2.1 Типы и виды тестирования

по доступу к исходному коду:

- ✔ **White box**
- ✔ **Grey box**
- ✔ **Black box**

White box. Тестировщик имеет доступ к исходному коду программ и может писать код, который связан с библиотеками тестируемого ПО. Это типично для юнит-тестирования, при котором тестируются только отдельные части системы.

Black box. При тестировании чёрного ящика, тестировщик имеет доступ к ПО только через интерфейсы, которые будут предоставлены заказчику.

Grey box. При тестировании серого ящика разработчик теста имеет доступ к исходному коду, но при непосредственном выполнении тестов доступ к коду, как правило, не требуется.

по запуску кода:

- ✔ Статическое
- ✔ Динамическое

При статическом тестировании программный код не выполняется — анализ программы происходит на основе исходного кода, который вычитывается вручную, либо анализируется специальными инструментами. Также к статическому тестированию относят тестирование требований, спецификаций, документации.

Остальное - это динамическое тестирование.

по степени автоматизации:

- ✔ ручное
- ✔ автоматизированное
- ✔ полуавтоматизированное
- ✔

по требованиям:

- ✔ позитивное
- ✔ Негативное
- ✔

по степени подготовленности:

- ✔ тестирование по документации
- ✔ интуитивное тестирование (ad hoc)

по объекту тестирования:

- ✔ функциональное
- ✔ нефункциональное

Каждое функциональное требование транслируется в тест-кейсы (используя техники «черного ящика») для того, чтобы проверить, что система функционирует в точности, как и описано в спецификации. Задача тестировщика - проверить, все ли функциональные требования действительно закодированы\реализованы.

Функциональное тестирование делится на несколько видов:

Smoke тестирование - это минимальный набор написанных тест-кейсов, определяющий, что билд готов к передаче в тестирование. Цель для команды тестирования – не нахождение дефектов, а убедиться, что вся функциональность работает стабильно и готова к тестированию. Занимает от 15 минут до 2х часов. Если не работают элементарные вещи, то билд отдают на доработку. Можно использовать средства автоматизаии.

Sanity тестирование заключается в том, чтобы проверить только исправленные дефекты, изменения из баг-трекинговой системы. Сосредоточен на узкой части функциональности.

Регрессионное тестирование (Regression testing) – повторное тестирование после внесение изменений в программное обеспечение или в его окружение (в новой версии приложения), чтобы убедиться, в том, что функции, которые работали в предыдущей версии системы, по-прежнему работают так, как ожидалось.

Альфа тестирование - это тестирование, обычно проводимое на ранней стадии разработки продукта и включающее имитацию реального использования продукта штатными разработчиками либо его реальное использование потенциальными клиентами.

Бета-тестировании - интенсивное использование почти готовой версии продукта с целью выявления максимального числа ошибок в его работе для их последующего устранения перед окончательным выходом (Релизом).

Нефункциональное тестирование также делится на несколько видов:

Тестирование производительности – тестирование поведение системы при различных нагрузках и при различных сценариях использования.

Основные виды тестирования производительности:

- Стрессовое тестирование (stress testing);
- Нагрузочное тестирование (Load testing)
- Тестирование стабильности (Stability testing)
- Объемное тестирование (Volume testing)

Стрессовое тестирование (Stress testing) – проверка системы при пиковых нагрузках, ограниченных ресурсах и восстановление после возвращению к нормальному состоянию.

Нагрузочное тестирование (Load testing) - проверка систем на различных уровнях нагрузки. Определяем, при какой максимальной нагрузке (максимальном количестве пользователей) система способна функционировать в соответствии с требованиями к производительности.

Тестирование стабильности (Stability testing) - оценка работоспособности системы при длительной нагрузке. Главная задача - выявить утечки памяти или другие проблемы, которые не позволяют системе стабильно работать.

Объемное тестирование (Volume testing) - тестирование проводится с увеличением не нагрузки и времени работы, а количества используемых данных, которые хранятся и используются в приложении.

