не брать на себя чрезмерную нагрузку.
- Я прошу предоставлять данные о состоянии дел в проекте.
- Я обязуюсь помочь членам команды, устранивая препятствия, возникающие в процессе разработки программного обеспечения.
- Я обязуюсь создать среду, управляемую данными, используя соответствующие рычаги управления бизнесом.

- Я обязуюсь учитывать потребности, возникающие в ходе совершенствования процесса, при определении бюджета.
- Я обязуюсь поддерживать и защищать процесс разработки программного обеспечения перед заказчиком.

Независимо от того, стремитесь вы к организации более зрелой модели процессов, такой как CMMI, или внедряете Agile-разработку во всей организации, руководителям важно понимать, как должны изменяться их собственные действия и ожидания. Руководители, подающие пример, сами практикуют новые способы работы, публично поощряют их и посыпают положительный сигнал всем остальным, что способствует изменению культуры.

Мой отец однажды сказал мне, что если я когда-нибудь окажусь в толпе людей, пытающихся войти в одну дверь, то я должен просто перебирать ногами — и толпа сама внесет меня. Я опробовал этот прием — он работает. То же самое верно и в отношении совершенствования процессов разработки программного обеспечения. Продолжайте двигаться в нужном направлении — и будете постепенно осваивать новые и лучшие способы работы.

Урок 55

У вас нет времени, чтобы совершить все ошибки, сделанные до вас

Как я уже упоминал в главе 1, у меня мало формального образования в области разработки программного обеспечения, всего три курса программирования, которые я прошел в колледже давным-давно. Однако с тех пор я научился гораздо большему, читая книги и статьи, посещая учебные курсы, конференции и встречи членов профессионального сообщества. Я сужу о ценности обучения по тому, сколько идей, которые хотел бы опробовать по возвращении к работе, я записываю. (Надеюсь, вы делаете то же самое, читая эту книгу.) Лучшим я считаю такое обучение, в ходе которого я узнаю о методах, которыми хотелось бы поделиться с коллегами, чтобы мы все стали более умелыми.

Получать знания от других людей гораздо эффективнее, чем обретать их самостоятельно. В этом и смысл моей книги: я хочу поделиться своим опытом, чтобы сэкономить ваше время на изучение и попытки применить те же методы, что использовал я. Все профессионалы

должны тратить часть своего времени на обретение знаний и расширение навыков в этой постоянно развивающейся области.

Однажды я читал серию лекций для группы разработчиков программного обеспечения. Я предлагал каждому члену этой группы выбрать журнальную статью или главу книги и кратко пересказать ее остальным. Попутно мы обсудили, как можно применять этот навык в нашей работе. Каждый, кто посещал лекции, делился своим опытом с остальными. Такие обучающие лекции способствовали распространению актуальной информации по всей группе. Было немного неловко, когда я читал лекции в новой группе в компании Kodak, члены которой пересказывали, главу за главой, содержимое моей первой книги *Creating a Software Engineering Culture*. Тем не менее у нас было несколько интересных дискуссий, и слушатели смогли понять мою точку зрения, поскольку я работал с ними над совершенствованием их процессов разработки.



Переход от идеи к рутинной практике всегда требует обучения.

Не все методы, которые я привез домой с конференции, дали ожидаемый результат. Один докладчик с энтузиазмом рекламировал свой метод стимулирования идей, который применялся во время семинаров по выявлению требований. Вернувшись к работе, я попробовал его предложение, но меня постигла неудача. Я был в восторге от вариантов использования, когда услышал, как выступающие на конференциях хвалят их потенциал. Попытавшись применить варианты использования в первый раз, я изо всех сил старался объяснить эту методику представителям пользователей и заставить их работать. Я преодолел эту первоначальную трудность и затем смог успешно применить варианты использования. Мне пришлось проявить упорство, чтобы преодолеть дискомфорт при изучении нового навыка и получить результат. Переход от идеи к рутинной практике всегда требует обучения.

Кривая обучения

Кривая обучения описывает, как человек обретает навыки выполнения новой задачи или применения нового приема в зависимости от своего опыта. В жизни мы постоянно сталкиваемся с бесчисленными



кривыми обучения. Всякий раз, пытаясь сделать что-то новое, мы встаем на новую кривую. Не нужно ожидать, что весь потенциал метода раскроется с первой попытки. Когда проектные группы пробуют использовать незнакомые методы, в их планах должно быть предусмотрено время, необходимое на то, чтобы освоиться. Если им не удастся освоить новую практику, то затраченное время будет потеряно навсегда.

Вы, несомненно, заинтересованы в повышении общей производительности труда (продуктивности), которой позволяет добиться ваш набор приемов. На рис. 7.2 показано, что в самом начале вы имеете определенный уровень продуктивности, который хотите повысить с помощью усовершенствованного процесса, практики, метода или инструмента. Ваш первый шаг — обучение и обретение некоего опыта. Ваша продуктивность немедленно падает, поскольку в периоды обучения вы не выполняете полезную работу (Glass, 2003).

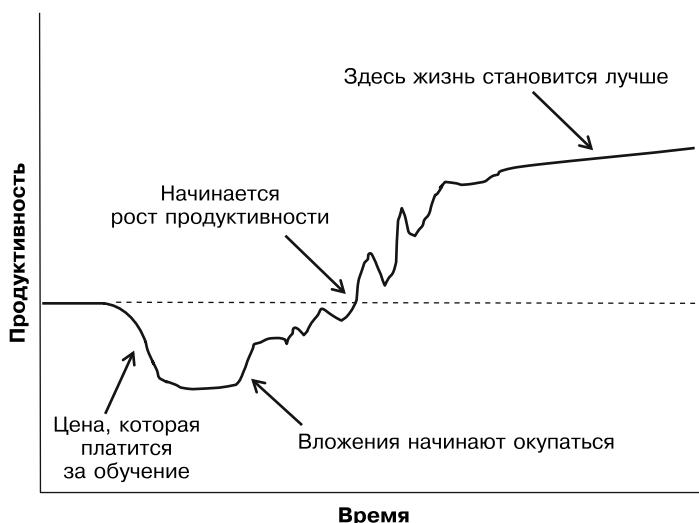


Рис. 7.2. Кривая обучения сначала снижает вашу производительность, а затем повышает ее

Производительность продолжает снижаться, пока вы тратите время на создание новых процессов, пытаетесь понять, как заставить работать новую технику, приобретаете новый инструмент и учитесь использовать его и т. д. По мере осваивания нового способа работы вы начинаете замечать первые успехи, а также некоторые неудачи — пилообразная

часть кривой роста продуктивности на рис. 7.2. Если все пойдет хорошо, то в конечном итоге ваши вложения окупятся и вы почувствуете, что эффективность, результативность и качество возросли. Помните о реальности кривой обучения, внедряя новые практики в личную деятельность, в деятельность своей команды и организации. И не поддавайтесь искушению сдаться до того, как вложения в обучение начнут окупаться.

Хорошие практики

Я нахожу забавным, когда кто-то жалуется на другого человека: «Он всегда думает, что его способ лучше». Конечно, он так думает! Зачем кому-то намеренно делать что-то, выбирая плохие способы? Это было бы глупо.

Проблема таких людей не в том, что они считают свой способ лучшим, а в том, что они не допускают мысли, что другие могут знать лучшие способы, и не желают учиться у них. Я собираю хорошие идеи и полезные приемы из всех встречающихся мне источников. Было бы глупо отказываться от чего-то, что явно лучше, чем то, что использовал я, только из-за гордыни.

Рецензирование коллегами дает хорошую возможность наблюдать за способами работы, которые используют другие. В ходе таких встреч можно увидеть, как кто-то использует незнакомые функции, хитрые приемы программирования или что-то еще, что зажигает в вашем мозгу лампочку. Однажды во время ревью кода я увидел, что стиль оформления комментариев, используемый другим программистом, явно превосходит мой. Я сразу же перенял его стиль и использую его до сих пор. Это простой способ учиться и совершенствоваться.

Люди часто заводят разговоры о лучших практиках, которые сразу же перерастают в споры о том, чья практика лучше для той или иной цели, в том или ином контексте. Простой поиск в Интернете даст вам обширный список статей и книг о лучших практиках разработки программного обеспечения (см., например: [Ford, 2017]). Все это хорошо, но «лучшая практика» — слишком строгий термин.

Мой совет — собирайте арсенал *хороших практик*. Чтобы попасть в него, прием просто должен быть лучше, чем тот, который вы используете сейчас. Например, в моей книге *Software Requirements*, написанной в соавторстве с Джой Битти (Joy Beatty), описываются более 50 хороших практик, связанных с разработкой требований и управлением



ими (Wiegers, Beatty, 2013). Некоторые из них я действительно считаю лучшими практиками в своей нише, но другие люди могут со мной не согласиться. Этот момент не стоит обсуждать — то, что работает у вас, может не работать у кого-то другого.



По мере накопления инструментов и методов придерживайтесь тех, которые вы успешно использовали в прошлом. Заменяйте текущую технику новой, только когда новая позволяет получить превосходные результаты во всех случаях. Часто техники могут мирно сосуществовать, и тогда у вас есть возможность выбирать между ними в зависимости от ситуации. Например, диаграмма действий UML — это, по сути, широкомасштабная блок-схема, но иногда простой блок-схемы более чем достаточно, чтобы увидеть все, что нужно. Так что овладейте обоими инструментами и используйте самый простой из них, позволяющий выполнять работу.

Как я упоминал выше, я сторонник обсуждения вариантов использования как метода выявления требований. В одной хорошей книге о вариантах использования авторы рекомендуют читателям отказаться от некоторых инструментов выявления требований: «Как и в случае со списками требований, мы рекомендуем исключить диаграммы потоков данных [Data Flow Diagram, DFD] из набора инструментов аналитика» (Kulak, Guiney, 2004). Я считаю это плохим советом, так как мне приходилось сталкиваться с ситуациями, когда диаграммы потоков данных были полезны при анализе требований. Почему бы не держать в ящике для инструментов несколько молотков на тот случай, если во время работы над проектом вы столкнетесь с разными гвоздями?

Урок 56

Здравый смысл и опыт иногда важнее определенного процесса



Я знал проектную команду, усердно пытавшуюся соответствовать жесткой структуре процессов, внедренной в их организации. Они выделили человека, считавшегося профессионалом, который должен был написать подробный план для этого шестимесячного проекта. По иронии, он завершил план незадолго до окончания проекта.

Такие истории портят репутацию SPI. Команда полностью упустила суть процесса: задача не в том, чтобы соответствовать процессу, а в том,

чтобы получить большую выгоду, применяя его. Процесс должен работать на вас, а не вы на него. Процессы, адаптированные к ситуации на основе опыта, ведут команды к повторному успеху. В каждой ситуации люди должны осознанно выбирать, масштабировать и адаптировать процессы для получения максимальной выгоды. Процесс — это фундамент, а не смирительная рубашка.



Просто наличие процесса не является гарантией его эффективности и уместности. Тем не менее процессы часто используются по уважительным причинам. Подвергать сомнению процесс — это нормально; ненормально нарушать его. Убедитесь, что понимаете обоснование и цель шага, который вы ставите под сомнение, прежде чем решить отказаться от него. В регулируемых отраслях в процесс добавляются дополнительные этапы, чтобы достичь обязательного соответствия системе управления качеством. Пропуск обязательного этапа может вызвать проблемы при попытке получить сертификат продукта. Однако часто этапы процесса существуют просто потому, что кто-то согласился с тем, что они будут полезны для работы команды и продукта клиента.

Процессы и ритмы

Организации внедряют процессы и методологии, чтобы добиться большей эффективности. Нередко процессы вносят значительный вклад в успех, хотя и не всегда. Процесс мог иметь смысл в момент написания, но теперь не подходит к текущей ситуации. Несмотря на благие намерения, люди иногда создают слишком сложные и ограничивающие процессы. Они могут выглядеть красиво, но если они не практичны, то люди будут их игнорировать. Определенный процесс представляет собой здравый смысл версии 1.0, но процессы не заменяют мышления. Здравый смысл, вырастающий из опыта, помогает мудрым практикам понять, когда разумно следовать процессу, а когда лучше немного изменить его, а затем усовершенствовать на основе новых знаний.



Всякий раз, когда люди не следуют процессу, который, по их утверждениям, они используют (и должны использовать), я вижу только три возможных варианта действий.

