Разработка через тестирование

Разработка через тестирование

Agile-методологии разработки

В феврале 2001 года 17 специалистов (консультантов и практиков) в местечке под названием Snowbird в штате Юта собрались, чтобы обсудить легковесные методики разработки. В результате родился документ «Манифест гибких методологий разработки» (Agile Manifesto).

Основные тезисы манифеста:

- Люди и их взаимодействие важнее процессов и инструментов;
- Готовый продукт важнее документации по нему;
- Сотрудничество с заказчиком важнее жестких контрактных ограничений;

Разработка через тестирование

- Реакция на изменения важнее следования плану.

Полный текст манифеста (и его переводы) доступны на сайте http://agilemanifesto.org

Модель MSF for Agile Software Development

Модель процессов MSF представляет общую методологию разработки и внедрения IT- решений. Особенность этой модели состоит в том, что благодаря своей гибкости и отсутствию жестко навязываемых процедур она может быть применена при разработке весьма широкого круга IT проектов.

Разработка через тестирование

Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной (waterfall) и спиральной (spiral).

Процесс MSF ориентирован на «вехи» (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду. Рассмотрим основные принципы модели процессов.

Разработка через тестирование

Взаимодействуйте с «заказчиками».

MSF настаивает на непрерывном взаимодействии с заказчиком в ходе всей работы над проектом. Удовлетворенный заказчик – главный приоритет проектной группы. Понимание бизнес-задач заказчика от проекта, потребностей его будущих пользователей требует максимальной полной вовлеченности заказчика в процесс создания решения.

Поощряйте свободный обмен информацией в проекте

Успех коллективной работы над проектом немыслим без наличия у членов проектной группы и заказчика единого видения (shared vision), т.е. четкого и одинакового понимания целей и задач проекта.

Разработка через тестирование

Изначально понимание того, что должно быть достигнуто в ходе работы над проектом, у заказчика может (и нередко) не совпадать с пониманием проектной группы. Лишь наличие единого видения способно внести ясность и обеспечить движение всех заинтересованных в проекте сторон к общей цели.

Формирование единого видения и последующее следование ему являются столь важными, что модель процессов MSF выделяет для этой цели специальную фазу (фаза «Выработка концепции»), которая заканчивается соответствующей вехой.

Разработка через тестирование

Следите за качеством продукта

MSF настаивает на том, что каждый участник проектной группы должен ощущать ответственность за качество разрабатываемого решения.

Она не может быть делегирована одним членом команды другому или же от одной ролевой группы другой.

Несмотря на наличие в команде ролевых групп “Удовлетворение потребителя” и “Тестирование”, прямая задача которых – следить за качеством решения и повышать его, все остальные ролевые группы также постоянно должны иметь в виду нужды конечного пользователя.

Разработка через тестирование

Проявляйте гибкость – будьте готовы к изменениям

MSF основывается на принципе непрерывной изменяемости условий проекта при неизменной эффективности управленческой деятельности. Проектная группа должна быть готова к переменам, и методология MSF предоставляет эффективный инструментарий для адекватной и своевременной реакции на изменения в проектной среде.

Ставьте «вехи»

Каждая итерация, каждая фаза процесса создания решения должна заканчиваться некоторым зримым результатом, некоторой вехой (milestone).

Разработка через тестирование

Наличие вех позволяет проектной группе и заказчику видеть движение проекта вперед.

Будьте готовы к внедрению сегодня

Работа проектной группы в идеале должна быть построена так, чтобы при возникновении такой потребности у заказчика текущее состояние разрабатываемого решения могло быть немедленно внедрено (естественно с той функциональностью, которая в данный момент реализована). Это означает, что итерации работы над проектом должны быть максимально короткими, и в каждый момент времени должна существовать текущая работоспособная версия решения

Разработка через тестирование

Такой подход дает возможность раннего обнаружения рисков, ошибок, пропущенных или недопонятых требований, дает возможность своевременно получать обратную реакцию от заказчика, а, значит, сокращает затраты.

Управление компромиссами

В силу свойственной IT-проектам неопределенности и рискованности, одним из ключевых факторов их успеха являются эффективные компромиссные решения (trade-offs).

Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (рамками).

Разработка через тестирование

Эти три переменные образуют треугольник компромиссов (tradeoff triangle), приведенный на рисунке 6.1.

