1. Тестирование программного обеспечения

Введение

**Тестирование программного обеспечения - это**

* Поиск ошибок
* Проверка соответствия ПО требованиям и здравому смыслу
* Оценка работоспособности ПО
* Способ контролировать качество ПО

**Причины ошибок в ПО:**

* Человеческий фактор
* Проблемы в описании требований к программному обеспечению
* Недостаток времени
* Недостаточно продуманная архитектура приложения
* Недостаточное знание бизнеса
* Нехватка профессиональных навыков и опыта
* Изменения «в последнюю минуту»

История тестирования ПО

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны.

Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов.

Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

**В 1960-х** много внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных.

Однако это невозможно:

#1 количество возможных входных данных очень велико;

#2 существует множество путей;

#3 сложно найти проблемы в архитектуре и спецификациях.

Итог: «исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

**В начале 1970-х годов** тестирование программного обеспечения обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения».

Впоследствии этот метод тестирования был признан неэффективным.

**Во второй половине 1970-х** тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает.

**Успешный тест — это тест, который обнаруживает ранее неизвестные проблемы.**

**В 1980-е годы** тестирование расширилось таким понятием, как предупреждение дефектов.

Стали высказываться мысли, что необходима методология тестирования, в частности, что тестирование должно включать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс.

В ходе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты.

**В начале 1990-х годов**в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества.

Начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования.

### Контроль и обеспечение качества

**Обеспечение качества** **(Quality Assurance - QA) - это**

совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения (ПО) информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО, для обеспечения требуемого уровня качества выпускаемого продукта.

**Контроль качества** **(Quality Control - QC) - это**

совокупность действий, проводимых над продуктом в процессе разработки, для получения информации о его актуальном состоянии в разрезах: "готовность продукта к выпуску", "соответствие зафиксированным требованиям", "соответствие заявленному уровню качества продукта".

**Тестирование программного обеспечения (Software Testing)** **- это**

одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

ГОСТ Р ИСО/МЭК 25010-2015

Информационные технологии (ИТ).

Системная и программная инженерия.

Требования и оценка качества систем и программного обеспечения (SQuaRE).

Модели качества систем и программных продуктов.

**Качество системы - это**

Это степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет, таким образом, оценить достоинства.

Эти заявленные и подразумеваемые потребности представлены в международных стандартах серии SQuaRE посредством моделей качества, которые представляют качество продукта в виде разбивки на классы характеристик, которые в отдельных случаях далее разделяются на подхарактеристики.

**Модели качества**

* Модель качества при использовании
* Модель качества продукта
* Модель качества данных ISO/IEC 25012:2008

**Модель качества при использовании**

* Эффективность (результативность)
* Производительность
* Удовлетворенность (полноценность, доверие, удовольствие, комфорт)
* Свобода от риска (смягчение отрицательных последствий:
  + экономического риска;
  + риска для здоровья и безопасности;
  + экологического риска.
* Покрытие контекста (полнота контекста, гибкость)

**Модель качества продукта**

* Функциональная пригодность:
  + функциональная полнота;
  + функциональная корректность;
  + функциональная целесообразность.
* Уровень производительности:
  + временные характеристики;
  + использование ресурсов;
  + потенциальные возможности.
* Совместимость:
  + сосуществование;
  + интероперабельность.
* Удобство использования:
  + определимость пригодности;
  + изучаемость;
  + управляемость;
  + защищенность от ошибки пользователя;
  + эстетика пользовательского интерфейса;
  + доступность.
* Надежность:
  + завершенность;
  + готовность;
  + отказоустойчивость;
  + восстанавливаемость.
* Защищенность:
  + конфиденциальность;
  + целостность;
  + неподдельность;
  + отслеживаемость;
  + подлинность.
* Сопровождаемость:
  + модульность;
  + возможность многократного использования;
  + анализируемость;
  + модифицируемость;
  + тестируемость.
* Переносимость:
  + адаптируемость;
  + устанавливаемость;
  + взаимозаменяемость.

## **Лекция №2.1 Модели разработки ПО**

### Введение

Чтобы лучше разобраться в том, как тестирование соотносится с программированием и иными видами проектной деятельности, для начала необходимо рассмотреть самые основы — модели разработки (lifecycle model) ПО (как часть жизненного цикла (software lifecycle) ПО). При этом разработка ПО является лишь частью жизненного цикла ПО, и здесь речь пойдет именно о разработке.