При тестировании производительности нас интересует:

- изменение времени выполнения операций в зависимости от интенсивности операций (где интенсивность операций = кол-во пользователей * кол-во операций на единицу времени).
- определение границы приемлемой производительности (где приемлемая производительность - это либо четко прописанное в ТЗ среднее время отклика системы, либо такая скорость работы, когда уже с приложением нормально работать невозможно).
- определение количества пользователей, которые могут одновременно работать с приложением.

Тестирование интерфейса пользователя (UI testing) - тестирование графического интерфейса пользователя для того, чтобы убедиться, что он соответствует принятым стандартам и их требованиям.

Тестирование удобства использования (Usability testing) - тестирование, определяющее, насколько продукт отвечает требованиям той аудитории, для которой он пишется.

Обычно результатом выполнения UI и Usability тестов является список рекомендаций и предложений по улучшению.

Тестирование совместимости (compatibility testing) - проверить, что приложение совместимо с определенными конфигурациями оборудования, операционными системами, базами данных, браузерами и т.д. (Задание в конспекте).

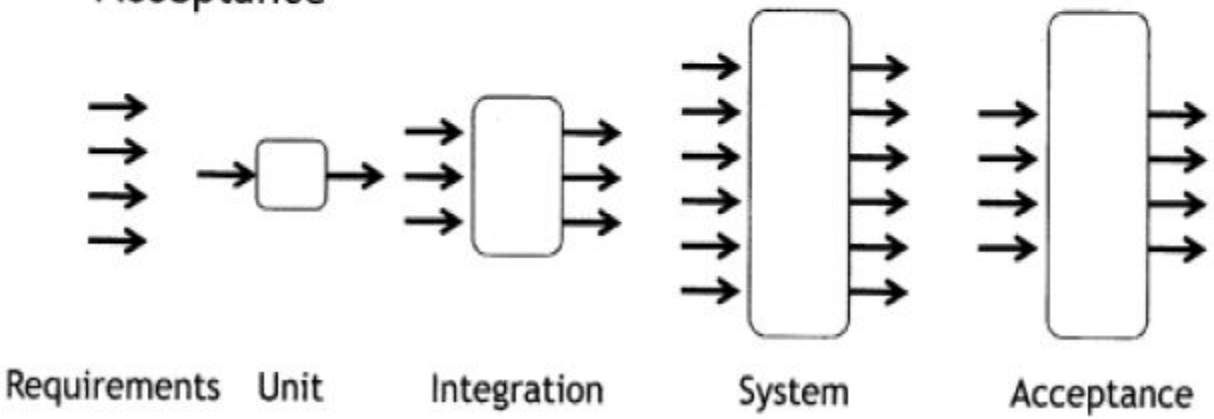
Тестирование безопастности (Sequrity Testing) – это стратегия тестирования, используемая для проверки безопаности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным. Основывается на трех основных принципах - это конфиденциальность, целостность и доступность.

Локализационное тестирование (Localization testing) - проверяет, правильно ли локализован продукт. То есть, переведен на другой язык и корректно работает с учетом национальных особенностей страны или региона, в котором будет продаваться и использоваться продукт.

2.2 Уровни тестирования

Применение

- Requirements
- Unit
- Integration
- System
- Acceptance



Модульное тестирование (Автономное или Unit-тестирование).

На данном уровне тестируются по отдельности небольшие элементы системы, максимально отделенные от других элементов и, в то же время, пригодные для тестирования. Такое тестирование обычно проводится сразу же вслед за разработкой каждого из элементов и направлено на оценку соответствия функциональности каждого из компонентов спроектированной “модели компонентов”.

Комплексное тестирование (Сборочное тестирование, integration testing)

На данном уровне тестируются объединенные элементы (компоненты или подсистемы) общей системы, чаще всего некоторая взаимодействующая между собой группа элементов.

Комплексное тестирование направлено не на проверку функционирования каждого из компонентов, а на проверку взаимодействия компонентов в соответствии с «Архитектурой системы».

Системное тестирование (system testing):

После того, как система собрана из составляющих компонентов, она должна быть протестирована на соответствие “Системным спецификациям” – реализованы ли все функциональные и нефункциональные требования к разрабатываемой системе.

На данном уровне тестируется приложение целиком. Системное тестирование выполняется через внешние интерфейсы программного обеспечения и тесно связано с тестированием пользовательского интерфейса (или через пользовательский интерфейс), проводимым при помощи имитации действий пользователей над элементами этого интерфейса.

