ГИБКАЯ МЕТОДОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ AGILE

Владимир Макаров

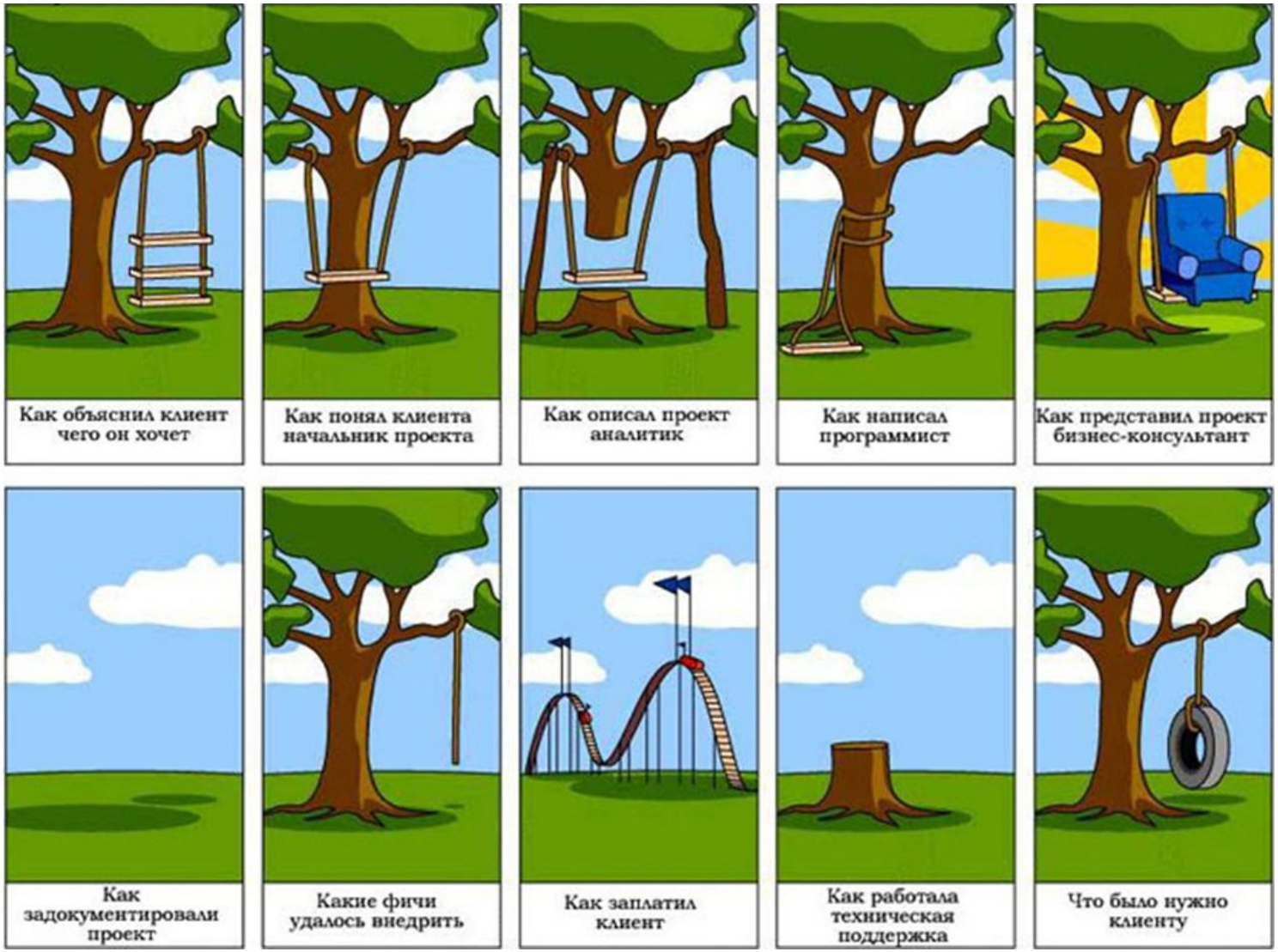
19.12.16

Распространенные проблемы разработки ПО

- Изменение требований непосредственно в процессе разработки.
- Нечеткое распределение ответственности за работу и ее результат.
- Наличие непрерывного потока мелких, «быстрореализуемых» требований, отвлекающих разработчиков и менеджеров от основного направления работ.