Материал данной лекции относится скорее к дисциплине «управление проектами», потому здесь рассмотрен крайне сжато.

Выбор модели разработки ПО влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д.

Моделей разработки ПО много, но в общем случае классическими можно считать водопадную, v-образную, итерационную инкрементальную, спиральную и гибкую.

Знать и понимать модели разработки ПО необходимо затем, чтобы уже с первых дней работы понимать, что происходит вокруг, что, зачем и почему тестировщик делает. Многие начинающие тестировщики отмечают, что ощущение бессмысленности происходящего посещает их, даже если текущие задания интересны. Чем полнее будет представление картины происходящего на проекте, тем яснее будет виден вклад тестировщика в общее дело.

Следует понимать, что никакая модель не является догмой или универсальным решением. Нет идеальной модели. Есть та, которая хуже или лучше подходит для конкретного проекта, конкретной команды, конкретных условий.

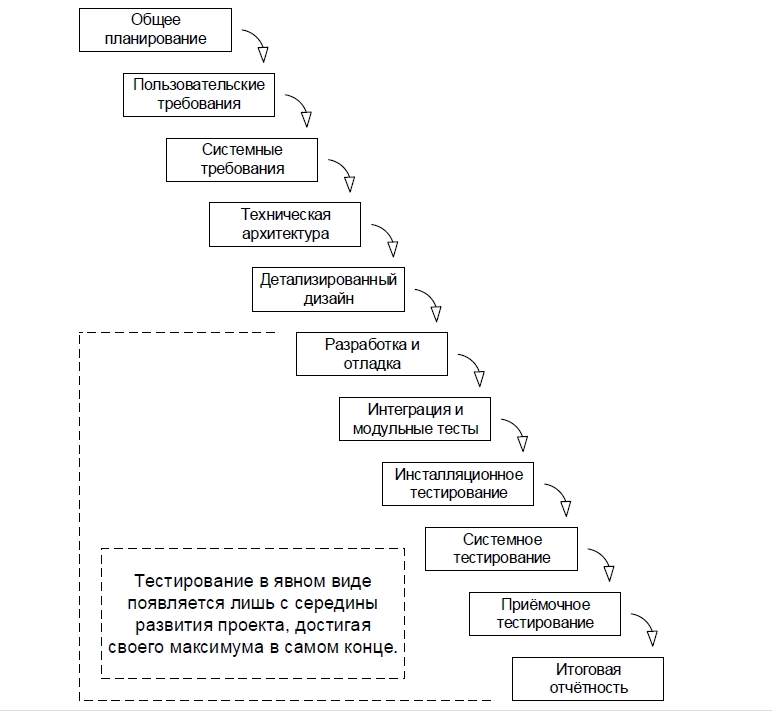
Начало формы

Водопадная модель

Конец формы

### Водопадная модель

Водопадная модель (waterfall model) сейчас представляет скорее исторический интерес, т.к. в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом.



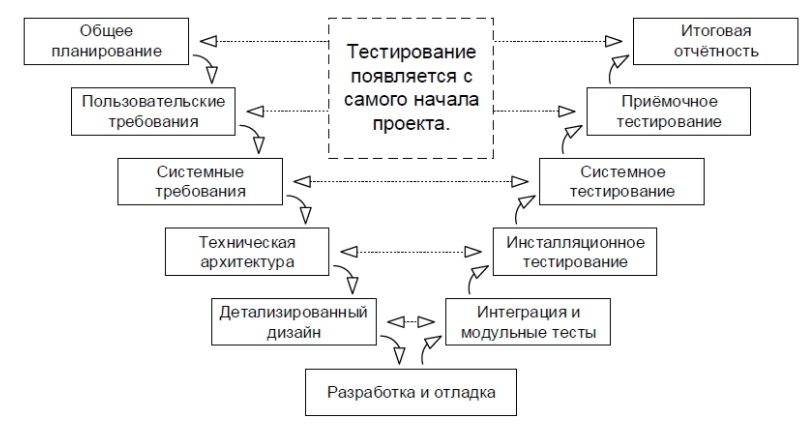
Очень упрощённо можно сказать, что в рамках этой модели в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке ПО приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.