1. Начните следовать процессу, поскольку это лучший из известных способов выполнения данного конкретного действия.

2. Если процесс не соответствует вашим потребностям, то измените его и сделайте более эффективным и практичным, а затем следуйте ему.
3. Откажитесь от процесса и перестаньте притворяться, что следуете ему.

Слово «процесс» оставляет у некоторых людей неприятный осадок, но так быть не должно. Процесс просто описывает, как отдельные лица и группы должны выполнять свою работу. Он может быть случайным и хаотичным, хорошо структурированным и дисциплинирующим или находиться где-то посередине. Ситуация с проектом должна диктовать, насколько строгим должен быть процесс.

Меня не волнует, как вы создаете небольшой сайт или приложение, при условии, что они работают правильно для клиента, но меня очень волнует, как люди создают медицинские устройства и транспортные системы.



Один из моих клиентов сказал мне: «У нас не было процесса, но был ритм». Это хороший способ описания неформального процесса. Клиент имел в виду, что у их команды не было задокументированных процедур, но все знали, какие действия должны выполнять, и, взаимодействуя друг с другом, работали без сбоев.

Другая крайность (полное отсутствие задокументированного процесса) заключается в применении модели систематического улучшения процессов, такой как пятиуровневая модель CMMI (Chrissis et al., 2003). На уровне зрелости 1 люди выполняют свою работу так, как считают нужным в данный момент. Последующие уровни зрелости вводят все более структурированные процессы и оценки в духе постоянного совершенствования. В организациях, достигших уровня 5, имеется комплексный набор процессов, которые команды постоянно выполняют и совершенствуют.



Иногда, когда организации достигают высокого уровня зрелости, начинают происходить интересные вещи. Моя подруга, работавшая в организации с уровнем зрелости 5, сказала: «На самом деле мы не следуем никакому процессу. Именно так мы и работаем». У них был набор четко определенных процессов, но моя подруга и ее коллеги настолько впитали элементы процесса, что следовали ему неосознанно и автоматически, основываясь на многолетнем накопленном, записанном и совместно используемом опыте. Это идеальная цель.

Не будьте категоричны

Я верю в процессы, но они не являются догмой. Как упоминалось в предыдущем уроке, я с удовольствием добавляю в свою копилку новые методы решения различных проблем, но не собираю пошаговые сценарии. Я работаю в рамках общей структуры процессов и использую те хорошие практики, которые считаю цennыми. За прошедшие годы в мире программного обеспечения было создано множество методологий разработки и управления, которые претендуют на звание панацеи. Но вместо того, чтобы строго следовать какой-либо из них, я предпочитаю выбирать их лучшие элементы и применять в зависимости от ситуации.

Методы Agile-разработки программного обеспечения получили широкое распространение с конца 1990-х годов. В «Википедии» определены не менее 14 важных систем и 21 широко используемая практика Agile-разработки ПО (Wikipedia, 2021b). Ее разработчики объединили различные практики, ожидая, что определенная группа методов и действий даст наилучшие результаты.

Иногда я встречаю пуристов, которые очень обеспокоены соответствием, скажем, методологии Scrum. Они опасаются, что отказ или замена определенных практик будет означать отказ от самой методологии. Согласно руководству *The Scrum Guide* (Schwaber, Sutherland, 2020), это действительно так:

Scrum, как определено далее, неизменна. Если использовать только отдельные компоненты Scrum, то это будет не Scrum. Scrum существует только как целое, но может быть вместилищем для других техник, методологий и практик.

Ни один подход к разработке программного обеспечения не является настолько совершенным, чтобы команды не могли настраивать его под себя, повышая его ценность.

Изобретатели метода должны определить, что представляет собой этот метод — и что с того? Цель в том, чтобы соответствовать требованиям Scrum (или чему-либо еще) или же качественно и быстро вы-



полнить работу? Я слышал, как люди жалуются, что некоторые практики «не очень соответствуют методам Agile-разработки» или «их действия противоречат основным принципам Agile». Я снова спрашиваю: «И что с того?» Эта конкретная практика помогает проекту и организации добиться успеха в бизнесе или нет? Именно этот аспект должен быть определяющим фактором при принятии решения об использовании практики. Ни один подход к разработке программного обеспечения не является настолько совершенным, чтобы команды не могли настроить его под себя с целью повысить его ценность. Я прагматик, а не пуритан.



Agile-разработка, как и все процессы и методы, не является самоцелью. Это лишь средство достижения успеха в бизнесе. Я предлагаю всем членам команды накапливать инструменты и практики, позволяющие выполнять свою и совместную работу и достигать общей цели. Для меня неважно, соответствует ли конкретная практика конкретной философии разработки или управления. Тем не менее практика должна предлагать лучший способ выполнения работы в конкретной ситуации. Если это не так, то используйте что-нибудь другое.

Урок 57

Адаптируйте готовые шаблоны документов



Научившись ценить наличие письменного набора требований, я стал составлять простые списки функций, запрашиваемых моими клиентами. Это было хорошее начало, но у меня имелись и другие важные знания, связанные с требованиями, которые касались не только функциональности. Было непонятно, куда их приложить. Затем я обнаружил шаблон спецификации требований к программному обеспечению (Software Requirements Specification, SRS), описанный в уже устаревшем стандарте IEEE 830, называвшемся IEEE Recommended Practice for Software Requirements Specifications. Я принял этот шаблон, так как он содержал множество разделов, которые помогли мне систематизировать разнообразную информацию о требованиях. С опытом я изменил шаблон, чтобы он лучше подходил для систем, разрабатываемых моими командами. Во врезке ниже показан шаблон SRS, который я в итоге создал вместе с моей коллегой Джой Битти (Wiegers, Beatty, 2013).

Расширенный шаблон спецификации требований к программному обеспечению содержит разделы для многих типов информации.

РАСШИРЕННЫЙ ШАБЛОН СПЕЦИФИКАЦИИ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

- 1. Введение.**
 - 1.1. Цель.
 - 1.2. Условные обозначения.
 - 1.3. Сфера применения проекта.
 - 1.4. Ссылки.
 - 2. Общее описание.**
 - 2.1. Перспектива продукта.
 - 2.2. Классы и характеристики пользователей.
 - 2.3. Операционная среда.
 - 2.4. Ограничения и реализация.
 - 2.5. Допущения и зависимости.
 - 3. Системные функции.**
 - 3.x. Системная функция X.
 - 3.x.1. Описание.
 - 3.x.2. Функциональные требования.
 - 4. Требования к данным.**
 - 4.1. Логическая модель данных.
 - 4.2. Словарь данных.
 - 4.3. Отчеты.
 - 4.4. Целостность, хранение и удаление данных.
 - 5. Требования к внешнему интерфейсу.**
 - 5.1. Пользовательские интерфейсы.
 - 5.2. Программные интерфейсы.
 - 5.3. Аппаратные интерфейсы.
 - 5.4. Коммуникационные интерфейсы.
 - 6. Атрибуты качества.**
 - 6.1. Удобство использования.
 - 6.2. Производительность.
 - 6.3. Безопасность.
 - 6.4. Защищенность.
 - 6.x. [другие].
 - 7. Требования к международизации и локализации.**
 - 8. Другие требования.**
- Приложение А. Глоссарий.**
- Приложение Б. Модели анализа.**



Шаблоны документов предлагают несколько преимуществ. Они определяют согласованные способы организации проектной информации. Согласованность облегчает людям, работающим с этими документами, поиск нужной информации. Шаблон также может выявить потенциальные пробелы в знаниях автора документа о проекте, напомнить об информации, которую, возможно, следует добавить. Я использовал и разрабатывал шаблоны для многих типов проектных документов, такие как:

- запрос предложения;
- документ о видении и области применения;
- примеры использования;
- спецификация требований к программному обеспечению;
- устав проекта;
- план управления проектом;
- план управления рисками и список рисков;
- план управления конфигурацией;
- план тестирования;
- извлеченные уроки;
- план действий по совершенствованию процесса.

Предположим, я решил использовать шаблон, приведенный во врезке выше, для структурирования информации о требованиях к новой системе. Я не начинаю заполнять шаблон сверху вниз, а пишу определенные разделы по мере накопления соответствующей информации. Возможно, через какое-то время я замечу, что раздел 2.5 «Допущения и зависимости» не заполнен. Это побуждает меня задаться вопросом, есть ли какая-то недостающая информация о предположениях и зависимостях, которую я должен выяснить. Возможно, мне нужно побеседовать с определенными заинтересованными сторонами. Может быть, никто еще не указал на какие-либо вероятные предположения или зависимости и нам следует их определить. Некоторые допущения или зависимости могли быть записаны где-то еще; тогда, возможно, их следует переместить в этот раздел или добавить ссылки на них. Или, может быть, действительно нет никаких известных предположений или зависимостей — я должен это выяснить. Пустой раздел напоминает, что мне есть над чем поработать.

Кроме того, я должен подумать, что делать, если определенный раздел неактуален для моего проекта. Один из вариантов — просто удалить его

из документа с требованиями по завершении работы. Но отсутствие раздела может вызвать у читателя вопрос: «Я не увидел ничего о допущениях и зависимостях. Есть ли такие? Спрошу-ка я у кого-нибудь». Конечно, можно сохранить заголовок раздела и оставить сам раздел пустым, но это заставит читателя задаться вопросом, завершен ли документ. Я предпочитаю оставить заголовок и добавить пояснение: «Для этого проекта не было выявлено никаких допущений или зависимостей». Явное сообщение вызывает меньше путаницы, чем неявное.



Создание подходящего шаблона с чистого листа идет медленно и бессистемно. Мне нравится начинать с универсального шаблона, а затем адаптировать его к размеру, характеру и потребностям каждого проекта. Вот что я имею в виду под адаптацией готовых шаблонов документов. Многие технические стандарты описывают шаблоны документов. К организациям, выпускающим технические стандарты по разработке программного обеспечения, относятся:



- Институт инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers, IEEE);
- Международная организация по стандартизации (International Organization for Standardization, ISO);
- Международная электротехническая комиссия (International Electrotechnical Commission, IEC)

Например, международный стандарт ISO/IEC/IEEE 29148 наряду с поясняющей информацией содержит предлагаемые шаблоны для описания программного обеспечения, заинтересованных сторон и спецификаций системных требований (ISO/IEC/IEEE 2018). Поискав в Интернете, вы найдете множество шаблонов различных документов, доступных для скачивания, которые помогут вам начать работу.

Поскольку такие общие шаблоны предназначены для широкого круга проектов, они могут вам не подойти. Но они дадут много идей об информации, которую следует добавить, и способах ее организации. Концепция использования готовых шаблонов подразумевает возможность адаптировать эти шаблоны к вашей ситуации, как перечислено ниже.

- Удалите разделы, которые вам не нужны.
- Добавьте разделы, которых нет в шаблоне, но которые помогут вашему проекту.
- Упростите или объедините разделы шаблона, если это не вызовет путаницы.

- Измените терминологию в соответствии с вашим проектом или культурой.
- Реорганизуйте содержимое шаблона, чтобы оно лучше соответствовало потребностям вашей аудитории.
- Разделите или объедините шаблоны связанных документов, если это целесообразно, чтобы не получить чрезмерно большие документы, не увеличить их количество и избежать избыточности.

Если ваша организация работает над проектами нескольких классов или размеров, то создайте наборы шаблонов, подходящие для каждого класса. Это лучше, чем начинать каждый проект со стандартных шаблонов, которые плохо подходят для конкретного случая. Однажды один мой клиент попросил меня разработать процессы и соответствующие шаблоны документов для его сложных системных инженерных проектов. Позже этот клиент запросил параллельный набор процессов и шаблонов документов для его более новых и небольших Agile-проектов. В таких проектах тоже нужно фиксировать всю необходимую информацию, сводя к минимуму усилия по документированию, поэтому я просто упростил исходные процессы и шаблоны.



Подобно другим важным компонентам процесса, хороший шаблон поддержит вашу работу, а не помешает ей.



Компании добиваются успеха не потому, что пишут отличные спецификации или планы, а потому, что создают высококачественные информационные системы или коммерческие приложения. Хорошо составленные ключевые документы могут способствовать этому успеху. Некоторые люди не используют шаблоны, опасаясь, что они наложат на проект излишние ограничения. Люди могут быть обеспокоены тем, что команда сосредоточится на заполнении шаблона, а не на создании продукта. Если у вас нет договорных требований, то вы не обязаны заполнять каждый раздел шаблона. И, конечно же, не обязаны заполнять шаблон до начала разработки. Подобно другим важным компонентам процесса, хороший шаблон поддержит вашу работу, а не помешает ей.