Рис. 6.1 - Треугольник компромиссов

Разработка через тестирование

После достижения равновесия в этом треугольнике изменение на любой из его сторон для поддержания баланса требует модификаций на другой (двух других) сторонах и/или на изначально измененной стороне.

Нахождение верного баланса между ресурсами, временем разработки и возможностями – ключевой момент в построении решения, должным образом отвечающего нуждам заказчика..

Матрица компромиссов проекта

Другое полезное средство управления проектными компромиссами – матрица компромиссов проекта (project tradeoff matrix).

Разработка через тестирование

Она отражает достигнутое на ранних этапах проекта соглашение между проектной группой и заказчиком о выборе приоритетов в возможных в будущем компромиссных решениях. Возможный вариант такой матрицы представлен на рисунке 6.2.

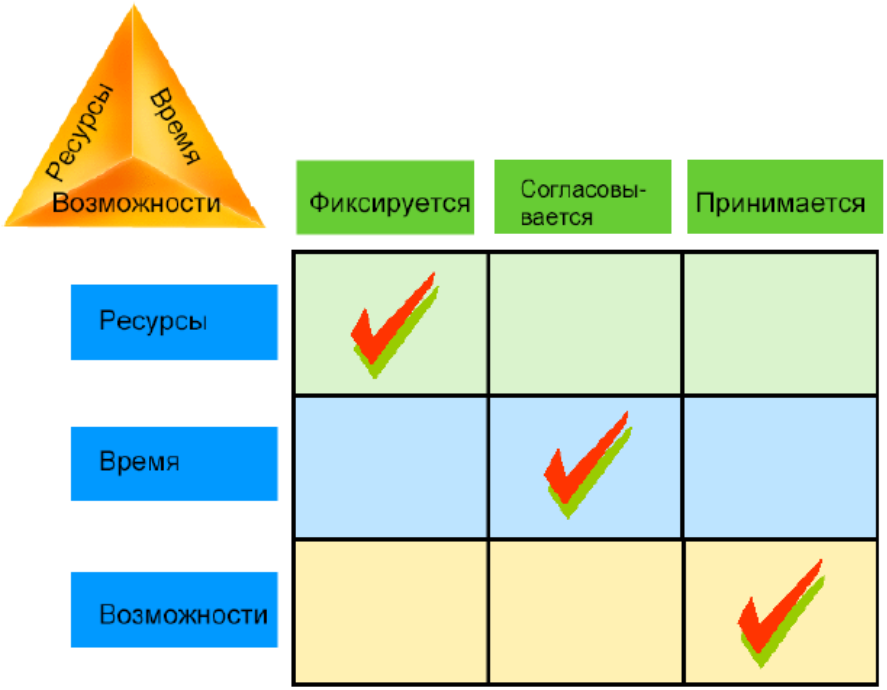

Рис. 6.2 - Матрица компромиссов (вариант)

Разработка через тестирование

Матрица компромиссов помогает обозначить проектное ограничение, воздействие на которое практически невозможно (колонка «Фиксируется»), фактор, являющийся в проекте приоритетным (колонка «Согласовывается»), и третий параметр, значение которого должно быть принято в соответствии с установленными значениями первых двух величин (колонка «Принимается»).

Разработка через тестирование

Модель Rational Unified Process (RUP)

Модель жизненного цикла RUP является довольно сложной, детально проработанной итеративно-инкрементной моделью с элементами каскадной модели.

В модели RUP выделяются 4 основные фазы, 9 видов деятельности (процессов). Кроме того, в модели описывается ряд практик, которые следует применять или руководствоваться для успешного выполнения проекта. RUP ориентирован на поэтапное моделирование создаваемого продукта с помощью языка UML (рис. 6.3).