К недостаткам водопадной модели принято относить тот факт, что участие пользователей ПО в ней либо не предусмотрено вообще, либо предусмотрено лишь косвенно на стадии однократного сбора требований. С точки зрения же тести-рования эта модель плоха тем, что тестирование в явном виде появляется здесь лишь с середины развития проекта, достигая своего максимума в самом конце.

Тем не менее водопадная модель часто интуитивно применяется при выполнении относительно простых задач, а её недостатки послужили прекрасным отправным пунктом для создания новых моделей. Также эта модель в несколько усовершенствованном виде используется на крупных проектах, в которых требования очень стабильны и могут быть хорошо сформулированы в начале проекта (аэрокосмическая область, медицинское ПО и т.д.)

### V-образная модель

V-образная модель (V-model) является логическим развитием водопадной. Как водопадная, так и v-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта.



Очень упрощённо можно сказать, что при использовании v-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

Хотя разновидностей v-образной модели существует много, все они используют четыре уровня тестирования, соответствующие четырем уровням разработки:

• Компонентное (модульное) тестирование

• Интеграционное тестирование

• Системное тестирование

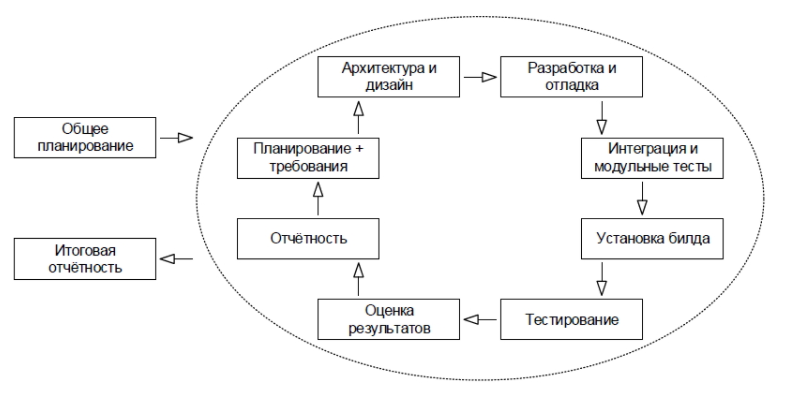
• Приемочное тестирование

На практике v-образная модель может иметь большее или меньшее количество уровней разработки и тестирования, все зависит от конкретного проекта и разрабатываемого продукта. Например, компонентное интеграционное тестирование может проводиться после компонентного, а системное интеграционное после системного. Артефакты программной разработки (такие как бизнес-сценарии или сценарии использования, технические требования, документы по дизайну и код) создающиеся во время процесса разработки часто используются для одного и более уровней тестирования. Ссылки на оригинальные рабочие артефакты можно найти в «Интегрированной модели зрелости процессов производства ПО» (CMMI) и «Жизненном цикле программного обеспечения» (IEEE/IEC 12207).  Верификация и валидация (а также ранняя разработка тестов) могут быть выполнены во время разработки ПО.

### Итерационная инкрементальная модель

Итерационная инкрементальная модель (iterative model, incremental model) является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность (а ISTQB-глоссарий даже не приводит единого определения, разбивая его на отдельные части):

* с точки зрения жизненного цикла модель является итерационной, т.к. подразумевает многократное повторение одних и тех же стадий;
* с точки зрения развития продукта (приращения его полезных функций) модель является инкрементальной.



Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям. Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде.

Длина итераций может меняться в зависимости от множества факторов, однако сам принцип многократного повторения позволяет гарантировать, что и тестирование, и демонстрация продукта конечному заказчику (с получением обратной связи) будет активно применяться с самого начала и на протяжении всего времени разработки проекта.