Даже если ваша организация не использует документы для хранения информации, при разработке проекта все равно нужно записывать

и сохранять определенные знания в какой-либо форме. Вы можете предпочтеть использовать контрольные списки вместо шаблонов, чтобы не упустить из виду что-то важное. Подобно шаблону, контрольный список помогает оценить, насколько полным является набор информации, но не помогает систематизировать ее согласованным образом.

Многие организации хранят требования и другие сведения о проекте в одном инструменте. Такие инструменты позволяют определять шаблоны для хранимых объектов данных, подобные разделам в традиционном документе. При необходимости пользователи могут создавать документы в виде отчетов на основе содержимого базы данных инструмента. Всем, кто использует такой инструмент, важно понимать, что он является основным хранилищем текущей информации. Сгенерированный документ — это лишь моментальный снимок содержимого базы данных, который завтра может устареть.

Я ценю простые инструменты, такие как шаблоны, контрольные списки и формы, избавляющие меня от необходимости заново изобретать способы работы над каждым проектом. Я не хочу создавать документы ради документов или тратить время на мероприятия, которые не делают мою работу пропорционально более ценной. Продуманные шаблоны напоминают мне и моим коллегам о том, как наиболее эффективно внести свой вклад в проект. Мне это кажется разумной степенью структурированности процесса.

Урок 58

Если не тратить время на учебу и совершенствование, то не стоит ждать, что следующий проект будет реализован лучше предыдущего

Город, в котором я живу, недавно пострадал от сильной снежной бури. В моем полностью электрифицированном доме без камина и генератора стало довольно холодно, когда отключили электричество. Мы с женой были хорошо подготовлены, поэтому легко справились с ситуацией. Однако, когда снова дали свет, я подумал о том, как еще лучше подготовиться к следующей чрезвычайной ситуации. Я купил несколько предметов, которые сделают наше пребывание в доме в периоды длительного отключения электроэнергии еще более безопасным и комфорtnым, обновил планы хранения и приготовления пищи и написал контрольный список на случай чрезвычайной ситуации.



Процесс размышления о событии, имеющий целью пережить (в данном случае буквально) следующее событие, называется *ретроспективой*. Все команды разработчиков программного обеспечения должны проводить ретроспектины в конце цикла разработки (выпуска или итерации), по завершении проекта и при возникновении неожиданного или разрушительного события.

Оглядываясь в прошлое

Ретроспектива позволяет учиться и совершенствоваться. Команда оглядывается в прошлое, определяет, что получилось хорошо, а что нет, и благодаря этому получает возможность применить полученный опыт в будущей работе. Организация, которая не тратит время на такие размышления, основывает свое стремление к улучшению будущих результатов на надежде, а не на совершенствовании, в фундаменте которого лежит опыт.

Термин «ретроспектива», также известный как обзор результатов разработки проекта, обзор по завершении действий и постмортем-отчет (даже если проект был реализован!), прочно устоялся в мире программного обеспечения. Основополагающим источником знаний в этой области является книга Нормана Л. Керта *Project Retrospectives* (Kerth, 2001). Ретроспектива помогает ответить на четыре вопроса.

1. Что получилось хорошо и что мы хотели бы повторить?
2. Что получилось не так хорошо и где в следующий раз следует поступить иначе?
3. Что нас удивило и может быть опасным в будущем?
4. Есть ли что-то, чего мы еще не понимаем и должны исследовать?

Другой способ организовать это обсуждение — спросить участников ретроспективы: «Если я начну новый проект, похожий на тот, который вы только что завершили, то что вы мне посоветуете? Могли бы вы дать мне рекомендации?» Этот совет от людей, умудренных опытом, должен помочь ответить на четыре вышеупомянутых вопроса.

Ретроспектива должна проводиться с привлечением всех участников и в уважительном ключе.

Ретроспектива — в большей степени акт межличностного общения, а не техническое мероприятие. Соответственно она должна проводиться с привлечением всех участников и в уважительном ключе. Ретроспектива не должна использоваться для обвинения кого-то, это механизм изучения того, как можно добиться большего успеха в следующий раз (Winters et al., 2020). Важно объективно и беспристрастно исследовать прошлый опыт. Каждый, кто участвует в ретроспективе, должен помнить первый закон Керта:

Какие бы факты ни вскрылись в ходе ретроспективы, мы должны понимать и искренне верить, что каждый старался делать свою работу максимально хорошо, с учетом имеющейся информации, его или ее навыков и способностей, доступных ресурсов и ситуации, сложившейся на тот момент.



Сообщество последователей Agile-разработки приняло идею постоянного обучения, роста и адаптации за счет включения ретроспективы в каждую итерацию (Scaled Agile, 2021d). Напомним, что один из принципов Манифеста Agile для разработки программного обеспечения гласит: «Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы» (Agile Alliance, 2021c). Короткие итерации дают частую возможность переосмыслить опыт и повысить продуктивность в предстоящих итерациях. Команды, плохо знакомые с идеями Agile или выполняющие переход на другую Agile-инфраструктуру, интенсивно учатся на ранних итерациях. В книге Эстер Дерби (Esther Derby) и Дианы Ларсен (Diana Larsen) *Agile Retpectives* (Derby, Larsen, 2006) описывается палитра из 30 действий, из которых вы можете составить план проведения ретроспективы, подходящий для вашей команды.

Масштабируйте время, инвестируемое в ретроспективу, в зависимости от объема работы над проектом, качества ее выполнения и количества нового материала, который еще предстоит узнать. Agile-команде может потребоваться от 30 до 60 минут на обсуждение итогов в конце двухнедельного спринта. В больших проектных группах Керт (Kerth, 2001) предлагает выделять до трех дней, если люди еще не проводили ретроспективу и столкнулись с большими проблемами, требующими изучения. Мне приходилось организовывать ретроспективы, длившиеся по полдня. Чем больше потенциальных рычагов для улучшения будущих результатов, тем целесообразнее тратить время на подведение итогов.

Структура ретроспектины

Ретроспектива — не сеанс рассмотрения жалоб в свободной форме. Это структурированная и ограниченная по времени последовательность действий: планирование, начало мероприятия, сбор информации, определение приоритетов проблем, анализ проблем и принятие решения о том, что делать с информацией (Wiegers, 2007; Wiegers, 2019g). На рис. 7.3 показаны основные входные и выходные данные ретроспектины. Члены команды делятся своим опытом работы над проектом: что произошло, когда и чем закончилось. Рекомендуется спрашивать мнение всех участников проекта, поскольку точка зрения каждого уникальна. Мне нравится начинать с позитива: «Давайте с гордостью вспомним, что у нас получилось сделать эффективно и как мы помогли друг другу добиться успеха».



Рис. 7.3. Участники ретроспектины обобщают опыт проекта и выявляют извлеченные уроки, новые риски, пути совершенствования и определяют готовность команды внести ценные изменения

В бесстрастной и непредвзятой среде участники обсуждения могут поделиться своими эмоциональными взлетами и падениями. Один член команды мог быть расстроен задержкой в реализации части проекта, помешавшей ему выполнить взятые на себя обязательства. Другой мог бы поблагодарить коллег за дополнительную помощь. Возможно, люди испытывали сильный стресс, поскольку и без того напряженный график скжился до нереалистичного уровня и им пришлось работать до изнен-

можения, чтобы уложиться в срок. Психологическое благополучие команды — важный фактор успеха проекта. Обращение к эмоциональной стороне во время ретроспективы позволяет найти идеи, повышающие чувство удовлетворенности работой.

Участники ретроспективы сообщают любые данные, собранные командой, в виде типичных метрик:

- размер — количество и объем требований, пользовательских историй и других элементов;
- трудозатраты — запланированные и фактические;
- время — запланированная и фактическая календарная продолжительность;
- качество — количество дефектов и их виды, производительность системы и другие качественные характеристики.

Вся эта информация позволяет команде получить полное представление о том, как проходила работа и как участники относились к проекту.

Результаты ретроспективы, показанные выше на рис. 7.3, можно разделить на несколько категорий. Первая — это извлеченные уроки, которые следует запомнить в целях применения в последующих циклах разработки. Вы можете составить списки того, что обязательно нужно сделать снова, чего делать ни в коем случае не нужно и что следует сделать иначе. Если в вашей организации есть репозиторий извлеченных уроков, то добавьте в эту коллекцию уроки, извлеченные в ходе ретроспективы, чтобы помочь себе при реализации будущих проектов.

Неприятные сюрпризы можно занести в список потенциальных рисков вашей организации, которые следует изучать в каждом проекте. (См. урок 32 «Если вы не контролируете риски своего проекта, то они будут контролировать вас».) Риски проекта, которые не были должным образом снижены, и вообще все, что пошло не совсем так, как предполагалось, — это хорошие отправные точки для мероприятий по совершенствованию процессов. Наиболее важным результатом ретроспективы является стремление команды внести изменения, улучшающие их трудовую жизнь (Pearls of Wisdom, 2014b).



После ретроспективы

Ретроспектива сама по себе не влияет на то, как пойдут дела в следующий раз, — ее результаты обязательно должны учитываться в текущей

деятельности по совершенствованию процессов. Достижение лучших результатов в будущем требует плана действий, описывающего пути изменения процессов, которые команда должна изучить до или во время следующего цикла разработки. Люди, проводящие ретроспектины, должны уделить время изучению способов решения прошлых проблем. В это время они не смогут заниматься решением задач проекта, поэтому усилия по совершенствованию должны быть добавлены в графики проекта. По-иному никак. Если проектная команда проводит ретроспективу, но руководство не предоставляет ресурсы для решения выявленных проблем, то такая ретроспектива бесполезна.

У команд, не имеющих свободных промежутков времени между циклами разработки, не будет возможности обдумать, изучить и переориентировать свои возможности (DeMarco, 2001). Поэтому добавляйте в график время для обучения и экспериментов, чтобы люди могли учиться эффективно применять новые практики, инструменты и методы. Без этого бессмысленно ожидать, что разработка следующего проекта пройдет более гладко. Проведение ретроспектив без внесения каких-либо изменений в процессы — пустая трата времени, отбивающая у участников охоту участвовать в них.

Ретроспектива не бесплатна — она требует времени, усилий и, возможно, финансовых вложений. Однако эти вложения с лихвой окупаются за счет повышения продуктивности команды в будущем. Ретроспективы должны сопровождаться непрерывным совершенствованием, запускать изменения, нацеленные на устранение известных источников боли. Команды, стремящиеся к постоянному совершенствованию, используют ретроспективы как мощный инструмент, помогающий достичь этой цели.

Ретроспективы особенно ценные, когда культура их проведения способствует выслушиванию и принятию во внимание откровенных отзывов. Ритуал ретроспективы объединяет группу людей, помогает им взглянуть на свой опыт с более широкой точки зрения, чем доступно любому отдельному участнику. По мере размышлений о только что завершенной работе группа может выработать новый способ развития, который, как ожидается, выведет ее на более высокое плато профессиональной деятельности. Высший признак успеха ретроспективы — наличие устойчивых изменений. Это говорит о том, что время, потраченное командой на размышления о прошлых событиях, приносит долговременную пользу.



Урок 59**Самая удручающая закономерность в индустрии программного обеспечения — повторение одних и тех же неэффективных действий снова и снова**

Когда я работал в Kodak, компания проводила ежегодную внутреннюю конференцию по качеству программного обеспечения. Однажды меня пригласили в комитет по планированию конференции. Я попросил показать методическое руководство по планированию и проведению конференций и услышал в ответ: «У нас его нет».



Меня это сильно удивило. Я думал, что группа, состоящая из специалистов по качеству и совершенствованию процессов, будет находиться на верхней строчке в списке групп, практикующих накопление процедур, контрольных списков и уроков, извлеченных из предыдущего опыта. Такой ресурс особенно ценен, когда состав участников меняется каждый год, как это было у нас. Каждый год группе планирования приходилось заново придумывать, как планировать и проводить конференцию, восполняя все пробелы с нуля. Это очень неэффективно! (См. урок 7 «Запись знаний обходится дешевле, чем повторное их обретение».) В тот год, когда я работал в комитете, мы начали писать методическое руководство. Я могу только надеяться, что люди, работавшие над планированием последующих конференций, обращались к этому ресурсу и поддерживали его в актуальном состоянии.



