**Гибкие процессы разработки ПО. Переменные проекта: стоимость, качество, сроки, объемы работ, их связь. XP и SCRUM**

**Гибкие методологии**

В течение 1990-х годов все больше разработчиков ПО начинали искать альтернативу традиционным, как правило, основанным на модели водопада, процессам разработки. К 2000 году существовало уже целое множество так называемых легковесных (lightweight) методологий. (Я использую термин «методология», а не «процесс», поскольку гибкие методологии включают в себя множество практик и технологий, выходящих за рамки описания процессов.)

В 2001 году группа создателей и экспертов по различным легковесным методологиям провела семинар, на котором были сформулированы основные принципы гибкой разработки ПО (так называемый Agile Manifesto). На том же семинаре было предложено новое название легковесных методологий – гибкая разработка (agile software development).

Общими особенностями гибких методологий являются:

- Ориентированность на людей – как разработчиков, так и заказчиков. Считается, что умение собрать в проектной команде «правильных» людей определяет успех или неудачу проекта в значительно большей степени, чем любые процессы или технологии.
- Использование устных обсуждений вместо формальных спецификаций везде, где это возможно. Обсуждения должны быть главным способом коммуникации как с заказчиком, так и внутри проектной команды.
- Итеративная разработка с возможно более короткой (в разумных пределах) продолжительностью итерации, при этом в результате каждой итерации выпускается полноценная работающая версия продукта.
- Ожидание изменений – в гибком процессе проектная команда не пытается зафиксировать требования в начале проекта и затем следовать жестко определенному плану. Изменения могут быть сделаны на сколь угодно позднем этапе проекта.

По всей видимости, из методологий гибкой разработки самое широкое распространение получило экстремальное программирование (eXtreme Programming, XP), поэтому именно его мы рассмотрим подробнее.

**XP**

Методология XP была создана Кентом Беком (Kent Beck) в 1996 году в ходе попытки спасти провальный проект по разработке системы расчета зарплаты для компании Крайслер. В 2000 году проект был закрыт, но XP к тому времени уже получила известность и начала распространяться среди разработчиков ПО.

XP наследует все общие принципы гибких методологий, достигая их при помощи двенадцати инженерных практик. Ниже описаны самые интересные из специфических технологий и практик XP:

- В проектной команде должен постоянно работать так называемый представитель заказчика – он обладает детальной информацией о необходимой функциональности, определяет приоритеты отдельных требований, оценивает качество создаваемой системы. Технически, представитель заказчика может быть и сотрудником фирмы разработчика – менеджером продукта, бизнес-аналитиком и т.п.
- Пользовательские истории – короткие неформальные описания прецедентов использования системы. В XP истории являются основным и, вместе с приемочными тестами, единственным средством спецификации требований. Поскольку истории очень лаконичны,

участникам проекта обычно требуются более детальная информация по функциональности системы – они получают ее непосредственно от представителя заказчика.

- Разработка через тестирование (test driven development) – в XP становится особенно важным, чтобы весь создаваемый код был покрыт автоматическими юнит-тестами (почему это так, станет понятно дальше). Этого можно добиться при помощи простого правила – новый код может быть написан исключительно для того, чтобы увеличить число успешно проходящих юнит-тестов. Фактически это означает, что перед реализацией новой функции разработчик должен создать соответствующий тест, а написание кода должно завершиться в тот момент, когда начнет проходить новый, а также все существующие тесты. Если после этого окажется, что новая функция реализована не полностью, необходимо создать еще один тест и повторить весь цикл заново.

- Архитектура системы должна быть максимально простой. XP не рекомендует проектировать в расчете на будущее развитие системы; идеальная архитектура должна не более чем поддерживать существующую функциональность. Цель такого минималистичного подхода к проектированию – избежать бесполезных инвестиций в архитектурные решения, которые часто оказываются выброшенными после очередного изменения требований. Вместо этого архитектура постоянно изменяется и развивается вместе с системой.

- Постоянное изменение архитектуры требует постоянной переработки и улучшения кода – рефакторинга. В XP поощряется коллективное владение кодом – увидев возможность улучшения в любом компоненте системы, разработчик может провести необходимые рефакторинги вне зависимости от того, кто является основным разработчиком компонента. Возможные ошибки, внесенные рефакторингом, должны быть тут же обнаружены автоматическими тестами.

- Все изменения, сделанные разработчиками, после автоматического тестирования практически сразу попадают в основной репозиторий. Таким образом этап интеграции как таковой отсутствует или, что то же самое, происходит постоянно. XP называет эту практику непрерывной интеграцией.

- Парное программирование – наверное, самая противоречивая практика XP. Использование парного программирования не означает, что на двух разработчиков в организации должен быть выделен только один компьютер, однако большая часть написания кода должна проходить в парах, подобно творчеству А. и Б. Стругацких. Считается, что при этом общая эффективность разработки повышается за счет более продуманых решений, меньшего количества ошибок, тщательного написания юнит тестов и т.п.