Результат : срыв сроков, раздувание бюджетов, потеря качества.

Распространенные проблемы разработки ПО



Agile

- Гибкая методология разработки (Agile software development) – это набор *принципов и правил*, в рамках которых осуществляется разработка ПО.

- Методология Agile – это семейство процессов разработки, а не единственный подход к разработке программного обеспечения.

Agile

Манифест Agile подписали представители следующих методологий:

- Extreme programming
- Scrum
- DSDM
- Adaptive Software Development
- Crystal Clear
- Feature-Driven Development
- Pragmatic Programming

Ценности Agile-методологии

- работающее программное обеспечение важнее, чем полная документация;
- личности и их взаимодействия важнее, чем процессы и инструменты;
- сотрудничество с заказчиком важнее, чем контрактные обязательства;
- реакция на изменения важнее, чем следование плану.

Принципы Agile-методологии

- удовлетворение клиента за счёт ранней и бесперебойной поставки качественного ПО;
- частая поставка рабочего ПО (не реже раз в месяц);
- возможность изменения требований, даже в конце разработки. Цель - конкурентоспособность продукта;
- тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;
- проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;
- рекомендуемый метод передачи информации – личный разговор (лицом к лицу);

Принципы Agile-методологии

- работающее ПО – лучший измеритель прогресса;
- заказчик, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределенный срок;
- постоянное внимание на улучшение технического мастерства и удобный дизайн;
- простота – искусство НЕ делать лишней работы;
- лучшие архитектура, требования и дизайн получаются у самоорганизованной команды;
- постоянная (частая) адаптация (улучшение эффективности работы) к изменяющимся обстоятельствам.