Приемочное тестирование (Acceptance testing)

Тестирование готового продукта конечными пользователями на реальном окружении, в котором будет функционировать тестируемое приложение. Приемочные тесты разрабатываются пользователями, обычно, в виде сценариев.

2.3 Методологии разработки ПО

Модель жизненного цикла программного обеспечения — структура, содержащая процессы действия и задачи, которые осуществляются в ходе разработки, использования и сопровождения программного продукта.

- "Водопад" или каскадная модель
- "Водоворот" или каскадная модель с промежуточным контролем
- V модель - разработка через тестирование
- Спиральная модель
- Итеративная модель
- Семейство гибких методологий Agile: Scrum, XP, Kanban

"Водопад" или каскадная модель

Модель предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Требования, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта. Каждая стадия завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.


Выработка системных требований
Выработка требований к ПО
Анализ
Проектирование
Кодирование
Тестирование
Эксплуатация

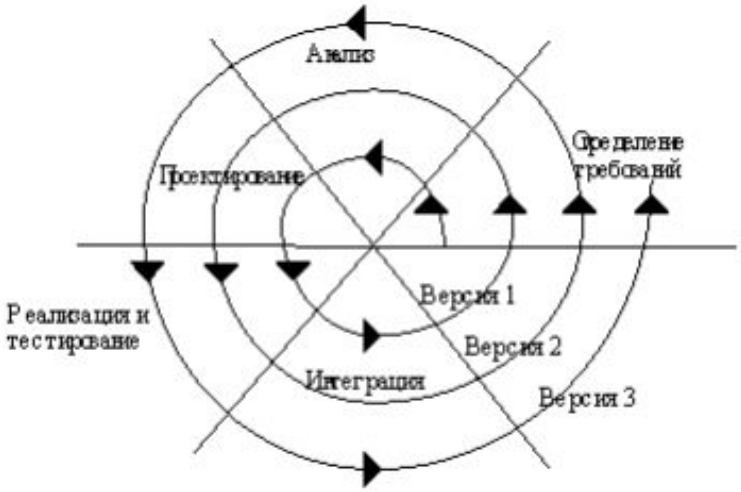
"Водоворот" или каскадная модель с промежуточным контролем

В этой модели предусмотрен промежуточный контроль за счет обратных связей.

V модель - разработка через тестирование

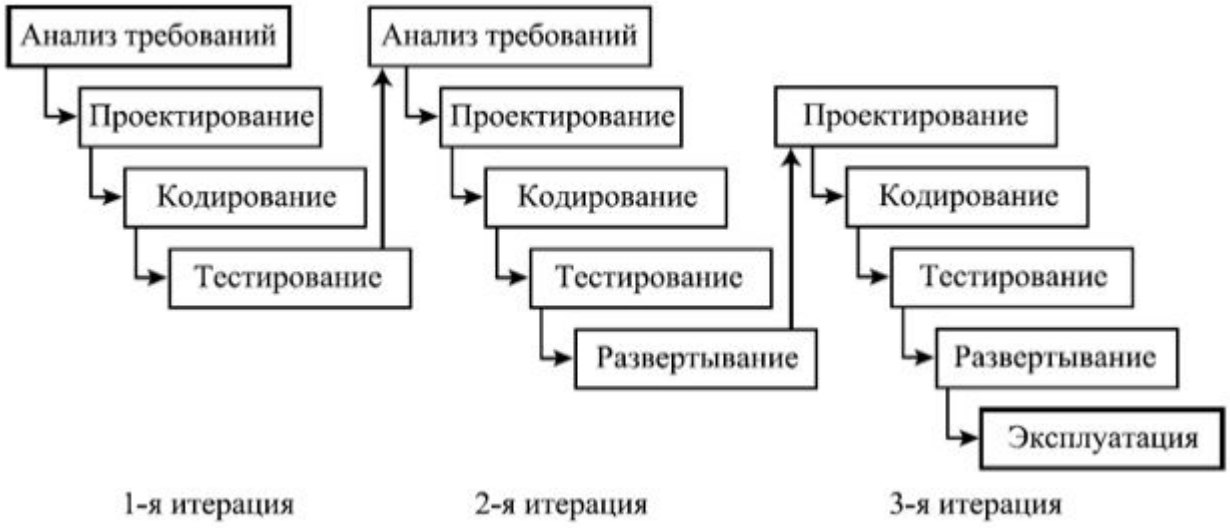
Спиральная модель

Общая идея спирального процесса заключается в том, чтобы на каждой итерации строить очередную версию программы, используя в качестве основы ее предыдущую версию. В этом случае процесс приобретает спиралевидный характер.

