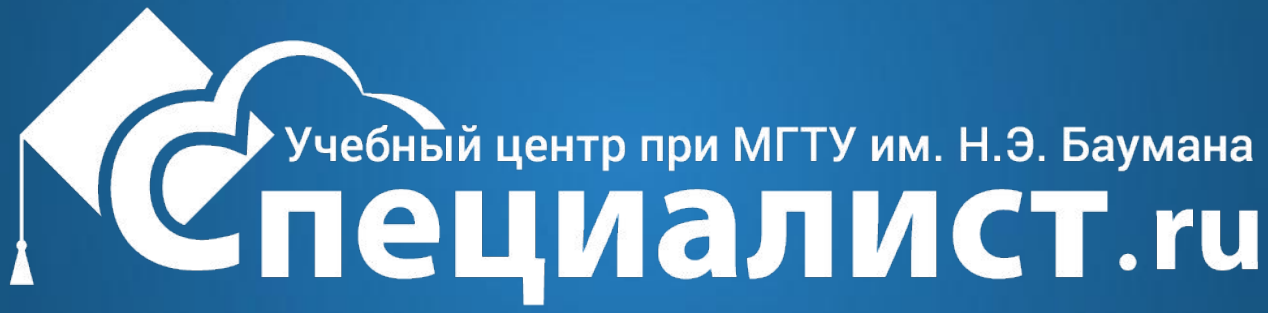


Тестирование ПО

Уровень 1. Тестировщик программного обеспечения

Модуль 1

Введение в тестирование программного обеспечения

Содержание

- Зачем нужно тестировать программы?
- Понятие качество ПО. Стандарты качества ПО.
- Атрибуты и характеристики качества ПО.
- Основные определения тестирования.
- Жизненный цикл ПО.
- Методологии разработки ПО.
- Жизненный цикл тестирования ПО.
- Принципы тестирования.
- Команда тестирования.

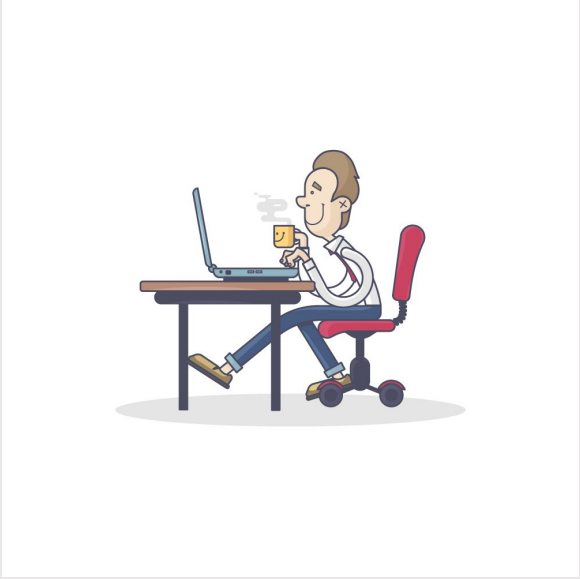
Зачем нужно тестировать?

Программное обеспечение, содержащее ошибки и недоработки ведёт

- Финансовым затратам
- Потере времени
- Ущербу деловой репутации
- Травмам или смерти

Стандарты качества ПО

Стандарт – набор правил и требований, предназначенных для обеспечения правильности действий всех организаций, которые выполняют описанные в стандартах процессы.



Стандарты качества ПО

- **IEEE 610** Computer dictionary, compilation of computer glossaries
- **IEEE 829** Standard for Software Test Documentation – описывает виды документов, служащих для подготовки тестов;

Стандарты качества ПО

- **ISO/IEC 9126-1** Программирование. Качество продукта. Модель качества
- **ISO/IEC 9126-2** Внешние метрики качества
- **ISO/IEC 9126-3** Внутренние метрики качества
- **ISO/IEC 9126-4** Метрики качества в использовании

Стандарты качества ПО

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

- **ISO/IEC 25010:2011** Модели качества систем и программного обеспечения
- **ISO/IEC 25021:2012** Элементы показателя качества
- **ISO/IEC 25020:2019** Основные принципы измерения характеристик качества

Стандарты качества ПО

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

- **ISO/IEC 25030:2019** Структура требований к качеству
- **ISO/IEC 25023:2016** Определение качества систем и программного продукта
- **ISO/IEC 25022:2016** Определение качества при использовании

Стандарты качества ПО

Тестирование программного обеспечения.

- **Часть 1.** Понятия и определения (ISO/IEC/IEEE 29119-1:2013 - ГОСТ Р 56920-2016)
- **Часть 2.** Процессы тестирования (ISO/IEC/IEEE 29119-2:2013 - ГОСТ Р 56921-2016)
- **Часть 3.** Документация для тестирования (ISO/IEC/IEEE 29119-3:2013 - ГОСТ Р 56922-2016)
- **Часть 4.** Методы тестирования (ISO/IEC/IEEE 29119-4:2015
- **Часть 5.**Тестирование на основе ключевого слова (ISO/IEC/IEEE 29119-5:2016)

Понятие качества ПО

Качество системы — это степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет, таким образом, оценить достоинства. (ISO/IEC 25010:2011)

Примеры заинтересованных лиц: разработчики, приобретатели, пользователи или клиенты