- Продолжительность рабочей недели не должна превышать 40 часов. По сравнению с обычной практикой постоянных переработок, в средне- и долгосрочной перспективе это повышает производительность проектной команды за счет уменьшения стресса и переутомления.

**Жизненный цикл**

Жизненный цикл проекта в XP состоит из последовательности релизов. Каждый релиз – это полноценная версия продукта, которую может использовать заказчик, и содержащая дополнительную функциональность по сравнению с предыдущим релизом. Релиз появляется в результате одной или нескольких итераций, длящихся от одной до четырех недель.

В XP не рекомендуется тратить много времени на планирование; сам процесс планирования называется игрой (planning game). Подробный план составляется только на очередную итерацию и ближайшие один-два релиза.

Планирование релиза состоит из следующих шагов:

- Заказчик формулирует свои требования в виде историй, которые оцениваются по трудоемкости разработчиками. Оценки трудоемкости делаются в так называемых идеальных

днях – времени, которое некий воображаемый разработчик потратит на реализацию истории при полном отсутствии отвлекающих факторов, параллельных задач, перерывов и т.п.

- Истории сортируются по приоритету, рискам, сложности реализации.
- Определяется фактическая производительность команды (какое число реальных дней соответствует идеальному дню).
- На основании фактической производительности определяется, какие истории войдут в очередной релиз и за какое время он будет завершен.

Итерация планируется аналогичным образом.

- Предварительно определяется набор историй для итерации.
- Истории разбиваются на задачи (tasks), которые распределяются между разработчиками. В отличие от историй, которые описывают поведение системы с точки зрения пользователя, задачи составляются на техническом уровне, например «модифицировать схему базы данных» или «провести рефакторинг».
- Разработчики оценивают свои задачи. Казалось бы, это ненужное повторение операции оценки историй, однако на этом этапе уточненные оценки должны быть более реалистичными. Поскольку задачи теперь оценивают их непосредственные исполнители, в оценках учитывается индивидуальная производительность, зависящая от опыта, знакомства с используемыми технологиями и т.п.
- На основе уточненных оценок задач подсчитывается общая загрузка команды. В зависимости от загрузки некоторые истории могут быть перенесены на следующую итерацию или, наоборот, добавлены в текущую.

Периодически в ходе проекта измеряется фактическая производительность команды. Если она начинает сильно отличаться от значения, которое было использовано при планировании, график выхода релизов должен быть пересмотрен.

**Практика**

Во многом XP была создана как попытка описать процессы и практики, которые часто как бы сами собой появляются в эффективных, сплоченных командах разработчиков. Может показаться, что процесс в таких командах отсутствует, и, скорее всего, то же самое будут утверждать сами разработчики. Однако это не так, поскольку работа профессиональной команды всегда четко (хотя и неформально) организована и не имеет ничего общего с беспорядком и хаосом.

Это во многом определяет условия, необходимые для эффективного использования XP. Прежде всего, XP имеет шансы работать только в команде опытных, профессиональных разработчиков. Поскольку большую роль в XP играет прямое общение, команда не должна быть разбита на несколько частей – внедрение XP в распределенной географически команде будет крайне рискованным мероприятием. По той же причине возможный размер команды ограничен сверху – по всей видимости, числом в 10-15 человек.

Другие практики XP приносят свои ограничения. Далеко не всегда можно обеспечить постоянное присутствие представителя заказчика в проектной команде (например, если потенциальные пользователи системы делятся на несколько классов с частично конкурирующими требованиями).

Поскольку XP практически не делает попыток предотвратить размывание границ проекта (scope creep), будет не очень хорошей идеей использовать XP в проекте с фиксированной ценой. Фактически проекты XP обладают жестким графиком, но переменными границами, поэтому предпочтительным типом контракта будет повременная оплата (time&materials).

Практика поддержания максимально простой архитектуры может завести в тупик в конце проекта, когда окажется, что для реализации завершающих историй требуется полное перепроектирование системы (оказывается, нам нужно было предусмотреть возможность интеграции с системой SAP R/3, а также перевода на японский язык всего пользовательского интерфейса!). Даже если подобная ситуация не возникнет, нет никакой гарантии, что создаваемая ad hoc архитектура не будет намного более запутанной и сложной, чем было бы продуманное заранее решение.

Подобным образом может неконтролируемо откладываться разрешение некоторых нетехнологических рисков, наподобие неопределенных границ проекта.

Несмотря на все перечисленные ограничения, XP может замечательно работать в подходящих для него условиях. Благодаря крайне низким накладным расходам, в таких ситуациях этот процесс может показать исключительную эффективность.