Этот опыт высветил явление, слишком распространенное в индустрии программного обеспечения: повторение ошибок прошлого снова и снова (Brössler, 2000; Glass, 2003). (В этом месте книги некоторые авторы вставили бы знаменитую цитату философа Джорджа Сантаяны (George Santayana): «Тот, кто не помнит своего прошлого, обречен пережить его вновь» — но я не буду этого делать.) У нас имеются бесчисленные коллекции лучших приемов в области разработки программного обеспечения и сборники уроков, извлеченных из практики, включая этот. Существует огромное количество книг по разным аспектам разработки программного обеспечения и управления проектами. И все же многие проекты продолжают сталкиваться с проблемами, поскольку не практикуют мероприятия, способствующие успеху.

Преимущества обучения

Группа Standish Group публикует свой отчет CHAOS каждые несколько лет, начиная с 1994 года. (CHAOS — это не описание, а остроумно составленная аббревиатура от Comprehensive Human Appraisal for Origining Software¹.) Отчеты CHAOS, основанные на данных тысяч проектов, показывают процент проектов полностью успешных, столкнувшихся с трудностями или потерпевших неудачу. Успех определяется как комбинация из трех показателей: завершение в срок, не превышение бюджета и удовлетворение потребностей клиентов и пользователей (The Standish Group, 2015). Цифры, представленные в отчете, менялись с годами, но доля полностью успешных проектов по-прежнему не превышает 40 %. Но самое обескураживающее, пожалуй, состоит в том, что факторы, приводящие к проблемам и неудачам, почти не меняются.



Наблюдение за закономерностями в результатах ведет к новым парадигмам выполнения работы. Например, аналитический паралич и устаревающие требования в долгосрочных проектах помогли мотивировать стремление к поэтапной разработке. Некоторые данные в отчетах CHAOS показывают, что проекты, практикующие Agile, имеют более высокий процент успеха, чем водопадные проекты (The Standish Group, 2015).

Между программистом-любителем и опытным инженером-программистом лежит огромная пропасть.

Разработка программного обеспечения отличается от других технических областей тем, что вы можете выполнять полезную работу, имея минимальное формальное образование и опыт, по крайней мере, до определенного момента. Никто не будет просить врача-любителя удалить ему аппендицис, но многие программисты-любители имеют достаточный объем знаний, чтобы писать небольшие приложения. Однако между программистом-любителем и опытным инженером-программистом, способным работать над большими и сложными проектами в сотрудничестве с другими, лежит огромная пропасть.

¹ Комплексная оценка персонала при разработке программного обеспечения. — Примеч. пер.

Ныне многие молодые специалисты приходят в отрасль через академическую программу в области информатики, разработки программного обеспечения или смежных областей. Однако каждый специалист по разработке ПО, имеющий формальное образование или обучавшийся самостоятельно, должен продолжать поглощать постоянно растущий объем знаний и учиться эффективно их применять. В сфере ПО многие аспекты меняются очень быстро. Чтобы идти в ногу с современными технологиями, порой приходится бежать. К счастью, информация, подобная той, которую я собрал здесь, долго остается актуальной.

Каждая организация, занимающаяся разработкой программного обеспечения, накапливает неформальную коллекцию локальных знаний, основанных на опыте. Вместо того чтобы рассказывать байки у костра, как это принято в устной фольклорной традиции, я предлагаю записывать полученные на горьком опыте знания в сборник извлеченных уроков (Wieggers, 2007). Предусмотрительные руководители проектов и разработчики с удовольствием обратятся к нему, приступая к новому начинанию.

Преимущества рассуждения

В начале своей карьеры, работая ученым-исследователем, я был знаком с коллегой по имени Аманда. Когда она начинала новый исследовательский проект, ее иногда можно было увидеть откинувшись на спинку офисного кресла и уставившейся в потолок. Она думала. Аманда усердно изучала внутреннюю техническую литературу компании, содержащую результаты подобных проектов. Она старалась узнать все о предметной области, эффективных и неэффективных подходах, применявшихся в прошлом, и только потом приступала к своим экспериментам. Они были трудоемкими и дорогостоящими, поэтому Аманда стремилась всегда двигаться в верном направлении. Изучение материалов предыдущих исследований сделало ее высокоэффективным ученым. Наблюдение за Амандой подтвердило, что важно извлекать уроки из истории, как отраслевой, так и местной, прежде чем с головой погрузиться в новый проект.



Эта глава посвящена совершенствованию процессов разработки программного обеспечения. Отдельные лица, проектные группы и организации должны постоянно совершенствовать свои навыки разработки и управления. Первоначальная цель пятиуровневых моделей зрелости процессов состояла в том, чтобы определить процессы, благодаря которым любая команда сможет успешно завершить любой проект.

Независимо от конкретной методологии или платформы, выбранной для структурирования усилий по совершенствованию, каждая организация, занимающаяся разработкой программного обеспечения, так или иначе стремится постоянно добиваться успеха.



СЛЕДУЮЩИЕ ШАГИ: СОВЕРШЕНСТВОВАНИЕ ПРОЦЕССА РАЗРАБОТКИ

1. Определите, какие уроки, описанные в этой главе, имеют отношение к вашему опыту совершенствования процессов разработки.
2. Можете ли вы, опираясь на свой опыт, вспомнить какие-либо другие уроки, связанные с совершенствованием процессов, которыми стоит поделиться с коллегами?
3. Вспомните три самые болезненные для вас точки и проанализируйте их первопричины, чтобы выявить факторы, на которые можно направить усилия на первом этапе совершенствования процессов.
4. Перечислите описанные в этой главе методы, способные помочь в решении связанных с совершенствованием процессов проблем, которые мы определили во врезке «Первые шаги» в начале главы. Как каждый метод может повысить эффективность и результативность вашей команды?
5. Как бы вы определили, приносит ли желаемые результаты каждый метод, озвученный на шаге 4? Насколько ценные для вас эти результаты?
6. Определите любые препятствия, которые могут затруднить применение методов, перечисленных на шаге 4. Как бы вы справились с ними? Заручились бы поддержкой коллег, готовых помочь вам в реализации этих методов?
7. Какие тренинги, книги, руководства или другие ресурсы могли бы помочь вашей организации успешнее проводить мероприятия по совершенствованию процессов?

Глава 8

Что дальше?

К настоящему моменту мы рассмотрели довольно много материала. Я поделился с вами 59 уроками разработки программного обеспечения и управления проектами, которые накопил за свою 50-летнюю карьеру. Эти уроки делятся на шесть областей: требования, проектирование, управление проектами, культура и командная работа, качество и совершенствование процессов. Безусловно, есть и другие жизненно важные аспекты разработки программного обеспечения, которые я не затронул, например кодирование, тестирование и управление конфигурацией. Теперь я хочу задать вам единственный вопрос:

Что вы собираетесь изменить в своей работе, прочитав эту книгу?

Если вы прорабатывали врезки «Первые шаги» и «Следующие шаги», имеющиеся в каждой главе, то у вас должен накопиться список проблем в этих шести областях, характерных для вашей команды или организации. Возможно, вы проанализировали каждую проблему, чтобы понять ее основные причины и следствия. Возможно, вы определили, какие преимущества может дать решение этих проблем. Я надеюсь, что вы записали для себя много идей и методов, которые могли бы помочь решить эти проблемы и повысить ценность ваших проектов для клиентов.

Если вы не начнете делать что-то по-другому после обучения, то ваши инвестиции в это обучение не принесут вам никакой пользы. Я хочу, чтобы ваше время, потраченное на чтение этой книги, в конечном счете окупилось. Для этого вы должны решить, что делать дальше, а чтобы упростить вам задачу, расскажу еще об одном уроке.



Урок 60 Невозможно изменить все сразу



Независимо от того, сколько болевых точек, идей по улучшению или желательных направлений вы определите, скорость восприятия изменений людьми и организациями ограничена. Я видел в работе проектную группу из 20 высокомотивированных человек, которые одновременно проводили семь мероприятий по улучшению. Приоритеты были туманными, а ресурсы — рассредоточенными. Несмотря на весь энтузиазм, эта команда немного додилась на пути к переменам. Кроме того, изменения слишком много аспектов одновременно, вы не сможете определить, какое именно изменение привело к наблюдаемому результату.



Люди не могут совершенствовать свою работу быстрее, чем позволяют их индивидуальные способности, поэтому инициатива крупномасштабных изменений, навязанная руководством, может ошеломить тех, кого она касается. Попытка изменить слишком многое и сразу может затруднить выполнение проектной работы, поскольку люди будут пытаться понять новые веяния и стараться соответствовать им. Управлять скоростью изменений лучше с помощью инициативы по улучшению снизу вверх. Усилия на нижнем уровне могут дать хороший результат на уровне команды, но имеют потолок, выше которого трудно влиять на группы без помощи руководства. Мэри Линн Маннс (Mary Lynn Manns) и Линда Райзинг (Linda Rising) в своей книге *Fearless Change* (Manns, Rising, 2005) объясняют:

Акцент в восходящих изменениях делается на вовлечении людей и их информировании о происходящем, поэтому неопределенность и сопротивление сводятся к минимуму. Основываясь на нашем опыте, мы считаем, что изменения лучше всего проводить снизу вверх при поддержке со стороны руководства — как на местном, так и на более высоком уровне.

По аналогии с личностным ростом, совершенствование процессов разработки программного обеспечения — это путь, а не пункт назначения. Это циклическое, а не прямолинейное движение. Изобразить цикл изменений можно множеством способов. Самой известной из них является, пожалуй, модель «Планируй — делай — проверяй — действуй», предложенная пионером улучшения качества доктором У. Эдвардсом Демингом (Dr. W. Edwards Deming) и также известная как цикл Шу-

харта (Praxis, 2019; American Society for Quality, 2021d). На рис. 8.1 показан аналогичный цикл улучшения, который я предпочитаю циклу Деминга.

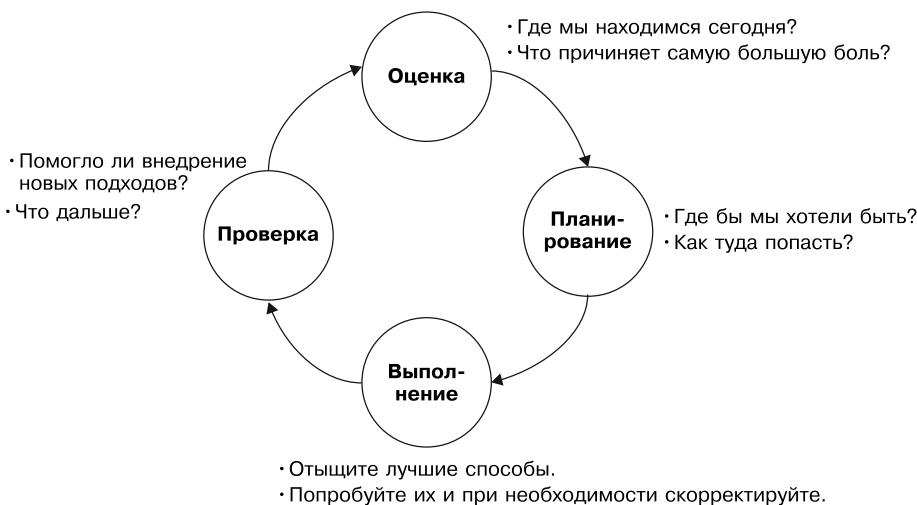


Рис. 8.1. Типичный цикл совершенствования процессов состоит из четырех этапов: оценка, планирование, выполнение и проверка

Шаг 1. Оценка. Определите, где вы находитесь сегодня: каких результатов достигли ваши проекты в настоящее время и насколько хорошо идут дела. Определите самые проблемные области и самые большие возможности для улучшения. Во врезках «Первые шаги» в предыдущих шести главах как раз описывается неформальная деятельность по оценке.

Шаг 2. Планирование. Решите, где вы хотели бы быть в будущем: каковы ваши цели по улучшению бизнеса и технических аспектов. Наметьте план, как этого достичь. Во врезках «Следующие шаги» указаны отправные точки, которые помогут вам наметить свой курс к более желанному будущему.