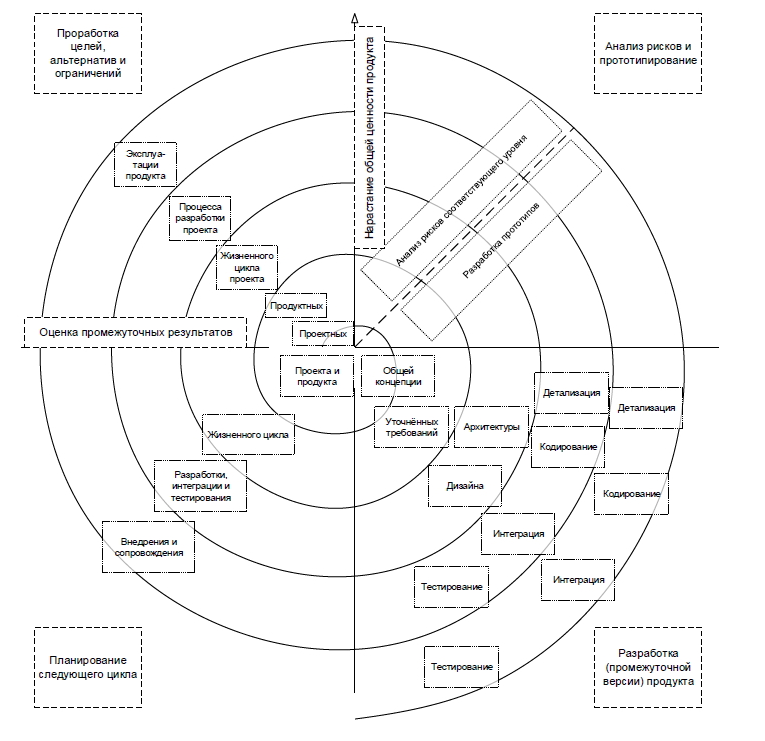
Во многих случаях допускается распараллеливание отдельных стадий внутри итерации и активная доработка с целью устранения недостатков, обнаруженных на любой из (предыдущих) стадий.

Итерационная инкрементальная модель очень хорошо зарекомендовала себя на объёмных и сложных проектах, выполняемых большими командами на протяжении длительных сроков. Однако к основным недостаткам этой модели часто относят высокие накладные расходы, вызванные высокой «бюрократизированностью» и общей громоздкостью модели.

### Спиральная модель

Спиральная модель (spiral model) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Схематично суть спиральной модели представлена на рисунке ниже.



Обратите внимание на то, что здесь явно выделены четыре ключевые фазы:

1. проработка целей, альтернатив и ограничений;
2. анализ рисков и прототипирование;
3. разработка (промежуточной версии) продукта;
4. планирование следующего цикла.

С точки зрения тестирования и управления качеством повышенное внимание рискам является ощутимым преимуществом при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом являются сложными и нестабильными (могут многократно меняться по ходу выполнения проекта).

Автор модели Barry Boehm подробно раскрывает эти вопросы и приводит множество рассуждений и рекомендаций о том, как применять спиральную модель с максимальным эффектом.

### Гибкая модель

Гибкая модель (agile model) представляет собой совокупность различных подходов к разработке ПО и базируется на т.н. «agile-манифесте»:

* Люди и взаимодействие важнее процессов и инструментов.
* Работающий продукт важнее исчерпывающей документации.
* Сотрудничество с заказчиком важнее согласования условий контракта.
* Готовность к изменениям важнее следования первоначальному плану.

Как несложно догадаться, положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в водопадной, v-образной, итерационной инкрементальной, спиральной и иных моделях. Причём здесь впервые был достигнут ощутимый результат в снижении бюрократической составляющей и максимальной адаптации процесса разработки ПО к мгновенным изменениям рынка и требований заказчика.

Очень упрощённо (почти на грани допустимого) можно сказать, что гибкая модель представляет собой облегчённую с точки зрения документации смесь итерационной инкрементальной и спиральной моделей при этом следует помнить об «agile-манифесте» и всех вытекающих из него преимуществах и недостатках.

Главным недостатком гибкой модели считается сложность её применения к крупным проектам, а также частое ошибочное внедрение её подходов, вызванное недопониманием фундаментальных принципов модели. Тем не менее можно утверждать, что всё больше и больше проектов начинают использовать гибкую модель разработки.

## **Лекция №2.2 Процесс тестирования ПО**

### Процесс тестирования ПО

* Планирование и управление
* Анализ и проектирование
* Внедрение и реализация
* Оценка критериев выхода и создание отчетов
* Действия по завершению тестов

#### Планирование и управление

**Планирование тестирования** - это действия, направленные на определение целей тестирования и описание задач тестирования для достижения этих целей и миссии.