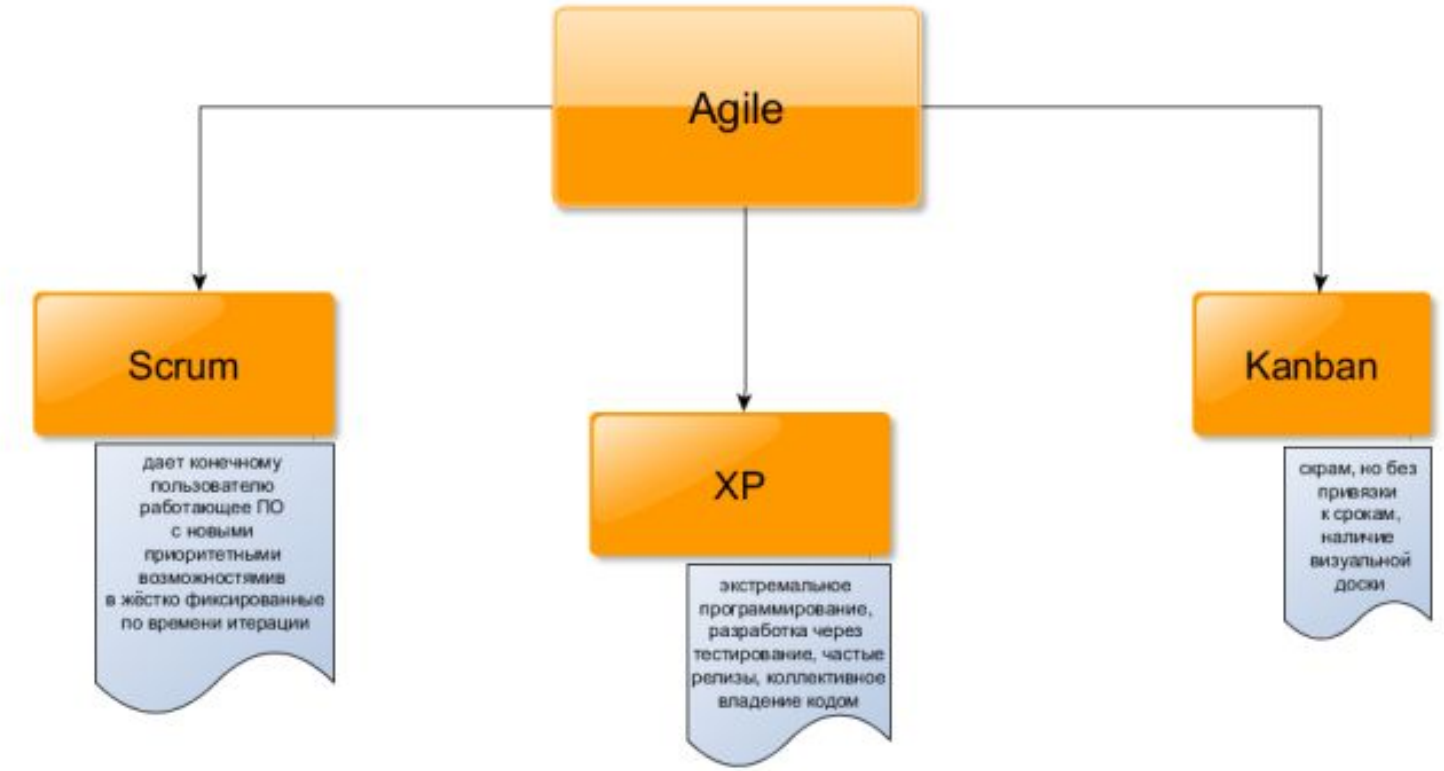


Итеративная модель

Итеративные или инкрементальные модели предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части.