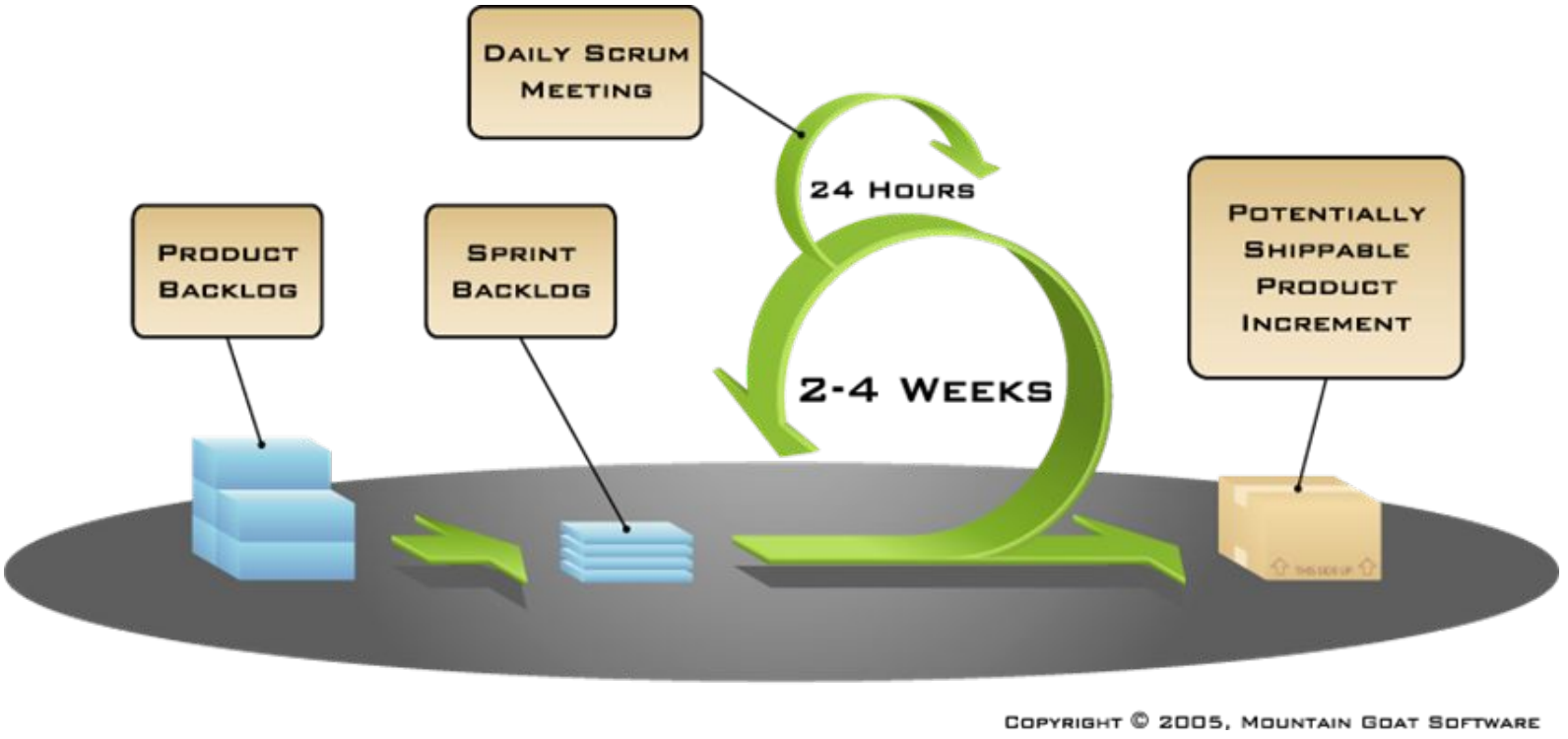
Scrum

Scrum – одна из наиболее общераспространенных методологий гибкой разработки ПО.

Scrum - набор принципов, на которых строится процесс разработки:

- жёстко фиксированные промежутки времени (**спринты** от 2 до 4 недель)
- работающее ПО для заказчика с функционалом, для которого определён наибольший приоритет.
- требуемый функционал в очередном спринте определяется до его начала на этапе планирования и не может изменяться на всём протяжении спринта.

Scrum


Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software

Состав команды спринта

1. Scrum Master (Скрам Мастер).
2. Product owner (Заказчик продукта).
3. Team (Команда).

Scrum Master

Scrum Master отвечает за успех Scrum (как принципа организации работы команды) в проекте.

Scrum Master является интерфейсом между менеджментом и командой. Как правило, эту роль в проекте играет менеджер проекта.

- создает атмосферу доверия;
- устраняет препятствия;
- делает проблемы и открытые вопросы видимыми;
- отвечает за соблюдение практик и процесса в команде.

Product Owner

Менеджер продукта -- для продуктовой разработки.
Менеджер проекта -- для внутренней разработки.
Представитель заказчика -- для заказной разработки.

Заказчик – это единая точка принятия окончательных решений для команды в проекте, это всегда один человек, а не группа или комитет.

Product Owner

- управляет рентабельностью, управляет ожиданиями заказчиков и всех заинтересованных лиц;
- отвечает за формирование видения продукта;
- координирует и определяет приоритет задач спринта;
- предоставляет понятные и тестируемые требования команде;
- взаимодействует с командой и заказчиком;
- отвечает за приемку кода в конце каждой итерации.

Заказчик ставит задачи **команде**. Не может ставить задачи конкретному члену проектной команды в течение спринта.

Team

- **Команда** является самоорганизующейся и самоуправляемой.
- **Команда** берет на себя обязательства по выполнению объема работ на спринт перед Product Owner.
- Вклад отдельных членов проектной команды не оценивается, так как это разваливает самоорганизацию команды. Оценивается единая группа.