Разработка через тестирование

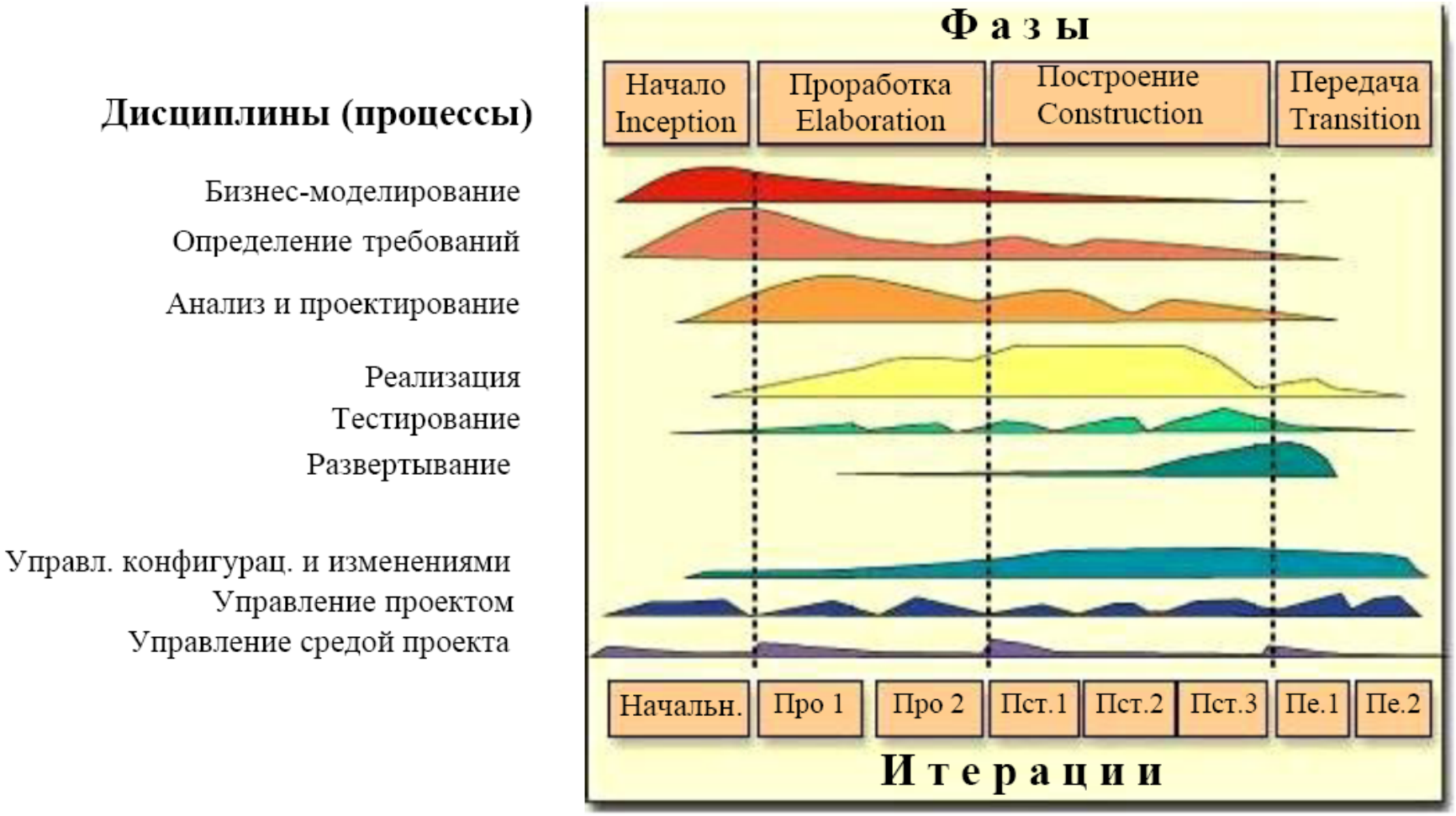

Рис. 6.3 - Модель RUP

Разработка через тестирование

Основными фазами RUP являются:

- **Фаза начала проекта (Inception).** Определяются основные цели проекта, бюджет проекта, основные средства его выполнения - технологии, инструменты, ключевой персонал, составляются предварительные планы проекта. Основная цель этой фазы - достичь компромисса между всеми заинтересованными лицами относительно задач проекта.
- **Фаза проработки (Elaboration).** Основная цель фазы - на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать стоящие перед системой задачи и в дальнейшем используется как основа разработки системы.

Разработка через тестирование

- **Фаза построения (Construction).** Основная цель этой фазы - детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры.
- **Фаза передачи (Transition).** Цель фазы - сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей.

В рамках каждой фазы возможно проведение нескольких итераций, количество которых определяется сложностью выполняемого проекта.