**Управление тестированием** - это постоянное сопоставление текущего положения дел с планом и отчетность о состоянии дел, включая отклонения от плана.

**Планирование тестирования ПО:**

* Анализ требований;
* Определение целей тестирования;
* Определение общего подхода к тестированию ([уровни тестирования](https://dist.belstu.by/mod/assign/view.php?id=57433), виды тестирования, критерия входа в тестирование);
* Интегрирование с разработкой ПО (требования, архитектура, дизайн, разработка, тестирование, релиз);
* Решение, какие роли нужны для выполнения тестирования, когда и как проводить тестирование и как оценивать результаты;
* Составление графика тестирования;
* Определение шаблонов для тестовой документации;
* Выбор метрик для мониторинга и контроля подготовки и проведения тестирования, исправления дефектов, проблем и рисков.

**Критерии входа в тестирование**определяют, когда нужно начинать тестирование. Примеры:

* Готовность и доступность тестового окружения;
* Готовность средства тестирования в окружении;
* Доступность тестируемого кода;
* Доступность тестовых данных.

#### Анализ и проектирование тестов

Это деятельность, во время которой общие цели тестирования материализуются в тестовые условия и тестовые сценарии.

**Активности на данном этапе процесса тестирования:**

* Рецензирование базиса тестирования:
  + функциональных и нефункциональных требований;
  + архитектуры;
  + дизайна;
  + технических требований к интерфейсу;
* Оценка тестируемости базиса тестирования и объектов тестирования;
* Идентификация и расстановка приоритетов тестирования;
* Выявление необходимых данных для поддержки тестовых условий и тестовых сценариев;
* Проектирование и установка тестового окружения и выявление необходимой инфраструктуры и инструментов;
* Создание двунаправленной трассируемости между тестовым базисом и тестовым сценарием.

#### Внедрение и реализация тестов

Это деятельность, где процедуры тестирования или автоматизированные сценарии задаются последовательностью тестовых сценариев, а также собирается любая информация, необходимая для выполнения тестов, разворачивается окружающая среда, и запускаются тесты.

**Активности на данном этапе процесса тестирования:**

* Завершение, реализация и расстановка приоритетов тестовых сценариев (включая проектирование тестовых данных);
* Написание автоматизированных сценариев тестирования;
* Проверка правильности настройки тестового окружения;
* Проверка и обновление двунаправленной трассируемости между тестовым базисом и тестовым сценарием;
* Выполнение процедур тестирования либо вручную, либо используя инструменты выполнения тестов, согласно заданному плану;
* Регистрация результатов выполнения тестов;
* Сравнение фактических и ожидаемых результатов;
* [Оформление отчетов об ошибках](https://dist.belstu.by/mod/assign/view.php?id=57431) и занесение их баг-трекинговую систему;
* Повторное тестирование областей, где были исправлены ошибки и областей, где могут появится новые ошибки после исправления уже известных ошибок (регрессионное тестирование).

#### Оценка критериев выхода и создание отчетов

Это деятельность, где выполнение тестов оценивается согласно определенным целям. Она должна быть выполнена для каждого уровня тестирования.

**Активности на данном этапе процесса тестирования:**

* Сверка протокола тестирования в сравнении с критериями выхода, определенными в плане тестирования;
* Анализ необходимости использования дополнительных тестов или изменения критериев выхода;
* Написание итогового отчета о тестировании для заинтересованных лиц.

**Критерии выхода**определяют, когда нужно заканчивать тестирование. Примеры: 

* Степень покрытие кода, функциональности или рисков тестами;
* Оценку плотности дефектов или измерение надежности;
* Стоимость;
* Остаточные риски (неисправленные дефекты или недостаток тестового покрытия какой-либо области);
* План, основанный на времени выхода ПО на рынок.

#### Действия по завершению тестов

Это сбор данных о завершенных испытаниях для объединения опыта, тестового обеспечения, фактов и цифр. Проводится после релиза.

**Активности на данном этапе процесса тестирования:**

* Проверка, что запланированные результаты достигнуты;
* Завершение и архивирование тестового обеспечения, тестового окружения и инфраструктуры тестирования для последующего использования;
* Передача тестового обеспечения организации сопровождения;
* Анализ полученных уроков для повышения зрелости процесса тестирования.