Team

- Отвечает за своевременную сборку и ее качество;
- Принимает решение по дизайну и имплементации;
- Разрабатывает ПО и предоставляет его заказчику;
- Отслеживает собственный прогресс совместно с Scrum Master;
- Отвечает за результат перед Product Owner.

12 техник Экстремального программирования

- Короткий цикл обратной связи (Fine scale feedback)
  - Разработка через тестирование (Test driven development)
  - Игра в планирование (Planning game)
  - Заказчик всегда рядом (Whole team, Onsite customer)
  - Парное программирование (Pair programming)
- Непрерывный, а не пакетный процесс
  - Непрерывная интеграция (Continuous Integration)
  - Рефакторинг (Design Improvement, Refactor)
  - Частые небольшие релизы (Small Releases)

12 техник Экстремального программирования

- Понимание, разделяемое всеми
  - Простота (Simple design)
  - Метафора системы (System metaphor)
  - Коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership)
  - Стандарт кодирования (Coding standard or Coding conventions)
- Социальная защищенность программиста (Programmer welfare):
  - 40-часовая рабочая неделя (Sustainable pace, Forty hour week)

User stories

Все начинается с **User Stories.**

- User Story (что-то типа рассказа пользователя) - это описание того как система должна работать. Каждая User Story написана на карточке и представляет какой-то кусок функциональности системы, имеющий логический смысл с точки зрения Заказчика. Форма - один-два абзаца текста понятного пользователю (не сильно технического).
- User Story пишется Заказчиком. Они похожи на сценарии использования системы, но не ограничиваются пользовательским интерфейсом. По каждой истории пишутся функциональные тесты, подтверждающие что данная история корректно реализована - их еще называют приемочными (Acceptance tests).

User stories

- Каждой User Story дается приоритет со стороны бизнеса (пользователь, заказчик, отдел маркетинга) и оценка времени выполнения со стороны разработчиков. Каждая история разбивается на задачи и ей назначается время когда ее начнут реализовывать.
- User Stories используются в XP вместо традиционных требований. Главное отличие User Story от требований (requirements) - уровень детализации. User Story содержит минимум информации, необходимой для обоснованной оценки того, сколько времени надо на ее реализацию.
- Типичная User Story занимает 1-3 недели идеального времени. История требующая менее 1 недели слишком детализирована. История требующая более 3 недель может быть разбита на части - отдельные истории.

Планирование релиза

Если менеджмент не устраивают сроки завершения, может появиться соблазн уменьшить оценки объема работ.

Вы никогда не должны этого делать. Заниженные оценки обязательно создадут множество проблем позже. Вместо этого ведите переговоры с менеджерами, разработчиками и заказчиками пока не выработаете приемлемый для всех план релиза !!!!

Планирование релиза

План Релиза разрабатывается на собрании по планированию Релиза.

- Релиз Планы описывают взгляд на весь проект и используются в дальнейшем для планирования итераций.
- Важно, чтобы технические люди делали технические решения и люди бизнеса - бизнес решения. Планирование Релиза определяет набор правил, которые позволяют всем принимать свои решения. Эти правила определяют метод выработки удовлетворяющего всех плана работ.
- Сущность собрания по планированию релиза для команды разработчиков в том, чтобы оценить каждую User Story в идеальных неделях. **Идеальная неделя** - это сколько по-вашему займет время выполнение задачи, если ничто больше вас не будет отвлекать.

Релизы

- Выпускать частые **небольшие Релизы. Постоянно измеряется Скорость проекта**

- Скорость Проекта (или просто скорость) это мера того, как быстро выполняется работа в вашем проекте.

- Чтобы измерить скорость проекта нужно посчитать объем User Stories, или как много (по времени) задач было выполнено за итерацию. Просто посчитайте суммарное время оценки объема работы (идеальное время).