Разработка через тестирование

Деятельности (основные процессы) RUP делятся на пять **рабочих** и четыре **поддерживающие**.

К **рабочим** деятельностям относятся:

- Моделирование предметной области (бизнес-моделирование, Business Modeling).

  Цели этой деятельности - понять бизнес-контекст, в котором должна будет работать система (и убедиться, что все заинтересованные лица понимают его одинаково), понять возможные проблемы, оценить возможные их решения и их последствия для бизнеса организации, в которой будет работать система.

Разработка через тестирование

- Определение требований (Requirements).

  Цели - понять, что должна делать система, определить границы системы и основу для планирования проекта и оценок ресурсозатрат в нем.
- Анализ и проектирование (Analysis and Design).

  Выработка архитектуры системы на основе ключевых требований, создание проектной модели, представленной в виде диаграмм UML, описывающих продукт с различных точек зрения..
- Реализация (Implementation).

  Разработка исходного кода, компонент системы, тестирование и интегрирование компонент.

Разработка через тестирование

- Тестирование (Test).

  Общая оценка дефектов продукта, его качество в целом; оценка степени соответствия исходным требованиям.

Поддерживающими деятельностями являются:

- Развертывание (Deployment).

  Цели - развернуть систему в ее рабочем окружении и оценить ее работоспособность.

- Управление конфигурациями и изменениями (Configuration and Change Management).
  Определение элементов, подлежащих хранению и правил построения из них согласованных конфигураций.

Разработка через тестирование

Поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.

- Управление проектом (Project Management).

  Включает планирование, управление персоналом, обеспечения связей с другими заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.
- Управление средой проекта (Environment).

  Настройка процесса под конкретный проект, выбор и смена технологий и инструментов, используемых в проекте.

Разработка через тестирование

Модель Extreme Programming (XP)

Экстремальное программирование является примером так называемого метода «живой» разработки (Agile Development Method).

Модель жизненного цикла XP является итерационно-инкрементной моделью быстрого создания (и модификации) прототипов продукта, удовлетворяющих очередному требованию (user story). Особенности этой модели представлены на рисунке 6.4.