XP является достаточно гибкой методологией. Не обязательно внедрять XP во всей компании, вполне разумно ограничиться теми командами и проектами, которые могут получить от этого реальный выигрыш. Например, для разработки ядра продукта можно использовать XP, а проекты по внедрению основывать на процессе RUP. Также не обязательно использовать все классические практики XP (на самом деле, мало кто это делает) – как правило, разумно ограничиться теми из них, которые сочетаются с корпоративной культурой и особенностями проектов.

**CMMI**

Модель Capability Maturity Model Integration (CMMI) была разработана в течение 1990-х годов в университете Карнеги-Меллона (Carnegie Mellon University.) совместно с Software Engineering Institute (SEI) и другими организациями. Одним из главных спонсоров разработки стало Министерство обороны США. CMMI была создана путем объединения (отсюда слово Integration в названии) трех более ранних специализированных моделей разработки: CMM for Software (SW-CMM), Electronic Industries Alliance Interim Standard (EIA/IS) 731 и Integrated Product Development Capability Maturity Model (IPD-CMM). Последняя версия спецификации CMMI 1.2 была опубликована в августе 2006 года.

Цель внедрения CMMI – построить инфраструктуру процессов, устанавливающую общий стандарт выполнения проектов внутри организации. Для каждого отдельного проекта стандартный процесс должен быть модифицирован в соответствии со спецификой этого проекта. При помощи формальных метрик измеряется эффективность процессов, а сами процессы постоянно оптимизируются.

Необходимо сказать, что CMMI не описывает какой-то конкретный процесс разработки. CMMI-совместимым может быть проект с водопадным, итеративным или другим жизненным циклом. Правильнее думать о CMMI как о наборе элементов процессов, приемов и методик, из которых, как из конструктора, нужно собрать законченный процесс.

Внедрение CMMI в организации может происходить на непрерывной (continuous) или ступенчатой (staged) основе. Содержание модели CMMI в обоих случаях одно и тоже, меняется только ее представление. Непрерывное представление дает возможность производить улучшения в отдельных процессных областях в произвольном порядке. Ступенчатое представление определяет четкую последовательность шагов по внедрению CMMI; каждый шаг соответствует достижению так называемого уровня зрелости (maturity level). Организации необходимо принять

решение, до какого уровня зрелости она намерена дойти, а также необходимо ли проходить официальную сертификацию на соответствие этому уровню.

Остановимся подробнее на ступенчатом представлении, поскольку именно оно чаще всего используется на практике.

**Структура CMMI (ступенчатое представление)**

Основными элементами модели CMMI являются процессные области, общие и специальные задачи, общие и специальные практики.

CMMI определяет 22 процессные области, такие как планирование проекта (Project Planning), управление рисками (Risk Management), разработка требований (Requirements Development) и т.д. В ступенчатом представлении процессные области сгруппированы по пяти уровням зрелости (от 1 до 5). В непрерывном представлении каждая процессная область находится на одном из шести (от 0 до 5) уровней производительности (capability level).

Процессы в каждой процессной области должны достигать ряда целей. Общие цели (generic goals) относятся к нескольким процессным областям. Специальные цели (specific goals) уникальны для своей процессной области.

Для достижения специальных и общих целей служат специальные и общие практики (practices). Практики – это деятельность или задачи, которые должны быть выполнены для достижения соответствующей цели.


Рисунок 4.

В качестве примера рассмотрим процессную область Планирование проекта (Project Planning). В нее входит определение проектных артефактов, разбиение работ на отдельные задачи и их оценка, планирование необходимых ресурсов, составление графика проекта, анализ рисков, и т.д. В результате планирования проекта создается план проекта (project plan) – основной документ для организации и контроля проектных работ, управления бюджетом и оценки доходности , управления изменениями и рисками.

ПРИМЕЧАНИЕ
*Часто путают план (plan) и график (schedule) проекта. План проекта – более широкое понятие; как правило, он включает в себя график.*

Процессная область Планирование проекта должна достигать трех специальных и двух общих целей. В рамках этих целей необходимо:

- Установить оценки (специальная цель): подготовить реалистичные оценки, такие как трудоемкость и стоимость проекта, для дальнейшего планирования.
- Разработать проектный план (специальная цель): создать документ, одобренный заинтересованными сторонами, описывающий жизненный цикл проекта, бюджет, ресурсы, риски, график, стратегию обеспечения качества и т.п.
- Получить обязательства участников проекта (специальная цель): участники проекта, ответственные за его выполнение, должны считать проектный план реалистичным и выполнимым, и подтвердить обязательство выполнить свои задачи в рамках заданного графика, ресурсов и качества.
- Учредить управляемый процесс (общая цель): общее требование к процессам на уровне зрелости 2.
- Учредить определенный (defined) процесс (общая цель): общее требование к процессам на уровне зрелости 3.

Перечислим практики, позволяющие достичь цели «разработать проектный план».