Шаг 3. Выполнение. Теперь самое сложное: начать работать как-то иначе. Вам придется познакомиться с практиками и методами, которые могут дать лучшие результаты, опробовать их путем привлечения команды энтузиастов и скорректировать под условия своей среды. Сохраняйте то, что работает, и изменяйте, заменяйте или отказывайтесь от того, что не работает.

Шаг 4. Проверка. Позвольте новым методам закрепиться, а затем проверьте, дают ли они ожидаемые результаты. Чтобы изменить направление, требуются время и терпение. Затраты на обучение неизбежны. Улучшение показателей бизнеса — это долгосрочный, но запаздывающий признак пригодности новых подходов для ваших проектов. Попробуйте определить промежуточные показатели, по которым можно понять, окупается ли то, что вы пытаетесь сделать.



Помните, что изменение — это циклическая работа. У вас всегда найдется над чем поработать. Возвращайтесь к шагу оценки в конце каждого цикла изменений, чтобы изучить новую ситуацию, а затем начните новый.

Совершенствование процессов разработки программного обеспечения — это путь, а не пункт назначения. Это циклическое, а не прямолинейное движение.

Приоритизация изменений

Фокус — ключевое слово в любой инициативе по совершенствованию. Вы можете определить больше желаемых изменений, чем могли бы внедрить, поэтому вам придется расставить приоритеты, чтобы сосредоточить всю свою энергию на получении максимальной выгоды. Затем вы должны выкроить время для реализации каждого изменения.

Выберите самые актуальные проблемы и начните работать над ними завтра.

Никогда не бывает подходящего времени для выполнения мероприятий по улучшению качества. Если вы намеренно не выделяете время, то никогда не найдете свободной минутки, не занятой другими хлопотами. Но если вы не сделаете первых шагов к изменениям, то не добьетесь прогресса. Каждый должен потратить часть своего времени на изучение способов повышения собственной продуктивности. Выберите самые



актуальные проблемы и начните работать над ними завтра. Я серьезно: завтра!

Пересмотрите свои заметки, сделанные при чтении врезок «Первые шаги» и «Следующие шаги» в каждой главе этой книги. В какой области было бы наиболее желательно увидеть некоторые улучшения и уменьшить неприятные последствия? Какие изменения могли бы быть наиболее ценными для бизнеса вашей организации? Какие изменения кажутся наиболее достижимыми при разумном вложении энергии и денег?

Как только вы определите основные проблемные области, выберите возможные решения, которые окупятся с высокой вероятностью. Первоначальные идеи вы найдете в десятках практик, описанных в этой книге. Я предоставил множество ссылок на другие источники, в которых можно узнать больше информации о практиках, кажущихся многообещающими. Расставляя приоритеты в своих действиях по их важности, срочности и стоимости, начните с изменений, которые можно внедрить быстро или которые могут оказать наибольшее влияние в ближайшее время. Ищите плоды на нижних ветках — быстродостижимые улучшения — и празднуйте эти успехи. Когда члены команды видят, что добиваются прогресса и начинают пожинать плоды, это мотивирует их и доказывает, что изменения возможны.



Проверка реальности

Обдумывая действия, которые, по вашему мнению, дадут лучшие результаты, помните о философии, которую я принял как консультант. Прежде чем дать совет клиенту, я провожу небольшую мысленную проверку, чтобы убедиться, что предлагаемое мною действие удовлетворяет двум критериям.

1. Высокая вероятность решения проблемы клиента.
2. Практичность и достижимость в среде клиента.



Эта проверка реальности гарантирует, что мой совет будет полезен и я не предложу ничего, что не подходит для проектов или культуры клиента. Применяйте этот фильтр к любым изменениям, которые рекомендуете самому себе или своей организации.

Подумайте, какие изменения организация готова принять, а какие следует отложить, пока не сформируется подходящая культурная или



техническая среда. В главе 5 были представлены некоторые характеристики здоровой и не очень здоровой культуры разработки программного обеспечения. Поскольку изменение культуры занимает больше времени, чем внедрение новых технических практик, руководители с самого начала должны направлять развитие культуры в нужном направлении. Это заложит основу, опираясь на которую команды смогут справиться с более крупными изменениями.



Я предлагаю применять системный подход даже на личном уровне. Начиная новый проект, я всегда определял две области разработки или управления проектами, в которых хотел бы совершенствоваться. Это может быть модульное тестирование, оценка, проектирование алгоритмов, рецензирование, управление сборкой или что-то еще. Работая над каждым проектом, я выделял часть времени на чтение литературы по выбранным мною темам. Я мог пройти курс обучения или посетить конференцию. Я искал возможности применить то, чему научился, и понимал, что потребуются некоторые усилия, чтобы выяснить, как получить наибольший эффект от изменений. Точно так же я поступаю и сейчас со своим хобби записи песен. Приступая к каждой песне, я стараюсь узнать больше об игре на гитаре, технике записи и продюсировании, а также об огромном наборе функций моего программного обеспечения для записи. Благодаря этому каждая следующая моя песня немного лучше предыдущей.



Вспомните, как в уроке 55 «У вас нет времени, чтобы совершить все ошибки, сделанные до вас» говорилось, что кривая обучения не является плавным и непрерывным переходом. У вас обязательно будут взлеты и падения. Но вы доберетесь туда, куда хотите попасть.

ПЛАНИРОВАНИЕ ДЕЙСТВИЙ

Изменения не происходят сами по себе. Расстановки приоритетов проблем и выбора действий по улучшению недостаточно. Если вы серьезно настроены на изменения, то должны относиться к ним как к проекту, в отношении которого есть цели, планы, ресурсы, отслеживание статуса и ответственность. В зависимости от масштаба изменения вы можете создавать планы действий для себя, проектной группы или организации в целом. Любой человек, отвечающий за инициативу по совершенствованию в масштабах всей организации, найдет полезной книгу *Making Process Improvement Work* (Potter, Sakry, 2002). Более про-



стые, но все же структурированные подходы хорошо работают на индивидуальном и командном уровнях.

В табл. 8.1 показан простой шаблон планирования действий. Загляните в будущее всего на неделю, на месяц и на полгода. Перечислите новые технические или управленческие практики, которые хотите попробовать в каждом из этих периодов. Опишите ситуацию, к которой вы могли бы применить их, и преимущества, которые вы ищете. Возможно, вам потребуются дополнительные ресурсы, которые помогут научиться применять эти новые методы, такие как обучение, инструменты, книги или консультационная помощь. Одни методы могут быть применимы только к вашей личной работе, а другие — затрагивать несколько сообществ, поэтому подумайте, сотрудничество с кем вам потребуется в каждом случае. Кроме того, выясните, что вам может помешать успешно внедрять новинки. Затем попытайтесь найти союзников, которые помогут разрушить эти препятствия.

Таблица 8.1. Простой шаблон, помогающий планировать действия по совершенствованию на ближайшую, среднесрочную и долгосрочную перспективу

Параметр	Следующая неделя	Следующий месяц	Следующие шесть месяцев
Новые практики, которые можно попробовать			
Ситуация, к которой можно их применить			
Преимущества, которые они могут дать			
Помощь или дополнительная информация, которые могут понадобиться			
Сотрудничество с кем может помочь			
Препятствия, которые могут помешать добиться успеха			
Кто может помочь разрушить их			

Планирование значительно увеличивает шансы на успешное внедрение изменений. Однако есть ловушка, которой следует осторегаться. На рис. 8.1 (см. выше) был представлен такой цикл изменений: оценка, планирование, выполнение, проверка. Я видел много организаций, которые тщательно оценивают текущую реальность, нацелены на улучшение в будущем и планируют, как этого добиться, только для того, чтобы остановиться на этапе выполнения. Это самая трудная часть — заставить себя отвлечься от работы над проектом, чтобы реализовать пункты плана и довести их до завершения. Независимо от важности намерений планы по улучшению, которые не превращаются в действия, бесполезны. Поэтому осторегайтесь соблазна пойти по легкому пути и продолжать работать так, как вы и члены вашей команды работали всегда. Это не приведет вас к желаемым результатам.

ВАШИ СОБСТВЕННЫЕ УРОКИ

Каждый опытный разработчик программного обеспечения накопил ряд уроков, полученных из его опыта. Эта книга содержит уроки, которые получил я, и уверен, что, помимо них, у вас есть собственные. Подумайте, как ваша команда могла бы объединить свои усилия и ключевые идеи с уроками, изложенными в данной книге. Подумайте, как поделиться этими уроками со всеми в своей организации, чтобы ускорить процесс обучения для всех. На следующем шаге вы можете выбрать из собранных уроков те практики, которые помогут вашей команде получить лучшие результаты.

Построение обучающейся и совершенствующейся организации начинается с людей, которые усвоили ценность извлечения жемчужин мудрости из своего опыта и обмена ими с другими. Я надеюсь, что вы воспользуетесь этой возможностью и воплотите в жизнь отдельные фрагменты того, что прочитали в данной книге. Пригласите своих товарищей по команде присоединиться к вам. Это может оказаться увлекательным занятием.

Приложение

Краткий перечень уроков

Требования

1. Если вы неверно определили требования, то неважно, насколько хорошо вы выполните остальную часть работы.
2. Основной результат разработки требований — общее видение и понимание.
3. Интересы всех сторон нигде не пересекаются так явственно, как в требованиях.
4. В требованиях в первую очередь важны особенности использования, а затем — функциональность.
5. Разработка требований — итеративный процесс.
6. Agile-требования не отличаются от других.
7. Запись знаний обходится дешевле, чем повторное ихобретение.
8. Главное требование к разработке — наложенное и эффективное общение.
9. Качественность требований каждый определяет по-своему.
10. Требования должны быть достаточно хорошими, чтобы разработка могла продолжаться с приемлемым уровнем риска.
11. Люди не просто так собирают требования.
12. Выявление требований должно помочь разработчикам услышать голос клиента.
13. Две распространенные практики выявления требований — телепатия и ясновидение. Но они не работают.

14. Большая группа людей не способна организованно покинуть горящую комнату, не говоря уже о том, чтобы сформулировать какое-то требование.
15. Когда принимаете решение о добавлении функций, избегайте расстановки приоритетов по децибелам.
16. Не задокументировав и не согласовав содержимое проекта, нельзя узнать, увеличивается ли его объем.

Проектирование

17. Проектирование — итеративный процесс.
18. Чем выше уровень абстракции, тем проще выполнять итерации.
19. Разрабатывайте продукты так, чтобы их легко было использовать правильно и трудно — неправильно.
20. Невозможно оптимизировать все желаемые атрибуты качества.
21. Проблемы легче предупредить, чем исправить.
22. Проблемы многих систем скрываются в интерфейсах.

Управление проектами

23. При планировании работ нужно учитывать разногласия.
24. Не давайте оценок наугад.
25. Айсберги всегда больше, чем кажутся.
26. Ваши переговорные позиции будут сильнее при наличии обосновывающих данных.
27. Не записывая оценки и не сверяя их с тем, что произошло на самом деле, вы всегда будете строить догадки, а не оценивать.
28. Не меняйте оценку в зависимости от того, что хочет услышать получатель.
29. Держитесь подальше от критического пути.
30. Задание либо полностью выполнено, либо не выполнено: частичное выполнение не засчитывается.
31. Команде проекта нужна гибкость в отношении хотя бы одного из пяти измерений: масштаба, плана, бюджета, персонала и качества.
32. Если вы не контролируете риски своего проекта, то они будут контролировать вас.

33. Клиент не всегда прав.
34. Мы слишком часто принимаем желаемое за действительное.

Культура и командная работа

35. Передача знаний не ведет к проигрышу.
36. Как бы сильно на вас ни давили, не берите на себя обязательства, которые не сможете выполнить.
37. Не ждите, что без обучения и освоения передовых практик производительность повысится как по волшебству.
38. Люди много говорят о своих правах, но права подразумевают ответственность.
39. Даже небольшие физические расстояния препятствуют общению и совместной работе.
40. Неформальные подходы, используемые небольшими сплоченными командами, плохо масштабируются.
41. Не стоит недооценивать сложность изменения культуры организации по мере перехода к новым методам работы.
42. Никакие инженерные или управленческие приемы не дадут эффекта, если вы имеете дело с неразумными людьми.