Разработка через тестирование

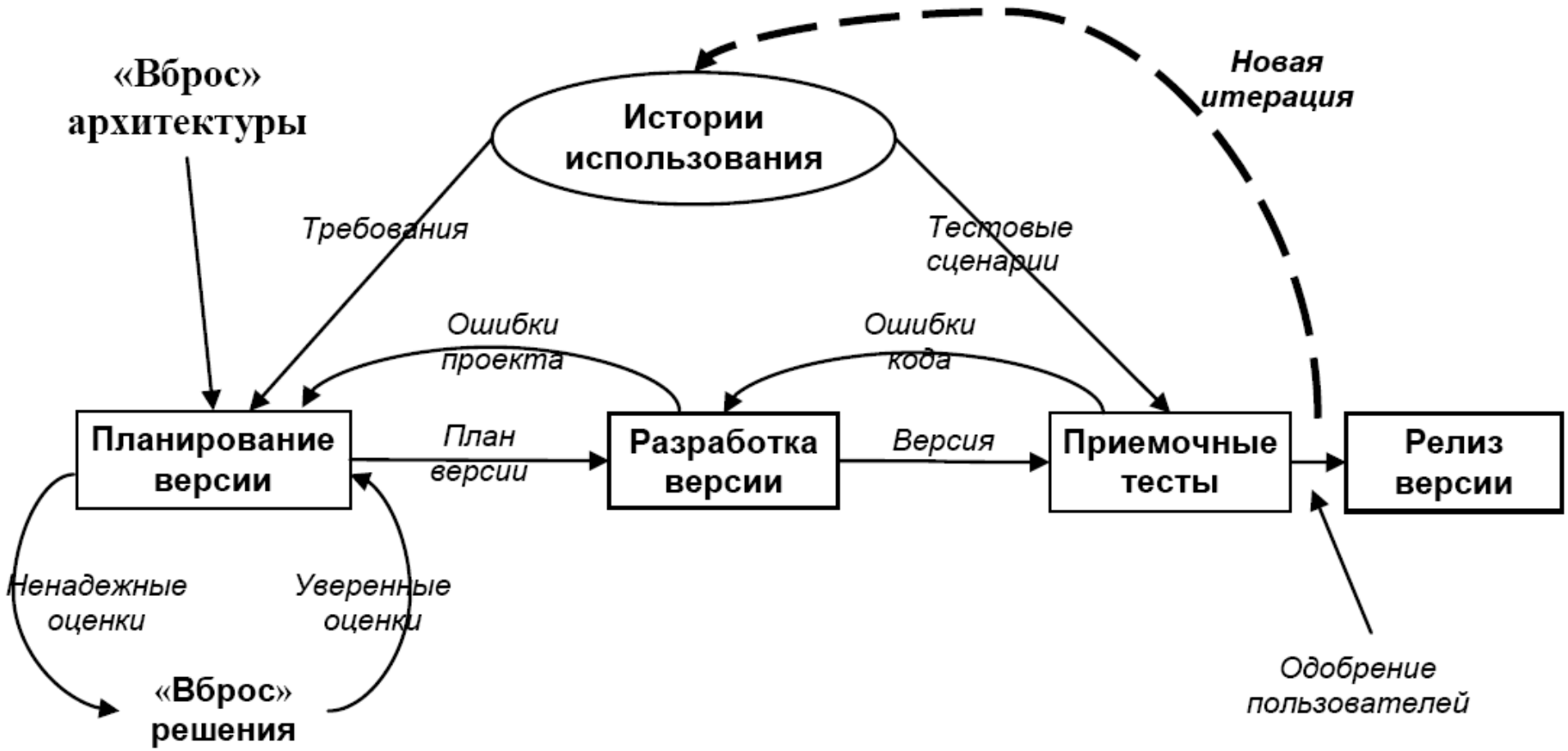

Рис. 6.4 - Модель Extreme Programming (XP)

Разработка через тестирование

Основными фазами модели можно считать:

- *«Вброс» архитектуры* – начальный этап проекта, на котором создается видение продукта, принимаются основные решения по архитектуре и применяемым технологиям. Результатом начального этапа является метафора (metaphor) системы, которая в достаточно простом и понятном команде виде должна описывать основной механизм работы системы.
- *Истории использования (User Story)* – этап сбора требований, записываемых на специальных карточках в виде сценариев выполнения отдельных функций.

Разработка через тестирование

Истории использования являются требованиями для планирования очередной версии и одновременной разработки приемочных тестов (Acceptance tests) для ее проверки.

- *Планирование версии (релиза).* Проводится на собрании с участием заказчика путем выбора User Stories, которые войдут в следующую версию. Одновременно принимаются решения, связанные с реализацией версии. Цель планирования - получение оценок того, что и как можно сделать за 1-3 недели создания следующей версии продукта.
- *Разработка* проводится в соответствии с планом и включает только те функции, которые были отобраны на этапе планирования.

Разработка через тестирование

- *Тестирование* проводится с участием заказчика, который участвует в составлении тестов.
- *Выпуск релиза* – разработанная версия передается заказчику для использования или бета-тестирования.

По завершению цикла делается переход на следующую итерацию разработки.

Особенности модели жизненного цикла XP проясняют следующие принципы этого метода. Прежде всего, это принципы «живой» (Agile) разработки ПО, зафиксированные в манифесте «живой» (Agile) разработки.

Разработка через тестирование

Кроме того, в XP есть несколько правил (техник), характеризующих особенности модели его жизненного цикла:

- *Живое планирование (planning game)* - как можно быстрее определить объем работ, который нужно сделать до следующей версии ПО. Решение принимается на основе, в первую очередь, бизнес-приоритетов заказчика и, во-вторую, технических оценок. Планы изменяются, как только они начинают расходится с действительностью или пожеланиями заказчика.
- *Частая смена версий (small releases)* - первая работающая версия должна появиться как можно быстрее, и тут же должна начать использоваться.

Разработка через тестирование

Следующие версии подготавливаются через достаточно короткие промежутки времени.

- *Простые проектные решения (simple design)* - в каждый момент времени система должна быть сконструирована так просто, насколько это возможно. Новые функции добавляются только после ясной просьбы об этом. Вся лишняя сложность удаляется, как только обнаруживается.
- *Разработка на основе тестирования* - сначала пишутся тесты, потом реализуются модули так, чтобы тесты срабатывали. Заказчики заранее пишут тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала.