- Установить бюджет и график проекта.
- Идентифицировать риски.
- Спланировать управление данными (определить источники и форматы данных, необходимые процедуры для миграции, репликации и получения данных, требования по безопасности и т.д.).
- Определить необходимые ресурсы.
- Определить требования к профессиональным навыкам сотрудников, запланировать тренинги.
- Запланировать вовлечение всех заинтересованных лиц.
- Создать проектный план.

Как уже говорилось, ступенчатое представление CMMI позволяет классифицировать организации по уровням зрелости.

Уровень 1: начальный (Initial). Процессы в организации формируются спонтанно и хаотично. Отдельные проекты могут завершаться успешно, однако вероятность их успешного завершения, а также соответствие запланированным графику и бюджету малопредсказуемы.

Уровень 2: управляемый (Managed). Основная задача при внедрении этого уровня – установить для каждого проекта стандартные процессы управления требованиями, планирования, наблюдения и контроля над проектом, управления конфигурациями. Естественно, процессы должны соответствовать требованиям CMMI, а именно достигать всех специальных целей в соответствующих процессных областях, а также общих целей, относящихся к уровню 2. Для каждого проекта периодически проводится анализ его соответствия установленным процедурам и, при необходимости, корректирование процессов. Также периодически измеряются и анализируются метрики (производительности, качества и т.п.)

Уровень 3: определенный (Defined). Уровень 3 включает в себя все требования уровня 2, а также добавляет множество обязательных процессных областей (разработка требований, интеграция, тестирование, управление рисками, и другие). Однако главное его отличие от уровня 2 заключается не в этом. В то время как уровень 2 требует определить процесс для каждого отдельного проекта, на уровне 3 набор стандартных процессов должен существовать на уровне всей организации. Процессы для отдельных проектов создаются при помощи настройки (tailoring) стандартных процессов организации, при этом изменения должны быть ограничены правилами настройки (tailoring guidelines). Настройка процессов – это также процессная область, определенная в CMMI как Organizational Process Definition.

Уровень 4: количественно управляемый (Quantitatively Managed). Этот уровень требует количественного измерения метрик производительности и качества используемых процессов. По сравнению с уровнем 3 уровень 4 дает возможность предсказания и сравнения (на основе статистических данных) измеряемых характеристик процессов.

Уровень 5: оптимизирующий (Optimizing). Уровень 5 – высшая ступень развития организации, использующей CMMI. При помощи метрик с уровня 4 организация постоянно изменяет свои процессы для того, чтобы улучшить производительность и качество создаваемых продуктов.

**Практика**

В каких организациях и на каких проектах лучше всего работает CMMI? Для того чтобы понять это, необходимо выделить основные характеристики модели.

CMMI вполне заслуженно считается тяжеловесной методологией, причем требует ощутимых затрат как во время, так и после внедрения. Тяжеловесность процессов нарастает с продвижением по уровням зрелости. Уже первый из интересных в практическом смысле уровней зрелости – уровень 3 – требует значительных усилий по обучению проектной команды, ведению проектной документации, периодическому аудиту процессов.

Как уже говорилось, под требования CMMI можно подогнать самые разнообразные процессы разработки, однако некоторые сочетания явно не имеют смысла. К примеру, вряд ли можно найти разумное применение процессу XP, дополненному формальным документированием требований и сбором метрик.

Что получает организация в обмен на усилия и ресурсы, потраченные на внедрение CMMI? Прежде всего, предсказуемость сроков и бюджета выполнения более или менее аналогичных проектов. Точность планирования – это основная цель 3 и 4 уровней зрелости CMMI, эффективность разработки становится ключевым фактором лишь на уровне 5. Благодаря хорошо документированным процессам и промежуточным результатам работ становится возможным быстро подключать к проекту новых сотрудников (возможно, более типичная ситуация – заменять ушедших).

Таким образом, представляется целесообразным использование CMMI в следующих условиях.

- В относительно больших организациях, которые могут позволить себе значительные накладные расходы на внедрение и использование процесса.
- При высокой текучести кадров, когда тем не менее необходимо поддерживать скорость разработки и качество продуктов на достойном уровне.
- В случаях, когда организация выполняет большое количество более или менее однотипных проектов, CMMI позволяет достаточно точно планировать сроки и бюджет, и выполнять проекты в этих рамках.
- Важная, а часто и определяющая причина для внедрения CMMI – возможность официальной сертификации на достижение определенного уровня зрелости. Нередко наличие или отсутствие сертификации по CMMI является решающим фактором в выборе компании-подрядчика.

Очевидно, что не имеет большого смысла использовать CMMI для одного или нескольких отдельных проектов. Внедрение должно происходить во всей организации, департаменте и т.д.