Качество

43. Решая вопрос о качестве программного обеспечения, вы можете выбирать: платить немало сейчас или позже, но еще больше.
44. Высокое качество естественным образом ведет к повышению производительности.
45. У организаций никогда нет времени, чтобы правильно создать программное обеспечение, но они находят ресурсы, чтобы исправить его позже.
46. Остерегайтесь малозаметных разрывов между плохим и хорошим.
47. Никогда не поддавайтесь уговорам руководителя или клиента сделать работу наспех.
48. Стремитесь к тому, чтобы дефект нашли коллеги, а не покупатели.
49. Разработчики программного обеспечения любят инструменты, но дурак с инструментами — это вооруженный дурак.

50. Сегодняшний проект, требующий немедленной реализации, завтра может превратиться в кошмар для службы сопровождения.

Совершенствование процессов

51. Остерегайтесь «менеджмента по Businessweek».
52. Не спрашивайте: «Что это даст мне?» Спрашивайте: «Что это даст нам?»
53. Боль — лучшая мотивация для изменения методов работы.
54. Внедряя новые методы работы, делайте это мягко, но непрерывно.
55. У вас нет времени, чтобы совершить все ошибки, сделанные до вас.
56. Здравый смысл и опыт иногда важнее определенного процесса.
57. Адаптируйте готовые шаблоны документов.
58. Если не тратить время на учебу и совершенствование, то не стоит ждать, что следующий проект будет реализован лучше предыдущего.
59. Самая удручающая закономерность в индустрии программного обеспечения — повторение одних и тех же неэффективных действий снова и снова.

Универсальный урок

60. Невозможно изменить все сразу.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- Achimugu P., Selamat A., Ibrahim R., Mahrin M. N. 2014. A systematic literature review of software requirements prioritization research // Information and Software Technology. № 56 (6). P. 568–585.
- ACM. 1999. The Software Engineering Code of Ethics and Professional Practice. Association for Computing Machinery and IEEE Computer Society. <https://ethics.acm.org/code-of-ethics/software-engineering-code>.
- Ageling W.-J. 2020. Here's Why Many Developers Hate Scrum. <https://medium.com/serious-scrum/here-is-why-many-developers-hate-scrum-3a43baa015d1>.
- Agile Alliance. 2021a. Definition of Done. <https://www.agilealliance.org/glossary/definition-of-done>.
- Agile Alliance. 2021b. Acceptance Testing. <https://www.agilealliance.org/glossary/acceptance>.
- Agile Alliance. 2021c. 12 Principles Behind the Agile Manifesto. <https://www.agile-alliance.org/agile101/12-principles-behind-the-agile-manifesto>.
- Agile Alliance, Eckstein J., Buck J. 2021. Changing the Culture by Changing Habits. <https://www.agilealliance.org/changing-the-culture-by-changing-habits>.
- Aleshire P. 2004. *Eye of the Viper: The Making of an F-16 Pilot*. Guilford, CT: Lyons Press.
- Amblor S. W. 2005. *The Elements of UML™ 2.0 Style*. New York: Cambridge University Press.
- Amblor S. W. 2006. Why Agile Software Development Techniques Work: Improved Feedback. <https://www.ambysoft.com/essays/whyAgileWorksFeedback.html>.
- American Society for Quality. 2021a. Quality Glossary — Q. <https://asq.org/quality-resources/quality-glossary/q>.
- American Society for Quality. 2021b. Joseph M. Juran. <https://asq.org/about-asq/honorary-members/juran>.
- American Society for Quality. 2021c. Cost of Quality (COQ). <https://asq.org/quality-resources/cost-of-quality>.

- American Society for Quality.* 2021d. What Is the Plan-Do-Check-Act (PDCA) Cycle? <https://asq.org/quality-resources/pdca-cycle>.
- Atwood J.* 2006. The Programmer's Bill of Rights. <https://blog.codinghorror.com/the-programmers-bill-of-rights>.
- Bakker J.* 2020. Top lessons learned from working with a 10x developer. <https://levelup.gitconnected.com/top-lessons-learned-from-working-with-a-10x-developer-51de12383e25>.
- Barlas S.* 1996. Anatomy of a Runaway: What Grounded the AAS // IEEE Software. № 13 (1). P. 104–106.
- Beatty J., Chen A.* 2012. Visual Models for Software Requirements. Redmond, WA: Microsoft Press.
- Beck K. et al.* 2001. Manifesto for Agile Software Development. <https://agilemanifesto.org/>¹.
- Beck K.* 2003. Test-Driven Development: By Example. Boston: Addison-Wesley².
- Beck K., Andres C.* 2005. Extreme Programming Explained: Embrace Change. 2nd Ed. Boston: Addison-Wesley³.
- Bentley J.* 2000. Programming Pearls. 2nd Ed. Boston: Addison-Wesley.
- Booch G., Rumbaugh J., Jacobson I.* 1999. The Unified Modeling Language User Guide. Reading, MA: Addison-Wesley⁴.
- Bright Hub PM.* 2009. Various Kinds of Risks Associated with Software Project Management. <https://www.brighthubpm.com/risk-management/47932-risks-involved-in-software-project-management>.
- Briski K. A., Chitale P., Hamilton V., Pratt A., Starr B., Veroulis J., Villard B.* 2008. Minimizing code defects to improve software quality and lower development costs. IBM Development Solutions Whitepaper. <ftp://ftp.software.ibm.com/software/rational/info/do-more/RAW14109USEN.pdf>.
- Brooks Jr., Frederick P.* 1995. The Mythical Man-Month: Essays on Software Engineering. Anniversary Edition. Reading, MA: Addison-Wesley.
- Brosseau J.* 2008. Software Teamwork: Taking Ownership for Success. Boston: Addison-Wesley.
- Brößler P.* 2000. Knowledge management at a software house: An experience report // Ruhe G., Bomarius F. (eds) Learning Software Organizations. SEKE 1999. Lecture Notes in Computer Science. Vol. 1756. Berlin, Heidelberg: Springer. <https://doi.org/10.1007/BFb0101419>.

¹ Сайт на русском языке: <https://agilemanifesto.org/iso/ru/manifesto.html>. — Приимеч. пер.

² Кент Б. Экстремальное программирование: разработка через тестирование. — СПб.: Питер, 2022.

³ Кент Б. Экстремальное программирование. — СПб.: Питер, 2017.

⁴ Буц Г., Якобсон И., Рамбо Дж. Язык UML. Руководство пользователя.

- Charette R. N.* 1996. Large-Scale Project Management Is Risk Management // IEEE Software. № 13 (4). P. 110–117.
- Charette R. N.* 2005. Why Software Fails // IEEE Spectrum. № 42 (9). P. 42–49. <https://spectrum.ieee.org/computing/software/why-software-fails>.
- Chrassis M. B., Konrad M., Shrum S.* 2003. CMMI: Guidelines for Process Integration and Product Improvement. Boston: Addison-Wesley.
- CMMI Institute.* 2017. Published Appraisal Results. <https://cmmiinstitute.com/pars>.
- Cohen E.* 2018. How to Use the Critical Path Method for Complete Beginners. <https://www.workamajig.com/blog/critical-path-method>.
- Cohn M.* 2004. User Stories Applied: For Agile Software Development. Boston: Addison-Wesley¹.
- Cohn M.* 2006. Agile Estimating and Planning. Boston: Addison-Wesley².
- Cohn M.* 2010. Succeeding with Agile: Software Development Using Scrum. Boston: Addison-Wesley³.
- Cohn M.* 2014. Know Exactly What Velocity Means to Your Scrum Team. <https://www.mountaingoatsoftware.com/blog/know-exactly-what-velocity-means-to-your-scrum-team>.
- Coleman B., Goodwin D.* 2017. Designing UX: Prototyping. Collingwood, VIC, Australia: SitePoint Pty. Ltd.
- Colorado State University.* n. d. Survey Research. <https://writing.colostate.edu/guides/guide.cfm?guideid=68>.
- Constantine L. L., Lockwood L. A.D.* 1999. Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. Reading, MA: Addison-Wesley.
- Construx.* 2010. Individual Productivity Variation in Software Development. <https://www.construx.com/blog/productivity-variations-among-software-developers-and-teams-the-origin-of-10x>.
- Cooper A., Reimann R., Cronin D., Noessel C.* 2014. About Face: The Essentials of Interaction Design. 4th Ed. Indianapolis: John Wiley & Sons, Inc.
- Costello K.* 2019. The Secret to DevOps Success. Gartner. <https://www.gartner.com/smarterwithgartner/the-secret-to-devops-success>.
- Covey S. R.* 2020. The 7 Habits of Highly Effective People. 25th Anniversary Edition. New York: Simon & Schuster⁴.

¹ Кон М. Пользовательские истории: гибкая разработка программного обеспечения.

² Кон М. Agile. Оценка и планирование проектов.

³ Кон М. Scrum: гибкая разработка ПО.

⁴ Кови С. Семь навыков высокоеффективных людей. Мощные инструменты развития личности.

- Crosby P. B.* 1979. *Quality Is Free: The Art of Making Quality Certain*. New York: McGraw-Hill.
- Cunningham W.* 1992. *The WyCash Portfolio Management System. OOPSLA '92 Experience Report*. <http://c2.com/doc/oopsla92.html>.
- Datt P.* 2020a. Difference Between Product Owner & Business Analyst Role. <https://premieragile.com/difference-between-product-owner-and-business-analyst>.
- Datt P.* 2020b. What is the Definition of Done (DoD) in Agile? <https://premieragile.com/definition-of-done-in-agile>.
- Davis A. M.* 1995. *201 Principles of Software Development*. New York: McGraw-Hill.
- Davis A. M.* 2005. *Just Enough Requirements Management: Where Software Development Meets Marketing*. New York: Dorset House Publishing.
- DeMarco T.* 1979. *Structured Analysis and System Specification*. Upper Saddle River, NJ: Yourdon Press.
- DeMarco T.* 2001. *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency*. New York: Broadway Books.
- DeMarco T., Lister T.* 2003. *Waltzing with Bears: Managing Risk on Software Projects*. New York: Dorset House Publishing.
- DeMarco T., Lister T.* 2013. *Peopleware: Productive Projects and Teams*. 3rd Ed. Boston: Addison-Wesley¹.
- Derby E., Larsen D.* 2006. *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf².
- DOT.* 1998. *Audit Report: Advance Automation System, Federal Aviation Administration Report Number AV-1998-113*. Washington, DC: Office of Inspector General, U.S. Department of Transportation.
- Feldmann C. G.* 1998. *The Practical Guide to Business Process Reengineering Using IDEF0*. New York: Dorset House Publishing.
- Fisher R., Ury W., Patton B.* 2011. *Getting to Yes: Negotiating Agreement Without Giving In*. 3rd Ed. New York: Penguin Books³.
- Foord M.* 2017. 30 best practices for software development and testing. <https://opensource.com/article/17/5/30-best-practices-software-development-and-testing>.
- Gamma E., Helm R., Johnson R., Vlissides J.* 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley⁴.

¹ Демарко Т., Листер Т. Человеческий фактор. Успешные проекты и команды.

² Дерби Э., Ларсен Д. Agile-ретроспектива. Как превратить хорошую команду в великую.

³ Фишер Р., Паттон Б., Юри У. Гарвардский метод переговоров. Как всегда добиваться своего.

⁴ Гамма Э., Джонсон Р., Хелм Р. Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2021.