Разработка через тестирование

- *Постоянная переработка (refactoring)* - системы для устранения излишней сложности, увеличения понятности кода, повышения его гибкости. При этом предпочтение отдается более элегантным и гибким решениям, по сравнению с просто дающими нужный результат.

- *Программирование парами (pair programming)* - весь код пишется двумя программистами на одном компьютере, что повышает его качество (отсутствие ошибок, понятность, читаемость, …).

Разработка через тестирование

- *Постоянная интеграция (continuous integration)* - система собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда пара программистов оканчивает реализацию очередной функции.

- *40-часовая рабочая неделя* - сверхурочная работа рассматривается как признак больших проблем в проекте. Не допускается сверхурочная работа 2 недели подряд - это истощает программистов и делает их работу значительно менее продуктивной.

Разработка через тестирование

TDD - Test Driven Development

Разработка на основе тестов основывается на повторении коротких циклов разработки: изначально пишется тест, покрывающий желаемое изменение, затем пишется программный код, который реализует желаемое поведение системы и позволит пройти написанный тест. Затем проводится рефакторинг написанного кода с постоянной проверкой прохождения тестов.

Взгляд на TDD со стороны разработчика: вы пишете один модульный тест, а затем разрабатываете минимально-достаточно производственного кода для выполнения этого теста.

Разработка через тестирование

Модульное тестирование фокусируется на каждой небольшой функциональности системы.

Эта методология позволяет добиться создания пригодного для автоматического тестирования приложения и очень хорошего покрытия кода тестами, так как ТЗ переводится на язык автоматических тестов, то есть всё, что программа должна делать, проверяется. Также TDD часто упрощает программную реализацию: исключается избыточность реализации — если компонент проходит тест, то он считается готовым.

Разработка через тестирование

Иногда процесс TDD называют красно-зеленым тестом:

1. Напишите тест.
2. Запустите тест. Тест пройдет неудачно, поскольку нет реализованного кода. КРАСНЫЙ
3. Напишите ровно столько кода реализации, чтобы тест прошел.
4. Запустите тест еще раз. Тест пройден. ЗЕЛЕНЫЙ
5. Рефакторинг. Шаг рефакторинга предназначен только для очистки кода и улучшения его качества. Обновлений для основных функций и поведения нет.
6. Повторить. Пока все тесты не пройдены, процесс повторяется.

Разработка через тестирование

Архитектура программных продуктов, разрабатываемых таким образом, обычно лучше (в приложениях, которые пригодны для автоматического тестирования, обычно очень хорошо распределяется ответственность между компонентами, а выполняемые сложные процедуры декомпозированы на множество простых).

Стабильность работы приложения, разработанного через тестирование, выше за счёт того, что все основные функциональные возможности программы покрыты тестами и их работоспособность постоянно проверяется.

Разработка через тестирование

Сопровождаемость проектов, где тестируется всё или практически всё, очень высока — разработчики могут не бояться вносить изменения в код, если что-то пойдёт не так, то об этом сообщат результаты автоматического тестирования.

Подробнее с принципами TDD вы можете ознакомиться, прочитав книгу Кента Бека "Экстремальное программирование. Разработка через тестирование".

ATDD - Acceptance TDD

ATDD сфокусирован на вопросе – делает ли решение то, что оно должно делать?

Разработка через тестирование

(Синонимом ATDD используют также термин Story Test Driven Development (STDD)).

Вы пишете один приемочный тест. Этот тест удовлетворяет требованиям спецификации или удовлетворяет поведению системы. После этого пишете достаточно производственного / функционального кода, чтобы выполнить этот приемочный тест. Приемочный тест фокусируется на общем поведении системы.

ATDD используется нетехнический язык и действует как интеграционный тест. Метод улучшает сотрудничество между разработчиками, пользователями и специалистами по контролю качества при определении критериев приемки.

Разработка через тестирование

ATDD можно считать командной работой. Для применения ATTD используются следующие методы:

- Анализ реальных сценариев.
- Определение критериев приемки тестовых сценариев.
- Фокус на разработку согласно принятым тест-кейсам.

Преимущества разработки через приемочное тестирование:

- Четкий анализ требований.
- Улучшение взаимодействия между пользователями и разработчиками, контроль качества.
- Руководство по всему процессу разработки.