Вряд ли использование CMMI принесет какие-либо преимущества при разработке новых продуктов, а особенно в исследовательских проектах (research and development). Поскольку при этом отнюдь не исчезают все накладные расходы, связанные с этой моделью, на мой взгляд,

нецелесообразно использовать CMMI для разработки новых продуктов или сервисов. Однако проекты по внедрению этих продуктов вполне могут эффективно использовать CMMI.

**Обзор методологии SCRUM**

Введение

Scrum - одна из самых популярных методологий гибкой разработки. Одна из причин ее популярности - простота. Scrum по-настоящему прост, его можно описать в одной короткой статье, что я и постараюсь сделать в этом обзоре.

Основа Scrum

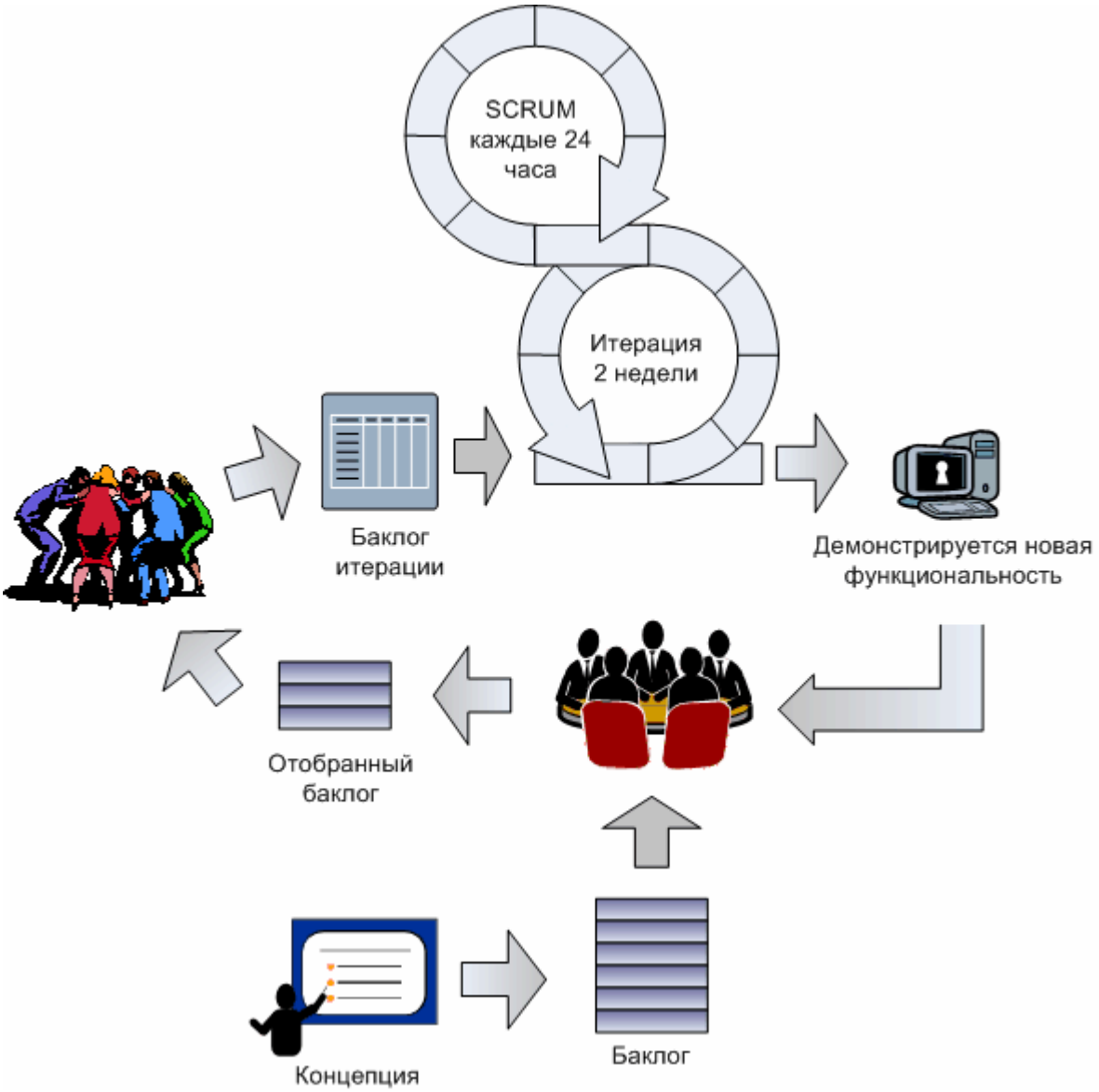


Рис. 1. Основа Scrum

**Роли**

В методологии Scrum всего три роли.
**Scrum Master**
**Product Owner**
**Team**

**Скрам Мастер (Scrum Master)**

Скрам Мастер (Scrum Master) - самая важная роль в методологии. Скрам Мастер отвечает за успех Scrum в проекте. По сути, Скрам Мастер является интерфейсом между менеджментом и командой. Как правило, эту роль в проекте играет менеджер проекта или тимлид. Важно подчеркнуть, что Скрам Мастер не раздает задачи членам команды. В Agile команда является самоорганизующейся и самоуправлямой.