- Gilb T. 2005. Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage. Oxford, England: Elsevier Butterworth-Heinemann.
- Gilb T., Graham D. 1993. Software Inspection. Reading, MA: Addison-Wesley.
- Glass R. L. 2003. Facts and Fallacies of Software Engineering. Boston: Addison-Wesley¹.
- Goldratt E. M. 1997. Critical Chain. Great Barrington, MA: The North River Press.
- Gottesdiener E. 2002. Requirements by Collaboration: Workshops for Defining Needs. Boston: Addison-Wesley.
- Grady R. B. 1999. An Economic Release Decision Model: Insights into Software Project Management // Proceedings of the Applications of Software Measurement Conference, 227–239. Orange Park, FL: Software Quality Engineering.
- Grady R. B., Slack T. Van. 1994. Key Lessons in Achieving Widespread Inspection Use // IEEE Software. № 11 (4). P. 46–57.
- Gray M. 2020. Is the Way You Use Burndown Charts Helping or Holding You Back? <https://medium.com/better-programming/the-definitive-guide-to-burn-down-charts-a176db096294>.
- Hasan M. S., Mahmood A. A., Alam Md. J., Hasan Sk. Md. N., Rahman F. 2010. An Evaluation of Software Requirement Prioritization Techniques // International Journal of Computer Science and Information Security. № 8 (9). P. 83–94.
- Haskins B., Stecklein J., Brandon D., Moroney G., Lovell R., Dabney J. 2004. Error Cost Escalation through the Project Life Cycle // Proceedings of the 14th Annual International Symposium of INCOSE, 1723–1737. Toulouse, France. International Council on Systems Engineering.
- Hatch N. 2019. 10 Critical Culture Change Elements in Agile Transformation. https://www.insight.com/en_US/content-and-resources/blog/10-critical-culture-change-elements-in-agile-transformation.html.
- Hilliard A. 2018. A Look at Software Development Culture. <https://www.accelerance.com/blog/software-development-culture>.
- Holland D. 1999. Document Inspection as an Agent of Change // Software Quality Professional. № 2 (1). P. 22–33.
- Hossain Md. S. 2018. Rework and Reuse Effects in Software Economy // Global Journal of Computer Science and Technology (C): Software & Data Engineering. № 18 (4-C). P. 35–50. https://globaljournals.org/GJCST_Volume18/5-Rework-and-Reuse-Effects.pdf.
- IIBA. 2015. A Guide to the Business Analysis Body of Knowledge (BABOK Guide). 3rd Ed. Toronto, ON, Canada: International Institute of Business Analysis.
- ISO/IEC. 2011. ISO/IEC 25010:2011(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System

¹ Гласс Р. Факты и заблуждения профессионального программирования.

- and software quality models. <https://www.iso.org/obp/ui/#iso:std:i-so-iec:25010:ed-1:v1:en>.
- ISO/IEC/IEEE.* 2018. ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering. <https://www.iso.org/standard/72089.html>.
- Jones C.* 1994. Assessment and Control of Software Risks. Upper Saddle River, NJ: Yourdon Press.
- Jones C.* 2006. Social and Technical Reasons for Software Project Failures // Cross-Talk. № 19 (6). P. 4–9. <https://apps.dtic.mil/dtic/tr/fulltext/u2/a487371.pdf>.
- Juran.* 2019. Features of Quality & Definition of Quality Excellence. <https://www.juran.com/blog/features-of-quality-definition-of-quality-excellence>.
- Kaley A.* 2021. Mapping User Stories in Agile. <https://www.nngroup.com/articles/user-story-mapping>.
- Kaner C., Bach J., Pettichord B.* 2002. Lessons Learned in Software Testing: A Context-Driven Approach. New York: John Wiley & Sons.
- Karten N.* 1994. Managing Expectations: Working with People Who Want More, Better, Faster, Sooner, NOW! New York: Dorset House Publishing.
- Kerievsky J.* 2005. Refactoring to Patterns. Boston: Addison-Wesley¹.
- Kerth N. L.* 2001. Project Retrospectives: A Handbook for Team Reviews. New York: Dorset House Publishing².
- Klement A.* 2013. Replacing The User Story With the Job Story. <https://jtbd.info/replacing-the-user-story-with-the-job-story-af7cdee10c27>.
- Koopman P.* 2010. Better Embedded System Software. Pittsburgh: Drummadrochit Press.
- Krasner H.* 2018. The Cost of Poor Quality Software in the US: A 2018 Report. Consortium for IT Software Quality. <https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/The-Cost-of-Poor-Quality-Software-in-the-US-2018-Report.pdf>.
- Kukreja N., Boehm B., Payyavula S. S., Padmanabhuni S.* 2012. Selecting an Appropriate Framework for Value-Based Requirements Prioritization // Proceedings of the 2012 20th IEEE International Requirements Engineering Conference, 303–308. Los Alamitos, CA: IEEE Computer Society Press.
- Kulak D., Guiney E.* 2004. Use Cases: Requirements in Context. 2nd Ed. Boston: Addison-Wesley.
- Larman C.* 2004. Agile and Iterative Development: A Manager’s Guide. Boston: Addison-Wesley.

¹ Керивески Дж. Рефакторинг с использованием шаблонов.

² Керт Н. Ретроспектива проекта: как проектным командам оглядываться назад, чтобы двигаться вперед.

- Leffingwell D.* 2011. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Boston: Addison-Wesley.
- Leonard A.* 2020. Committing to collaboration. <https://increment.com/remote/committing-to-collaboration-version-control>.
- Löwe N.* 2015. Our Responsibility as Software Developers. <https://www.infoq.com/articles/Responsible-Software-Development>.
- Lucidchart.* 2021. How to perform a stakeholder analysis. <https://www.lucidchart.com/blog/how-to-do-a-stakeholder-analysis>.
- MacKay J.* 2021. Context switching: Why jumping between tasks is killing your productivity (and what you can do about it). <https://blog.rescuetime.com/context-switching>.
- Mancuso S.* 2016. Cohesion — The cornerstone of Software Design. <https://www.codurance.com/publications/software-creation/2016/03/03/cohesion-cornerstone-software-design>.
- Manns M. L., Rising L.* 2005. Fearless Change: Patterns for Introducing New Ideas. Boston: Addison-Wesley.
- Marasco J.* 2007. What Is the Cost of a Requirement Error? <https://www.sticky-minds.com/article/what-cost-requirement-error>.
- Mathieson S. A.* 2019. How diversity spurs creativity in software development. <https://www.computerweekly.com/feature/How-diversity-spurs-creativity-in-software-development>.
- McAllister D.* 2017. Software Waste & The Cost of Rework. <https://www.linkedin.com/pulse/software-waste-cost-rework-david-mcallister>.
- McConnell S.* 1996. Rapid Development: Taming Wild Software Schedules. Redmond, WA: Microsoft Press.
- McConnell S.* 2004. Code Complete: A Practical Handbook of Software Construction. 2nd Ed. Redmond, WA: Microsoft Press.
- McConnell S.* 2006. Software Estimation: Demystifying the Black Art. Redmond, WA: Microsoft Press.
- McConnell S.* 2010. Origins of 10X — How Valid is the Underlying Research? <https://www.construx.com/blog/the-origins-of-10x-how-valid-is-the-underlying-research>.
- McGreal D., Jocham R.* 2018. The Professional Product Owner: Leveraging Scrum as a Competitive Advantage. Boston: Addison-Wesley.
- McMenamin S., DeMarco T., Hruschka P., Lister T., Robertson J., Robertson S.* 2021. Happy to Work Here: Understanding and Improving the Culture at Work. New Atlantic.
- McPeak A.* 2017. What's the True Cost of a Software Bug? <https://smartbear.com/blog/software-bug-cost>.
- Merrill C.* 2019. Software Maintenance: Understanding the 4 Main Types. <https://www.zibtek.com/blog/software-maintenance-understanding-the-4-main-types>.

- Microsoft.* 2017. SQL Injection. <https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-injection>¹.
- Miller R. E.* 2009. The Quest for Software Requirements. Milwaukee: MavenMark Books.
- Minott Z.* 2020. How I Outperformed More Experienced Developers as a Junior Developer (and How You Can Too). <https://medium.com/better-programming/how-i-outperformed-more-experienced-developers-as-a-junior-developer-and-how-you-can-too-19bc6206fa68>.
- Moore G. A.* 2014. Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers. 3rd Ed. New York: Harper Business².
- Musa J. D.* 1993. Operational profiles in software-reliability engineering // IEEE Software. № 10 (2). P. 14–32.
- Nagappan R.* 2020a. Moving beyond user story templates. <https://uxdesign.cc/moving-beyond-user-story-templates-79a421c6445c>.
- Nagappan R.* 2020b. The iron triangle and Agile. <https://uxdesign.cc/the-iron-triangle-and-agile-7b66d3c72a51>.
- Nichols B.* 2020. Programmer Moneyball: Challenging the Myth of Individual Programmer Productivity. https://insights.sei.cmu.edu/sei_blog/2020/01/programmer-moneyball-challenging-the-myth-of-individual-programmer-productivity.html.
- Nielsen J.* 2020. 10 Usability Heuristics for User Interface Design. <https://www.nngroup.com/articles/ten-usability-heuristics>.
- NIST.* 2002. Planning Report 02-3. The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards & Technology. <https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf>.
- Nussbaum H.* 2020. Rework is Costing Your Company Millions — It's Time to Cut Back. <https://codeclimate.com/blog/rework-costs-millions>.
- Page-Jones M.* 1988. The Practical Guide to Structured Systems Design. 2nd Ed. Englewood Cliffs, NJ: Prentice Hall.
- Page-Jones M.* 2000. Fundamentals of Object-Oriented Design in UML. Boston: Addison-Wesley.
- Paulk M. C., Weber C. V., Curtis B., Chrassis M. B.* 1995. The Capability Maturity Model: Guidelines for Improving the Software Process. Reading, MA: Addison-Wesley.
- Pearls of Wisdom.* 2014a. PEARL XIX: Effective steps to reduce technical debt: An agile approach. <https://agilepearls.wordpress.com/tag/technical-debt>.

¹ Статья на русском языке: <https://learn.microsoft.com/ru-ru/sql/relational-databases/security/sql-injection?view=sql-server-ver16>. — Примеч. пер.

² Мур Дж. Преодоление пропасти. Как вывести технологический продукт на массовый рынок.

- Pearls of Wisdom.* 2014b. PEARL XXIII: Guidelines for Successful and Effective Retrospectives. <https://agilepearls.wordpress.com/2014/05/23/pearl-xxii-guidelines-for-successful-and-effective-retrospectives>.
- PMI.* n. d. What is Project Management? Project Management Institute. <https://www.pmi.org/about/learn-about-pmi/what-is-project-management>.
- PMI.* 2017. PMI's Pulse of the Profession 2017: 9th Global Project Management Survey. <http://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf>.
- Podeswa H.* 2009. The Business Analyst's Handbook. Boston: Course Technology.
- Podeswa H.* 2021. The Agile Guide to Business Analysis and Planning: From Strategic Plan to Continuous Value Delivery. Boston: Addison-Wesley.
- Potter N., Sakry M.* 2002. Making Process Improvement Work. Boston: Addison-Wesley.
- Praxis.* 2019. Shewhart cycle. <https://www.praxisframework.org/en/library/shewhart-cycle>.
- Pronschinske M.* 2017. Lessons from 7 highly successful software engineering cultures. <https://techbeacon.com/app-dev-testing/lessons-7-highly-successful-software-engineering-cultures>.
- Pugh K.* 2005. Prefactoring: Extreme Abstraction, Extreme Separation, Extreme Readability. Sebastopol, CA: O'Reilly Media, Inc.
- Pugh K.* 2006. Interface Oriented Design: With Patterns. Pragmatic Bookshelf.
- Radice R. A.* 2002. High Quality Low Cost Software Inspections. Andover, MA: Paradoxicon Publishing.
- Resologics.* 2021. Team agreements: A key to high-performing, happy teams. <https://www.resologics.com/resologics-blog/2017/7/12/team-agreements-a-key-to-high-performing-happy-teams>.
- Rettig M.* 1990. Software Teams // Communications of the ACM. № 33 (10). P. 23–27.
- Rice D.* 2016. How to Avoid Brittle Code. <https://www.gocd.org/2016/03/24/how-to-avoid-brittle-code>.
- Robertson S., Robertson J.* 2013. Mastering the Requirements Process: Getting Requirements Right. 3rd Ed. Boston: Addison-Wesley.
- Rothman J.* 1999. How to Use Inch-Pebbles When You Think You Can't. <https://www.jrothman.com/articles/1999/01/how-to-use-inch-pebbles-when-you-think-you-cant>.
- Rothman J.* 2000. What Does It Cost You To Fix A Defect? And Why Should You Care? <https://www.jrothman.com/articles/2000/10/what-does-it-cost-you-to-fix-a-defect-and-why-should-you-care>.
- Rothman J.* 2004. Using Inch-Pebbles to Track Project State. <https://www.jrothman.com/articles/2004/02/using-inch-pebbles-to-track-project-state>.