Разработка через тестирование

Основная цель ATDD и TDD - определить подробные, выполнимые требования для вашего решения точно в срок (JIT). JIT означает принятие во внимание только тех требований, которые необходимы в системе, что повышает эффективность.

BDD - Behaviour Driven Development

Это разработка, основанная на описании поведения. Например, определенный человек (или люди) пишет описания вида "я как пользователь хочу чтобы когда нажали кнопку пуск тогда показывалось меню как на картинке".

Разработка через тестирование

Из-за некоторого методологического сходства TDD и BDD часто путают, в чем же отличие?

Концепции обоих подходов похожи, сначала идут тесты и только потом начинается разработка, но предназначение у них совершенно разное.

TDD — это больше о программировании и тестировании на уровне технической реализации продукта, когда тесты создают сами разработчики.

BDD предполагает описание тестировщиком или аналитиком пользовательских сценариев на естественном языке — если можно так выразиться, на языке бизнеса.

Разработка через тестирование

Тексты сценариев записываются в определенной форме. Заметим что ATDD и BDD используют схожий подход Given – When – Then (GWT). Given - описывает какой-то контекст, when - задаёт происходящее событие, then – представляет тестируемый результат. Пример BDD сценария:

Сценарий: На счету есть деньги

- **Given** счет с деньгами **AND** валидная карточка **AND** банкомат с наличными
- **When** клиент запрашивает наличные
- **Then** убедиться, что со счета было списание **AND** наличные выданы **AND** карточка возвращена.

Сравнение TDD, ATDD и BDD далее в таблице:

Разработка через тестирование

Parameters	TDD	BDD	ATDD
Definition	TDD is an approach of development that focuses on implementation of features	BDD is an approach of development that focuses on system behaviors.	ATDD is a technique that focuses defining accurate requirements.
Participants	Developers	Developers, Customers, QAs	Developers , Customers, QAs
Language	Testing language of used programming language	Simple native language	Simple native language
Main Focus	Unit test	Understanding requirements	Defining acceptance criteria.

Основы создания качественных программных систем

Качество ПО

Стандартизация качества ПО.

Стандартизация – деятельность, направленная на разработку и установление требований, норм, характеристик, как обязательных для выполнения, так и рекомендуемых, обеспечивающая право потребителя на приобретение товаров надлежащего качества, а также право на безопасность и комфортность труда.

Цель стандартизации – достижение оптимальной степени упорядочения посредством широкого и многократного использования установленных положений, требований, норм для решения реально существующих, планируемых или потенциальных задач.

Качество ПО

Основные результаты деятельности по стандартизации – повышение степени соответствия продукта (услуги), процессов их функциональному назначению, устранение технических барьеров в международном товарообмене, содействие научно-техническому прогрессу и сотрудничеству в различных областях.

Объект стандартизации – продукция, процесс, услуга, для которых разрабатывают требования, характеристики, параметры, правила.

Область стандартизации – совокупность взаимосвязанных объектов стандартизации.

Качество ПО

Стандарт – нормативно-технический документ, устанавливающий комплекс норм, правил и требований к объекту стандартизации, разработанный на основе согласия и на обобщенных результатах научных исследований, технических достижений и практического опыта, направленный на достижение оптимальной пользы для общества.

Уровень стандартизации – вид стандартизации, зависящий от географического, экономического, политического региона участников, принимающих участие в стандартизации.

Схема уровней стандартизации приведена на рисунке 7.1

Качество ПО

Рисунок 7.1 – Схема уровней стандартизации.

Качество ПО

Международная стандартизация – стандартизация, открытая для соответствующих органов любой страны.

Региональная стандартизация – стандартизация, открытая для соответствующих органов государств одного географического, политического или экономического региона.

Национальная стандартизация – стандартизация в одном государстве. Национальная стандартизация подразделяется на виды: государственная и отраслевая.

Внутрифирменная стандартизация – стандартизация внутри предприятия/фирмы..

Качество ПО

Внутрифирменные стандарты – набор внутрифирменных инструкций и руководств значительного объема, которые постоянно корректируются в целях совершенствования и изменения среды их применения.

Роль стандартизации в управлении качеством

При разработке программных средств для достижения заданных требований необходимо планирование и управление обеспечением качества в течение всего цикла создания программ.