Основные обязанности Скрам Мастера таковы:
Создает атмосферу доверия,
Участвует в митингах в качестве фасилитатора
Устраняет препятствия
Делает проблемы и открытые вопросы видимыми
Отвечает за соблюдение практик и процесса в команде

Скрам Мастер ведет Daily Scrum Meeting и отслеживает прогресс команды при помощи Sprint Backlog, отмечая статус всех задач в спринте.

ScrumMaster может также помогать Product Owner создавать Backlog для команды

**Product Owner**

Product Owner - это человек, отвечающий за разработку продукта. Как правило, это product manager для продуктовой разработки, менеджер проекта для внутренней разработки и представитель заказчика для заказной разработки. Product Owner - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один человек, а не группа или комитет.

Обязанности Product Owner таковы:
Отвечает за формирование product vision
Управляет ROI
Управляет ожиданиями заказчиков и всех заинтересованных лиц
Координирует и приоритизирует Product backlog
Предоставляет понятные и тестируемые требования команде
Взаимодействует с командой и заказчиком
Отвечает за приемку кода в конце каждой итерации

Product Owner ставит задачи команде, но он не вправе ставить задачи конкретному члену проектной команды в течении спринта.

**Команда (Team)**

В методологии Scrum команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Product Owner. Работа команды оценивается как работа единой группы. В Scrum вклад отдельных членов проектной команды не оценивается, так как это разваливает самоорганизацию команды.

Обязанности команды таковы:
Отвечает за оценку элементов баклога
Принимает решение по дизайну и имплементации
Разрабатывает софт и предоставляет его заказчику
Отслеживает собственный прогресс (вместе со Скрам Мастером).
Отвечает за результат перед Product Owner

Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные размер команды - 7 плюс минус 2.

Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками - разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды. Команда состоит из инженеров, которые вносят свой вклад в общий успех проекта в соответствии со своими способностями и проектной необходимостью. Команда самоорганизуется для выполнения конкретных задач в проекте, что позволяет ей гибко реагировать на любые возможные задачи.

Для облегчения коммуникаций команда должна находиться в одном месте (colocated). Предпочтительно размещать команду не в кубиках, а в одной общей комнате для того, чтобы уменьшить препятствия для свободного общения. Команде необходимо предоставить все необходимое для комфортной работы, обеспечить досками и флипчартами, предоставить все необходимые инструменты и среду для работы.

**Артефакты**
**Product Backlog**

Product Backlog - это приоритезированный список имеющихся на данный момент бизнес-требований и технических требований к системе.

Product Backlog включает в себя use cases, defects, enhancements, technologies, stories, features, issues, и т.д.. Product backlog также включает задачи, важные для команды, например "провести тренинг", "добить всем памяти"

| Feature | Estimation, man*it | Priority |
|---|---|---|
| **Интеграция с IntHR** | | |
| Синхронизация сотрудников, орг структуры, позиций и т.д. с учетом multisite | **2** | Must |
| **ФИН** | | |
| Расчет утилизации и проч. финансовых показателей | **1** | Must |
| Расчет затрат по реальным зарплатам | **1** | Must |
| **Редактирование справочных таблиц** | **1** | Must |
| Закрытый период | | |
| Заказчики | | |
| Внутренние ставки | | |
| **Закрытый период по проектам** | **1** | Should |
| **Поддержка разных типов проектов** | **1** | Should |
| Административный проект подразделения | | |
| Проект обучения | | |
| **Multisite** | | |
| Синхронизация данных между серверами, управление синхронизацией | **4** | Must |
| Отчеты уровня компании | **2** | Must |
| **Внешний бюджет** | | |
| Доработка интерфейса (общий стиль, тулбары, usability, AJAX) | **2** | Should |
| Additional view | **2** | Should |
| By Month | | |
| By Person | | |
| Прочие доходы | **1** | Should |
| Автоматизация PSR | **1** | Must |

Рис. 2. Пример Product Backlog

Product Backlog постоянно пересматривается и дополняется - в него включаются новые требования, удаляются ненужные, пересматриваются приоритеты. За Product Backlog отвечает Product Owner. Он также работает совместно с командой для того, чтобы получить приближенную оценку на выполнение элементов Product Backlog для того, чтобы более точно расставлять приоритеты в соответствии с необходимым временем на выполнение.