- Rothman J.* 2012. Management Myth #1: The Myth of 100 % Utilization. <https://www.jrothman.com/articles/2012/01/management-myth-1-the-myth-of-100-utilization>.
- Rozanski N., Woods E.* 2005. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Boston: Addison-Wesley.
- St. Augustine's College.* n. d. Rights and responsibilities of software developers. <https://sddhsc.wordpress.com/hsc/9-1-development-and-impact-of-software-solutions/9-1-1-social-and-ethical-issues/rights-and-responsibilities-of-software-developers>.
- Sanket.* 2019. The exponential cost of fixing bugs. <https://deepsource.io/blog/exponential-cost-of-fixing-bugs>.
- Sas D., Avgeriou P.* 2020. Quality attribute trade-offs in the embedded systems industry: an exploratory case study // Software Quality Journal. № 28 (2). P. 505–534. <https://doi.org/10.1007/s11219-019-09478-x>.
- Scaled Agile.* 2021a. Iteration Planning. <https://www.scaledagileframework.com/iteration-planning>.
- Scaled Agile.* 2021b. Agile Architecture in SAFe. <https://www.scaledagileframework.com/agile-architecture>.
- Scaled Agile.* 2021c. Nonfunctional Requirements. <https://www.scaledagileframework.com/nonfunctional-requirements>.
- Scaled Agile.* 2021d. Iteration Retrospective. <https://www.scaledagileframework.com/iteration-retrospective>.
- Schwaber K., Sutherland J.* 2020. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. <https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>.
- SEI.* 2020. SEI CERT Coding Standards. <https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards>.
- Senycia T.* 2020. 8 Collaboration Tools You Need To Work With Remote Developers. <https://youteam.io/blog/8-tools-you-need-to-work-with-remote-developers>.
- Shore J.* 2010. The Art of Agile Development: The Planning Game. https://www.jamesshore.com/v2/books/aoad1/the_planning_game.
- Simmons E.* 2001. Quantifying Quality Requirements Using Planguage. <https://www.geocities.ws/g/i/gillani/SE%272%20Full%20Lectures/ASE%20-%20%20Planguage%20Quantifying%20Quality%20Requirements.pdf>.
- Sliger M.* 2012. Agile estimation techniques. Paper presented at PMI® Global Congress 2012 — North America, Vancouver, BC, Canada. Newtown Square, PA: Project Management Institute. <https://www.pmi.org/learning/library/agile-project-estimation-techniques-6110>.
- Soni V.* 2020. A Practical Prioritization Approach for Technical Debt. <https://productcoalition.com/a-practical-prioritization-approach-for-technical-debt-f1eb31b8e409>.

- Spolsky J.* 2001. Human Task Switches Considered Harmful. <https://www.joelonsoftware.com/2001/02/12/human-task-switches-considered-harmful>.
- StackExchange.* n. d. The Programmers Bill of Responsibilities. <https://softwarereengineering.stackexchange.com/questions/29177/the-programmers-bill-of-responsibilities>.
- Stretton A.* 2018. Relating causes of project failure to an organizational strategic business framework // PM World Journal. Vol VII. Issue I. January. <https://pmworldlibrary.net/wp-content/uploads/2018/01/pmwj66-Jan2018-Stretton-relating-project-failures-to-strategic-framework-featured-paper.pdf>.
- The American Heritage Dictionary of the English Language.* 2020. <https://www.ahdictionary.com>.
- The Mann Group.* 2019. Gentle Pressure, Relentlessly Applied: Agreement on Approach. https://myemail.constantcontact.com/May-Newsletter---Our-New-Motto--Gentle-Pressure--Relentlessly-Applied.html?soid=1103316067083&aid=e5dv_kWEGOA.
- The Standish Group.* 2014. Exceeding Value. The Standish Group International, Inc. https://www.standishgroup.com/sample_research_files/Exceeding%20Value_Layout.pdf.
- The Standish Group.* 2015. CHAOS Report 2015. The Standish Group International, Inc. https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf.
- Thomas S.* 2008a. Agile Project Scope. <https://itsadeliverything.com/agile-project-scope>.
- Thomas S.* 2008b. Agile Project Planning. <https://itsadeliverything.com/agile-project-tplanning>.
- Tutorials Point.* 2021. Software Design Basics. https://www.tutorialspoint.com/software_engineering/software_design_basics.htm.
- 280 Group.* 2021. Product Manager Roles and Responsibilities. <https://280group.com/what-is-product-management/roles/product-manager>.
- Visual Paradigm.* 2020. Daily Scrum Meeting — A Quick Guide. <https://www.visual-paradigm.com/scrum/daily-scrum-meeting-quick-guide>.
- Walker B., Soule S. A.* 2017. Changing Company Culture Requires a Movement, Not a Mandate. <https://hbr.org/2017/06/changing-company-culture-requires-a-movement-not-a-mandate>.
- Weinberg G.* 2012. Agile and the Definition of Quality. <https://secretsofconsulting.blogspot.com/2012/09/agile-and-definition-of-quality.html>.
- Weinberger M.* 2019. Where Are They Now? What happened to the people in Microsoft's iconic 1978 company photo. <https://www.businessinsider.com/microsoft-1978-photo-2016-10>.
- Wiegers K. E.* 1989. The Laws of Computing // ST-Log. № 31 (May). P. 97–98. https://www.atarimagazines.com/st-log/issue31/097_1_FOOTNOTES_THE_LAWS_OF_COMPUTING.php.

- Wiegers K. E. 1996. Creating a Software Engineering Culture. New York: Dorset House Publishing.
- Wiegers K. E. 1998a. Know Your Enemy: Software Risk Management // Software Development. № 6 (10). P. 38–42.
- Wiegers K. E. 1998b. Improve Your Process with Online ‘Good Practices’ // Software Development. № 6 (12). P. 45–50.
- Wiegers K. E. 2002a. Peer Reviews in Software: A Practical Guide. Boston: Addison-Wesley.
- Wiegers K. E. 2002b. Karl Wiegers on Humanizing Peer Reviews // STQE. № 4 (2). P. 22–28. http://www.processimpact.com/articles/humanizing_reviews.pdf.
- Wiegers K. E. 2003. See You in Court // Software Development. № 11 (1). P. 36–40.
- Wiegers K. E. 2006a. More About Software Requirements: Thorny Issues and Practical Advice. Redmond, WA: Microsoft Press.
- Wiegers K. E. 2006b. Estimation Safety Tips. <https://www.stickyminds.com/article/estimation-safety-tips>.
- Wiegers K. E. 2007. Practical Project Initiation: A Handbook with Tools. Redmond, WA: Microsoft Press.
- Wiegers K. E. 2019a. Why Modeling Is an Essential Business Analysis Technique. <https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/5438/Why-Modeling-Is-an-Essential-Business-Analysis-Technique.aspx>.
- Wiegers K. E. 2019b. Why Is Software Always Ninety Percent Done? <https://medium.com/swlh/why-is-software-always-ninety-percent-done-38e125c8b35c>.
- Wiegers K. E. 2019c. Rethinking the Triple Constraint: Five Project Dimensions. <https://medium.com/swlh/rethinking-the-triple-constraint-five-project-dimensions-b3593c364b11>.
- Wiegers K. E. 2019d. Negotiating Achievable Commitments. <https://medium.com/swlh/negotiating-achievable-commitments-6575b3d73b20>.
- Wiegers K. E. 2019e. Mind the Crap Gap. <https://karlwiegers.medium.com/mind-the-crap-gap-61f314fe9678>.
- Wiegers K. E. 2019f. The Core Question about Building Better Software. <https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/5315/categoryId/35/The-Core-Question-about-Building-Better-Software.aspx>.
- Wiegers K. E. 2019g. Project Retrospectives: Looking Back to Look Ahead. <https://medium.com/swlh/project-retrospectives-looking-back-to-look-ahead-f77ab9d4591c#:~:text=Hold%20a%20retrospective%20whenever%20you,you%20with%20those%20that%20remain>.
- Wiegers K. E. 2021. The Thoughtless Design of Everyday Things. Plantation, FL: J. Ross Publishing.
- Wiegers K., Joy B. 2013. Software Requirements. 3rd Ed. Redmond, WA: Microsoft Press¹.

¹ Вигерс Карл И., Битти Дж. Разработка требований к программному обеспечению.

- Wiegers K., Joy B.* 2016. Agile Requirements: What's the Big Deal? <https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/3573/Agile-Requirements-Whats-the-Big-Deal.aspx>.
- Wikic2.* 2006. Developer Bill of Responsibilities. <https://wiki.c2.com/?DeveloperBillOfResponsibilities>.
- Wikic2.* 2008. Developer Bill Of Rights. <https://wiki.c2.com/?DeveloperBillOfRights>.
- Wikipedia.* 2021a. Design for X. Last modified July 3, 2021. https://en.wikipedia.org/wiki/Design_for_X.
- Wikipedia.* 2021b. Agile software development. Last modified July 8, 2021. https://en.wikipedia.org/wiki/Agile_software_development.
- Winters T., Manshreck T., Wright H.* 2020. Software Engineering at Google: Lessons Learned from Programming Over Time. Sebastopol, CA: O'Reilly Media, Inc¹.
- Wright E.* 2016. The Cover Oregon Debacle. Citizens Against Government Waste. <https://www.cagw.org/thewastewatcher/cover-oregon-debacle>.

¹ Райт Х., Манирек Т., Винтерс Т. Делай как в Google. Разработка программного обеспечения. — СПб.: Питер, 2021.

Карл Вигерс

**Жемчужины разработки.
Чему мы научились за 50 лет создания ПО**

Перевела с английского Л. Киселева

Руководитель дивизиона	<i>Ю. Сергиенко</i>
Ведущий редактор	<i>Н. Гринчик</i>
Литературный редактор	<i>Н. Хлебина</i>
Художественный редактор	<i>В. Мостишан</i>
Корректоры	<i>Л. Галаганова, Т. Никифорова</i>
Верстка	<i>Л. Егорова</i>

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес:
194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 02.2024. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные
профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214,
тел./факс: 208 80 01.

Подписано в печать 17.01.24. Формат 70×100/16. Бумага офсетная. Усл. п. л. 29,670. Тираж 1000. Заказ 0000.

Чед Фаулер

ПРОГРАММИСТ-ФАНАТИК

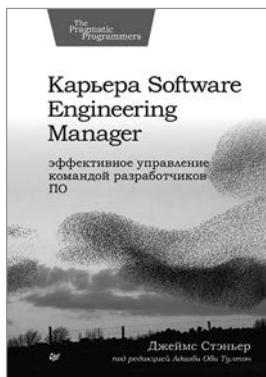


В этой книге вы не найдете описания конкретных технологий, алгоритмов и языков программирования — ценность ее не в этом. Она представляет собой сборник практических советов и рекомендаций, касающихся ситуаций, с которыми порой сталкивается любой разработчик: отсутствие мотивации, выбор приоритетов, психология программирования, отношения с руководством и коллегами и многих других. Подобные знания обычно приходят лишь в результате многолетнего опыта реальной работы. По большому счету перед вами ярко и увлекательно написанное руководство, которое поможет быстро сделать карьеру в индустрии разработки ПО любому, кто поставил себе такую цель. Конечно, опытные программисты могут найти некоторые идеи автора достаточно очевидными, но и для таких найдутся темы, которые позволят пересмотреть устоявшиеся взгляды и выйти на новый уровень мастерства. Для тех же, кто только в самом начале своего пути как разработчика, чтение данной книги, несомненно, откроет широчайшие перспективы.

[КУПИТЬ](#)

Джеймс Стэнье

КАРЬЕРА SOFTWARE ENGINEERING MANAGER. ЭФФЕКТИВНОЕ УПРАВЛЕНИЕ КОМАНДОЙ РАЗРАБОТЧИКОВ ПО



Перед вами неожиданно открылась возможность возглавить команду разработчиков ПО? Пора становиться менеджером? Как решить, подходит ли вам такой шаг в карьере? И если да, то чему нужно научиться, чтобы добиться успеха? С чего начать? Как понять, что вы все делаете правильно? Что вообще означает «менеджмент»? Джеймс Стэнье делится секретами, которые необходимо знать, чтобы успешно управлять командой разработчиков.

Смена статуса с «инженер-разработчик» на «руководитель команды» не должна вас пугать — инженеры могут быть менеджерами, причем идеальными.

Отбросьте болтовню и сосредоточьтесь на практических методах и инструментах. Вы станете эффективным лидером команды, на которого будут равняться ваши сотрудники.

Великие менеджеры могут сделать мир лучше. Присоединяйтесь к нам!

КУПИТЬ