#### Процесс разработки ПО:

* Анализ требований
* Разработка архитектуры
* Проектирование дизайн
* Имплементация
* Отладка
* Инсталляция приложения на Production

| **Соотношение процессов разработки и тестирования** | |
| --- | --- |
| **Процесс разработки** | **Процесс тестирования** |
| 1. Анализ требований | 1. Анализ требований |
| 2. Разработка архитектуры | 2. Планирование тестирования |
| 3. Проектирование дизайн | 3. Анализ и проектирование тестов |
| 4. Имплементация | 4. Реализация и выполнение тестов |
| 5. Отладка | 5.1. Реализация и выполнение тестов 5.2. Оценка критериев выхода и отчетность |
| 6. Инсталляция приложения на Production | 6. Действия по завершению тестирования |
| **Сравнение процессов контроля и обеспечения качества** | |
| **Процесс контроля качества** | **Процесс обеспечения качества** |
| * Планирование и управление * Анализ и проектирование * Внедрение и реализация * Оценка критериев выхода и создание отчетов * Действия по завершению тестов | * [Тестирование требований](https://dist.belstu.by/mod/assign/view.php?id=57437) * Реализация unit-тестов * Реализация интеграционных тестов * Code review * Интеграция фичи в систему * Alpha тестирование * Отладка * Beta тестирование |

### Уровни тестирования

* Компонентное тестирование
* Интеграционное тестирование
* Системное тестирование
* Приемочное тестирование

**Компонентное тестирование**

Компонентное тестирование (также известное как модульное) занимается поиском дефектов и верификацией функционирования программных модулей, программ, объектов, классов и т.п., которые можно протестировать изолированно. Это может быть сделано изолированно от остальной части системы, в зависимости от контекста ЖЦ разработки и системы. В процессе могут быть использованы заглушки, драйвера и эмуляторы.

Компонентное тестирование может включать как тестирование функциональности и специфичных нефункциональных характеристик, таких как поведение ресурсов (например, поиск утечки памяти) или тестирование надежности, так и структурное тестирование (например, покрытие кода). Тестовые сценарии разрабатываются на основе артефактов процесса разработки, таких как спецификация компонентов, дизайн или модель БД.

Обычно, компонентное тестирование производится с доступом к тестируемому коду и с поддержкой рабочего окружения, такого как фреймворк модульного тестирования или утилиты отладки. На практике компонентное тестирование обычно производится разработчиками, которые пишут код. Дефекты обычно исправляются сразу после того, как становятся известны, без занесения их в базу дефектов.

Один из подходов к компонентному тестированию – составить автоматизированные тестовые сценарии до кодирования. Это называется

разработкой, управляемой тестированием. Этот подход состоит из множества итераций и основывается на циклах разработки тестовых сценариев, написании и интеграции небольшого участка кода и выполнении компонентного тестирования, корректируя любые проблемы и выполняя тесты, пока не будет получен положительный результат.

**Интеграционное тестирование**

Интеграционное тестирование проверяет интерфейсы между компонентами, взаимодействие различных частей системы, таких как операционная системы, файловая система, аппаратное обеспечение, и интерфейсы между системами. Интеграционное тестирование может состоять из одного или более уровней и может быть выполнено на тестовых объектах разного размера следующим образом:

1. Компонентное интеграционное тестирование проверяет взаимодействие между программными компонентами и производится после компонентного тестирования.

2. Системное интеграционное тестирование проверяет взаимодействие между системами или между аппаратным обеспечением и может быть выполнено после системного тестирования. В этом случае, разработчики могут управлять только одной стороной интерфейса. Однако, это может рассматриваться как риск. Бизнес-процессы могут включать последовательность систем; могут быть важны кроссплатформенные различия.

Чем больше объем интеграции, тем труднее становится изолировать дефекты отдельного компонента или системы, которые могут привести к увеличению рисков и требующих дополнительного времени для решения проблем. Стратегии системного интеграционного тестирования могут основываться на архитектуре системы (такой как нисходящая или восходящая), функциональных задачах, последовательности обработки транзакций или других аспектах системы и ее компонентов. Для того, чтобы упростить процесс изоляции отказа и как можно раньше обнаруживать дефекты, интеграция должна проводиться по возрастающей, а не происходить по сценарию «большого взрыва».

Тестирование специфичных нефункциональных характеристик ( например, производительности), может быть включено в интеграционное тестирование, наравне с функциональным.