Agile – семейство гибких методологий разработки.



Scrum - одна из самых популярных методологий гибкой разработки. Одна из причин ее популярности - простота.

В Scrum всего три роли:

- Scrum Master
- Product Owner
- Team

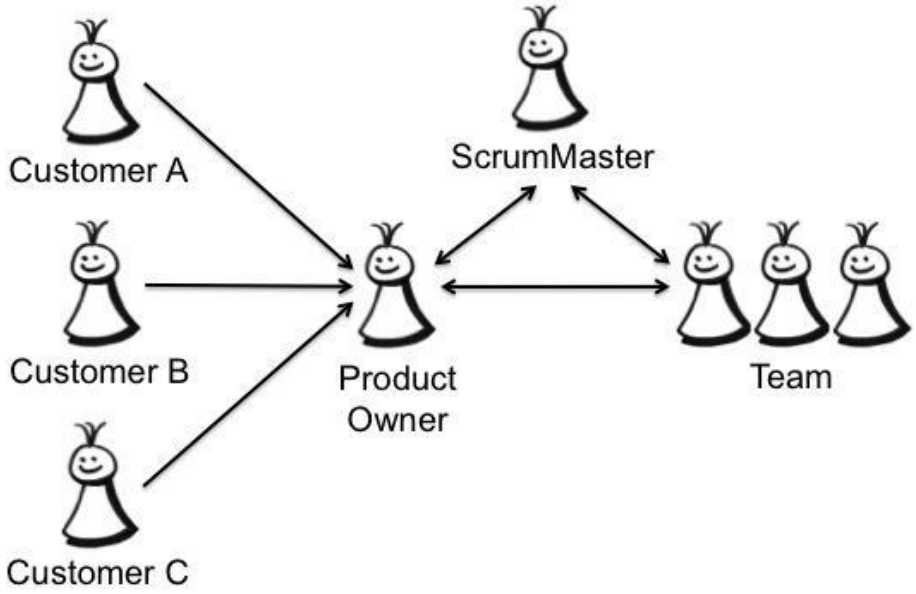
Скрам Мастер (Scrum Master) - самая важная роль в методологии. Скрам Мастер отвечает за успех Scrum в проекте. По сути, Скрам Мастер является интерфейсом между менеджментом и командой. В Agile команда является самоорганизующейся и самоуправляемой.

Основные обязанности Скрам Мастера таковы:

– Создает атмосферу доверия;
– Участвует в митингах;
– 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.

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. Единая точка принятия окончательных решений для команды в проекте.



Обязанности команды:

- ✔ 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.

Артефакты:

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

Product Backlog постоянно пересматривается и дополняется - в него включаются новые требования, удаляются ненужные, пересматриваются приоритеты. За Product Backlog отвечает Product Owner.

Sprint Backlog - содержит функциональность, выбранную Product Owner из Product Backlog. Все функции разбиты по задачам, каждая из которых оценивается командой.

В Scrum итерация называется **Sprint**. Ее длительность составляет 2-4 недели. Результатом Sprint является готовый продукт (build), который можно передавать (deliver) заказчику (по крайней мере, система должна быть готова к показу заказчику). В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта.

Планирование спринта (Sprint Planning Meeting) происходит в начале новой итерации Спринта. Выбираются задачи, обязательства по выполнению которых за спринт принимает на себя команда.

Ежедневное совещание (Daily Scrum meeting) . Длится не более 15 минут.

В течение совещания каждый член команды отвечает на 3 вопроса:

- Что сделано с момента предыдущего ежедневного совещания?
- Что будет сделано с момента текущего совещания до следующего?
- Какие проблемы мешают достижению целей спринта?

Ретроспективное совещание (Retrospective meeting) проводится после завершения спринта. Члены команды высказывают своё мнение о прошедшем спринте. Отвечают на два основных вопроса:

– Что было сделано хорошо в прошедшем спринте?

– Что надо улучшить в следующем?

Выполняют улучшение процесса разработки (решают вопросы и фиксируют удачные решения). Ограничена одним—тремя часами.