Качество ПО

Для управления качеством требуется классифицировать критерии в зависимости от классов программ и связать их с методами и этапами разработки. Набор показателей качества программных средств зависит от функционального назначения и свойств каждого программного средства.

Критерий применяется, если определена метрика и указан способ измерения. Основной метод измерения качества программ является тестирование. На рисунке 7.2 приведена схема процесса управления качеством программных средств.

Рисунок 7.2 – Схема управления качеством ПС.

Качество ПО

Качество программных средств обеспечивается стандартами:

- международные (ISO, IEC, ECMA);
- национальные (ГОСТ, DOD, MIL, ANSI, IEEE).

Группы стандартов, регламентирующих качество программ:

- общие для любых программ;
- формализующие показатели качества программ;
- отражающие планирование, методы и технологию управления качеством программ;
- поддерживающие технологический процесс создания сложных программных средств высокого качества.

Качество ПО

Общие стандарты – стандарты, обеспечивающие качество любых программ независимо от их свойств, назначения и характеристик. Стандарты качества программных средств как объектов разработки и производства ориентируются в терминологии, структуре и содержании на общие стандарты.

Общие стандарты – международные стандарты «ISO 8402. Управление качеством и обеспечение качества – Словарь», «ISO 9000. Системы менеджмента качества. Основные положения и словарь», «ISO 9001. Системы менеджмента качества. Требования».

Качество ПО

Стандарты, формализующие показатели качества программ, – стандарты, формализующие номенклатуру, понятия и содержание показателей качества различных классов программ. Показатели качества строятся иерархически по уровням: факторы, критерии, метрики. Показатели качества программ ориентируются на сложные программные средства высокого качества и на выбор из них разработчиком конкретного набора в зависимости от требований заказчика.

Международные стандарты: «ISO/IEC 9126. Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению»,

Качество ПО

«ANSI/IEEE 729. Глоссарий стандартизированных терминов по технике разработки программного обеспечения»,

«ANSI/IEEE 1061. Система показателей качества программного обеспечения.», «ANSI/IEEE1044. Стандартная классификация программных ошибок, отказов и сбоев»

и отечественный стандарт

«ГОСТ 28195-89. Оценка качества программных средств. Общие положения» – стандарты, формализующие показатели качества программ.

Качество ПО

Стандарты, отражающие планирование, методы и технологию управления качеством программ, – стандарты, регламентирующие методы, технологию и документацию для планирования и управления обеспечением качества программных средств. Международные стандарты: «ISO/IEC 688. Управление передачей информации между фазами жизненного цикла программных средств», «ANSI/IEEE 1058 Планы управления проектами создания программного обеспечения»,

«DI-S-30567A. План разработки программ для ЭВМ», «ISO/IEC 687. Управление конфигурацией программного обеспечения», «ANSI/IEEE 828. План управления конфигурацией программного обеспечения».

Качество ПО

В международном стандарте ISO 9126:1991 при выборе показателей качества учитываются принципы:

- ясность и измеряемость значений;
- отсутствие перекрытия между показателями;
- соответствие понятиям и терминологии;
- возможность уточнения и детализации.

Шесть характеристик оценивают программные средства с позиции пользователя, разработчика и управляющего проектом.

Характеристики подразделяются на 21 субхарактеристику программных средств.

Качество ПО

В стандарте ГОСТ 28195-89 на первом уровне выделены 6 показателей качества: надёжность, корректность, удобство применения, эффективность, универсальность, сопровождаемость.

Показатели качества подразделяются на 19 критериев качества второго уровня.

Критерии качества детализируются 240 метриками и оценочными элементами, принимающими значения от 0 до 1. Показатели, критерии и метрики выбираются в зависимости от назначения, функций и этапов жизненного цикла программных средств.

Качество ПО

В стандарте ГОСТ 28806-90 формализуются общие понятия программы, программного средства, программного продукта и качества. Приводятся определения 18 наиболее употребляемых терминов, связанных с оценкой характеристик программ. Уточнены понятия базовых показателей качества, приведённых в ГОСТ 28195-89.

Международные стандарты качества, безопасности, тестирования, процессов жизненного цикла программных средств, документации при тестировании программ приведены в таблице 7.1.