На каждой стадии интеграции тестировщики концентрируют все внимание именно на интеграции как таковой. Например, если интегрируется модуль А с модулем В, они проверяют взаимодействие модулей, а не функциональность каждого из них, т.к. она должна быть проверена во время компонентного тестирования. Для тестирования могут использоваться как функциональный, так и структурный подходы.

В идеале, тестировщики должны понимать архитектуру и ее влияние на интеграционное планирование. Если интеграционное тестирование планируется до разработки компонентов или системы, эти компоненты могут разрабатываться в порядке, обеспечивающем наиболее эффективное тестирование.

**Системное тестирование**

Системное тестирование сконцентрировано на поведении тестового объекта как целостной системы или продукта. Область тестирования должна быть четко определена в главном плане тестирования либо плане тестирования для конкретного уровня тестирования.

Во время системного тестирования тестовое окружение должно быть как можно ближе к предполагаемому эксплуатационному окружению системы для минимизации риска пропуска отказов, связанных с эксплуатационным окружением системы.

Системное тестирование может включать тесты, основанные на рисках или спецификациях требований, бизнес-процессах, сценариях использование системы, или других высокоуровневых текстовых описаниях или моделях поведения системы, взаимодействия с ОС и системными ресурсами.

Системное тестирование должно заниматься исследованием функциональных и нефункциональных требований к системе и качеством обрабатываемых данных. Тестировщики также должны уметь выполнять свои обязанности в случае неполных или недокументированных требований. Системное тестирование функциональных требований начинается с тестирования на основе спецификаций (тестирования методом черного ящика), для различных аспектов системы. Например, таблица решений может быть создана на основе бизнес-правил работы системы. Тестирование на основе структуры (тестирование методом белого ящика) может использоваться для оценки тщательности тестирования того или иного структурного элемента, такого как структура меню или навигация веб-страницы.

**Приемочное тестирование**

Приемочным тестированием системы чаще всего занимаются заказчики или пользователи системы, а также другие заинтересованные лица.

Основная цель приемочного тестирования – проверка работоспособности системы, частей системы или отдельных нефункциональных характеристик системы. Поиск дефектов не является главной целью приемочного тестирования.

Приемочное тестирование оценивает готовность системы к развертыванию и использованию, хотя это не обязательно самый последний уровень тестирования. Например, крупномасштабные тесты по системной интеграции можно провести именно во время приемочного тестирования системы.

Приемочное тестирование может проводиться в различные моменты ЖЦ разработки, например:

• Для коробочного продукта приемочное тестирование можно провести при установке или интеграции

• Приемочное тестирование удобства использования компонента можно провести во время компонентного тестирования

• Приемочное тестирование новой функциональности можно проводить до системного тестирования

**Типичные виды приемочного тестирования:**

* *Пользовательское приемочное тестирование*

Обычно проверяет готовность системы для использования в бизнесе.

* *Эксплуатационное (приемочное) тестирование*

Приемочное тестирование, проводимое системными администраторами, включает:

* Тестирование резервного копирования \ восстановления
* Восстановление после сбоев
* Управление пользователями
* Задачи сопровождения
* Задачи загрузки и миграции данных
* Периодическая проверка уязвимостей системы
* *Контрактное и правовое приемочное тестирование*

Контрактное приемочное тестирование выполняется для проверки требований, предъявляемых контрактом в к разрабатываемому ПО. Критерий приема должен быть определен непосредственно в контракте. Приемочное тестирование на соответствие стандартам выполняется для проверки соответствия стандартам государственным, юридическим или стандартам безопасности.

* *Альфа и бета тестирование (или тестирование в условиях эксплуатации)*

Разработчики рыночного, или коробочного, ПО часто хотят получить отзывы от потенциальных или существующих заказчиков до того, как начнется продажа продукта. Альфа тестирование выполняется организацией, разрабатывающей продукт, но не группой разработчиков. Бета тестирование, или тестирование в условиях эксплуатации, выполняется покупателями или потенциальными заказчиками на их собственных мощностях.

| **Верификация и валидация** | |
| --- | --- |
| **Верификация** | **Валидация** |
| * Компонентное тестирование * Интеграционное тестирование * Системное тестирование | * Приемочное тестирование |