Итерации плана

- **Итеративная разработка увеличивает гибкость процесса**. Разделите план на итерации продолжительностью от 2 до 3 недель. Сохраняйте постоянную продолжительность итерации на время проекта. Пусть итерации будут пульсом вашего проекта. Это тот ритм который позволит сделать измерение прогресса и планирование простым и надежным.
- **Не планируйте задач заранее.** Нужно планировать, что будет сделано в каждой итерации. Нарушением правил считается желание забегать вперед и делать то, что не запланировано в этой итерации. В этом случае становится возможным держать под контролем изменяющиеся требования Заказчика.

Обмен задачами

- Необходимо периодически менять задачи у разработчиков для уменьшения риска концентрации знаний и узких мест в коде. Если только один человек в команде может работать в данной области – БОЛЬШОЙ риск.
- Cross Training практика, которая позволяет избежать концентрации знаний в одном человеке. Cross Training - перемещение людей по коду в комбинации с парным программированием Вместо одного человека, который знает все о данном куске кода, каждый в вашей команде знает много о коде в каждом модуле.
- **Каждый день начинается с утреннего Собрания стоя**

Метафора системы

- System Metaphor - простая и понятная концепция, чтобы члены команды называли все вещи одинаковыми именами. Для понимания системы и исключения дублирующего кода чрезвычайно важно, как вы называете объекты.

- Создается система имен для своих объектов так, чтобы каждый член команды мог пользоваться ею без специальных знаний о системе. Чрезвычайно экономит время.

Рефакторинг

В программировании термин рефакторинг означает изменение исходного кода программы без изменения его внешнего поведения.

В экстремальном программировании и других гибких методологиях рефакторинг является неотъемлемой частью цикла разработки ПО: разработчики попеременно то создают новые тесты и функциональность, то выполняют рефакторинг кода для улучшения его логичности и прозрачности.

Рефакторинг

- Программисты, склонны держаться за дизайн долго после того как он становится неуклюжим. Продолжаем повторно использовать неудобный в сопровождении код поскольку он все еще как-то работает и мы боимся испортить его.
- *Рефакторинг в конечном итоге экономит время и улучшает качество продукта.*
- Безжалостно пересматривайте любой код для того, чтобы сохранять дизайн простым по мере разработки. Сохраняйте код ясным и понятным, чтобы его было легко понять, модифицировать и расширять. Удостоверьтесь, что все написано один и только один раз. В конечном итоге, это занимает меньше времени, чем доводить до ума запутанную систему.

Кодирование

- Заказчик всегда рядом.
- Весь код должен соответствовать принятому стандарту.
- Весь код должен быть создан парным программированием.
- Частая интеграция кода.
- Коллективное владение кодом.
- Оставлять оптимизацию на потом.

Тестирование

Любой код должен иметь **Unit Test.**

- **Unit тесты** играют ключевую роль в XP. Они позволяют быстро менять код не боясь наделать новых ошибок. Unit тест пишется для каждого класса, тест должен проверять все аспекты работы класса - тестировать все что может не работать.

- **Важно!!!** Тест для класса должен быть написан раньше самого класса. Это значит, что как только вы выпустите первый работающий результат, он будет поддерживаться системой тестирования. Для проведения тестов пишется специальная система тестирования со своим интерфейсом.

Тестирование

- Unit тест для класса хранится в общем репозитории вместе с кодом класса. Никакой код не может быть выпущен без Unit теста. Перед отдачей кода разработчик должен удостовериться, что все тесты проходят без ошибок. Никто не может отдать код, если все не прошли 100%. Иногда бывает неправильным или неполным код теста. В таком случае надо исправлять его.
- Искушением экономить на Unit тестах, когда мало времени, надо изживать. Чем сложнее написать тест, тем больше времени он потом сэкономит. Доказано практикой!

- Unit тесты позволяют осуществить коллективное владение кодом. Они позволяют относительно легко пересматривать плохой код. Также Unit тесты позволяют в любой момент иметь стабильно работающую систему.

Тестирование

- **Все Unit тесты должны проходить перед сдачей проекта.**
- **Если найден баг, то тесты корректируются или создаются новые.**
- **Функциональные тесты периодически выполняются и их результаты публикуются для членов команды.**