**Sprint Backlog**

Sprint Backlog содержит функциональность, выбранную Product Owner из Product Backlog. Все функции разбиты по задачам, каждая из которых оценивается командой. Каждый день команда оценивает объем работы, который нужно проделать для завершения задач.

| | Days Left in Sprint | 15 | 13 | 10 | 8 | |
|---|---|---|---|---|---|---|
| Who | Description | 7/22/2002 | 7/24/2002 | 7/26/2002 | 7/31/2002 | |
| | **Total Estimated Hours:** | **554** | **458** | **362** | **270** | **0** |
| - | **User's Guide** | - | - | - | - | - |
| SM | Start on Study Variable chapter first draft | 16 | 16 | 16 | 16 | |
| SM | Import chapter first draft | 40 | 24 | 6 | 6 | |
| SM | Export chapter first draft | 24 | 24 | 24 | 6 | |
| | **Misc. Small Bugs** | | | | | |
| JM | Fix connection leak | 40 | | | | |
| JM | Delete queries | 8 | 8 | | | |
| JM | Delete analysis | 8 | 8 | | | |
| TG | Fix tear-off messaging bug | 8 | 8 | | | |
| JM | View pedigree for kindred column in a result set | 2 | 2 | 2 | 2 | |
| AM | Derived kindred validation | 8 | | | | |
| | **Environment** | | | | | |
| TG | Install CVS | 16 | 16 | | | |
| TBD | Move code into CVS | 40 | 40 | 40 | 40 | |
| TBD | Move to JDK 1.4 | 8 | 8 | 8 | 8 | |
| | **Database** | | | | | |
| KH | Killing Oracle sessions | 8 | 8 | 8 | 8 | |
| KH | Finish 2.206 database patch | 8 | 2 | | | |
| KH | Make a 2.207 database patch | 8 | 8 | 8 | 8 | |
| KH | Figure out why 461 indexes are created | 4 | | | | |

Рис. 3. Пример Spint Backlog

Сумма оценок оставшейся работы может быть построена как график зависимости от времени. Такой график называется Sprint Burndown chart. Он демонстрирует прогресс команды по ходу спринта.

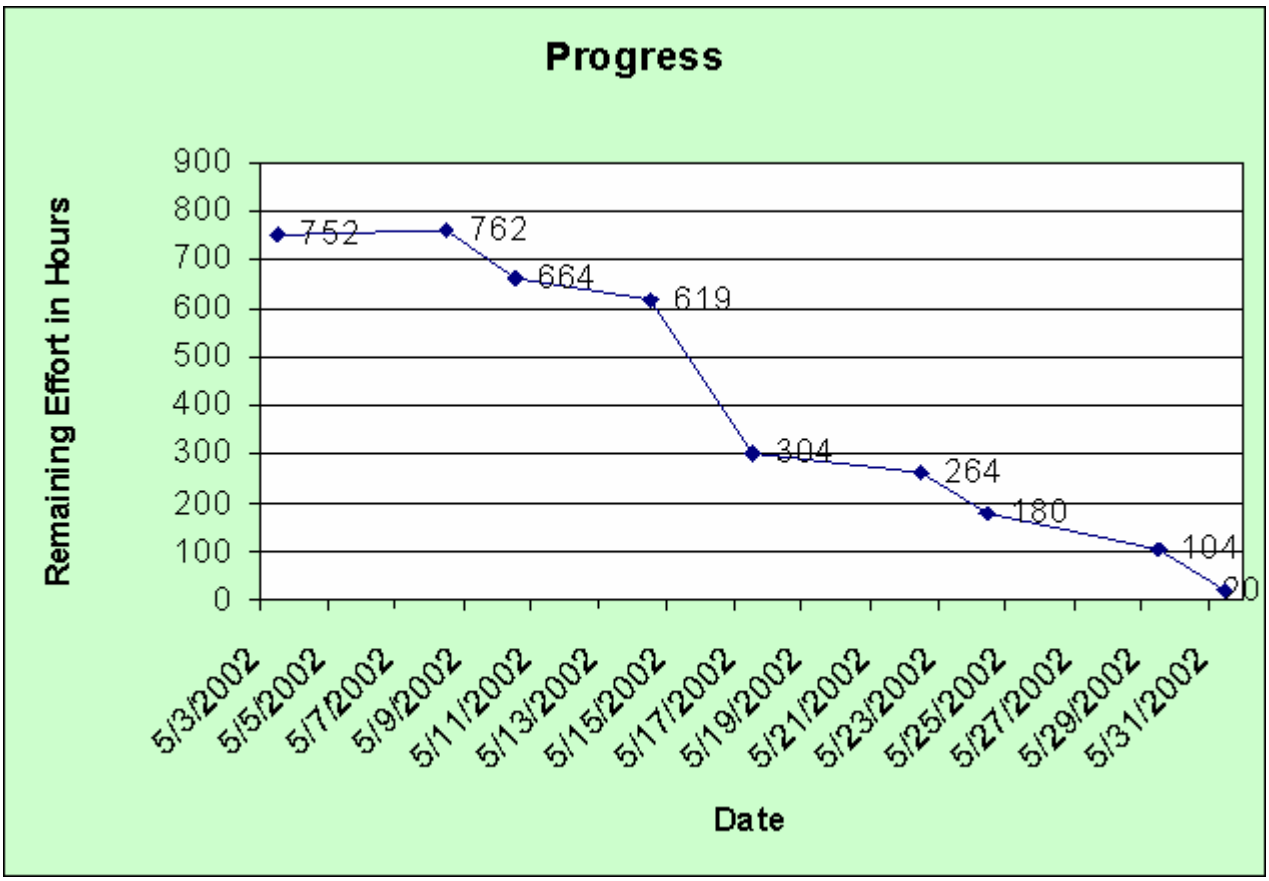


Рис. 4. Пример Sprint Burndown chart

**Спринт (Sprint)**

В Scrum итерация называется Sprint. Ее длительность составляет 1 месяц (30 дней).

Результатом Sprint является готовый продукт (build), который можно передавать (deliver) заказчику (по крайней мере, система должна быть готова к показу заказчику).

Короткие спринты обеспечивают быстрый feedback проектной команде от заказчика. Заказчик получает возможность гибко управлять scope системы, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в Product Backlog, приоритезируются наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов.

Каждый спринт представляет собой маленький "водопад". В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта.

Scope спринта должен быть фиксированным. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в спринте. Это означает, что Sprint Backlog не может быть изменен никем, кроме команды.

**Жизненный цикл спринта**
**Планирование спринта**

В начале каждого спринта проводится планирование спринта. В планировании спринта участвуют заказчики, пользователи, менеджмент, Product Owner, Скрам Мастер и команда.

Планирование спринта состоит из двух последовательных митингов.

**Планирование спринта, митинг первый**

Участники: команда, Product Owner, Scrum Master, пользователи, менеджмент

Цель: Определить цель спринта (Sprint Goal) и Sprint Backlog -функциональность, которая будет разработана в течение следующего спринта для достижения цели спринта.

Артефакт: Sprint Backlog

**Планирование спринта, митинг второй**

Участники: Скрам Мастер, команда

Цель: определить, как именно будет разрабатываться определенная функциональность для того, чтобы достичь цели спринта. Для каждого элемента Sprint Backlog определяется список задач и оценивается их продолжительность.

Артефакт: в Sprint Backlog появляются задачи

Если в ходе спринта выясняется, что команда не может успеть сделать запланированное на спринт, то Скрам Мастер, Product Owner и команда встречаются и выясняют, как можно сократить scope работ и при этом достичь цели спринта.

**Остановка спринта (Sprint Abnormal Termination)**

Остановка спринта производится в исключительных ситуациях. Спринт может быть остановлен до того, как закончатся отведенные 30 дней. Спринт может остановить команда, если понимает, что не может достичь цели спринта в отведенное время. Спринт может остановить Product Owner, если необходимость в достижении цели спринта исчезла.

После остановки спринта проводится митинг с командой, где обсуждаются причины остановки спринта. После этого начинается новый спринт: производится его планирование и стартуются работы.

**Daily Scrum Meeting**

Этот митинг проходит каждое утро в начале дня. Он предназначен для того, чтобы все члены команды знали, кто и чем занимается в проекте. Длительность этого митинга строго ограничена и не должна превышать 15 минут. Цель митинга - поделиться информацией. Он не предназначен для решения проблем в проекте. Все требующие специального обсуждения вопросы должны быть вынесены за пределы митинга.

Скрам митинг проводит Скрам Мастер. Он по кругу задает вопросы каждому члену команды
Что сделано вчера?
Что будет сделано сегодня?
С какими проблемами столкнулся?

Скрам Мастер собирает все открытые для обсуждения вопросы в виде Action Items, например в формате что/кто/когда, например
Обсудить проблему с отрисовкой контрола

Петя и Вася
Сразу после скрама

**Демо и ревью спринта**
Рекомендованная длительность: 4 часа

Команда демонстрирует инкремент продукта, созданный за последний спринт. Product Owner, менеджмент, заказчики, пользователи, в свою очередь, его оценивают. Команда рассказывает о поставленных задачах, о том как они были решены, какие препятствия были у них на пути, какие были приняты решения, какие проблемы остались нерешенными. На основании ревью принимающая сторона может сделать выводы о том, как должна дальше развиваться система. Участники миитинга делают выводы о том, как шел процесс в команде и предлагает решения по его улучшению.

Скрам Мастер отвечает за организацию и проведение этого митинга. Команда помогает ему составить адженду и распланировать кто и в какой последовательности что представляет.

Подготовка к митингу не должна занимать у команды много времени (правило - не более двух часов). В частности, именно поэтому запрещается использовать презентации в Power Point. Подготовка к митингу не должна занимать у команды более 2-х часов.