Типы пользователей

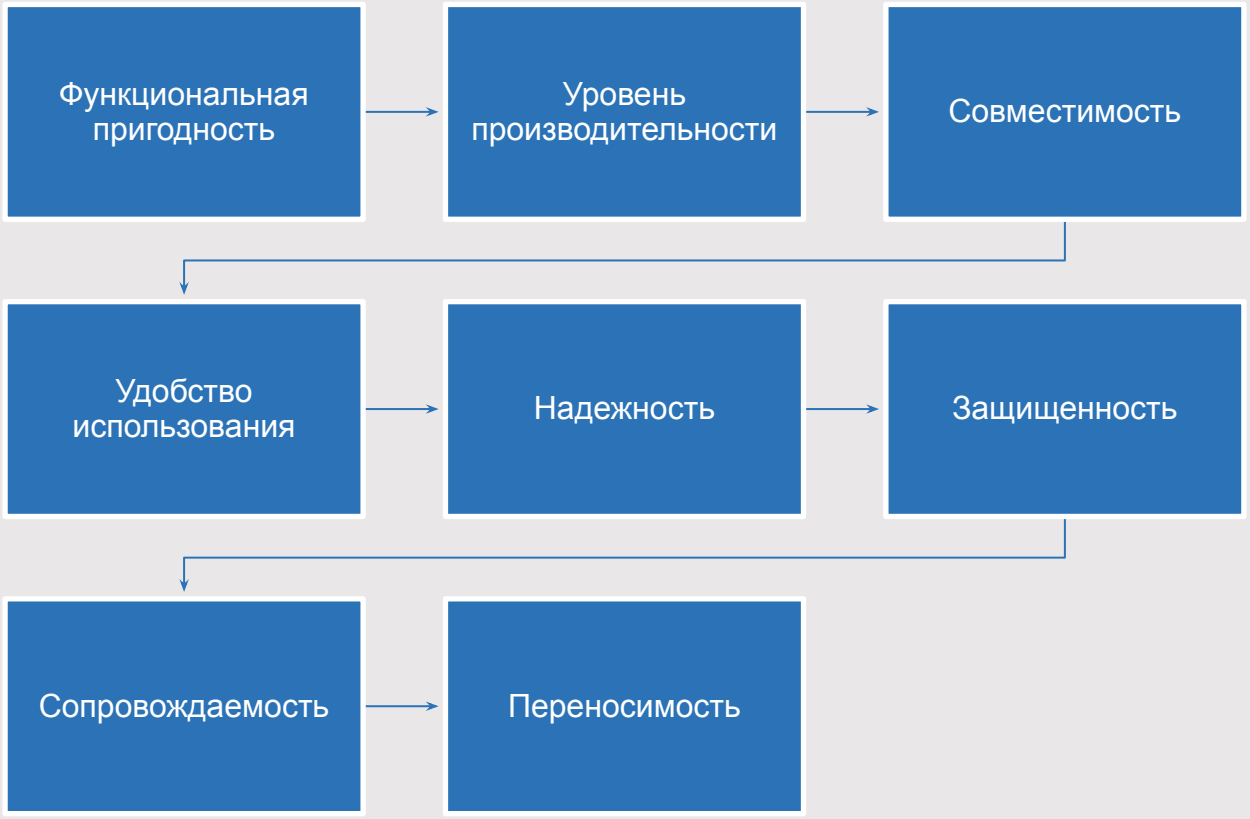
Основной пользователь — лицо, взаимодействующее с системой для достижения основных целей.

Вторичные пользователи — лица, осуществляющие поддержку (провайдер контента, системные, специалист по обслуживанию)

Косвенный пользователь— лицо, которое получает результаты, но не взаимодействует с системой.

Модель качества продукта

ISO/IEC 25010:2015



Применение модели качества

- Для определения требований, выработки показателей и выполнения оценки качества
- Для использования в качестве набора при спецификации или оценке качества ПО

Атрибуты и характеристики качества ПО

Функциональная пригодность

1. Функциональная полнота
2. Функциональная корректность
3. Функциональная целесообразность

Уровень производительности

1. Временные характеристики
2. Использование ресурсов
3. Потенциальные возможности

Атрибуты и характеристики качества ПО

Совместимость

1. Сосуществование
2. Функциональная совместимость

Удобство использования

1. Определяемость пригодности
2. Изучаемость
3. Управляемость
4. Защищенность от ошибки
5. Эстетика пользовательского интерфейса
6. Доступность

Атрибуты и характеристики качества ПО

Надежность

1. Завершенность
2. Готовность
3. Отказоустойчивость
4. Восстанавливаемость

Атрибуты и характеристики качества ПО

Защищенность

1. Конфиденциальность
2. Целостность
3. Неподдельность
4. Отслеживаемость
5. Подлинность

Атрибуты и характеристики качества ПО

Сопровождаемость, модифицируемость

1. Модульность
2. Возможность многократного использования
3. Анализируемость
4. Модифицируемость
5. Тестируемость

Атрибуты и характеристики качества ПО

Переносимость , мобильность

1. Адаптируемость
2. Устанавливаемость
3. Взаимозаменяемость

Основные определения

Контроль качества (Quality control) рабочие методы и активности, нацеленные на выполнение требований к качеству, являющиеся частью управления качеством.

Обеспечение качества (Quality assurance) процесс или результат формирования требуемых свойств и характеристик продукции по мере ее создания, а также поддержание этих характеристик при хранении, транспортировки и эксплуатации продукции.

Основные определения

Валидация (validation) – ожидания и потребности пользователя

Верификация (verification) – наши цели, сроки, задачи по разработке проекта

Основные определения

Объект тестирования (test object) компонент или система, которые должны быть протестированы

Базис тестирования (test basis) документ, на основании которого определяются требования к компоненту или системе. Документация, на которой базируются тестовые сценарии

Основные определения

Инфраструктура тестирования (test infrastructure) Артефакты, необходимые для проведения тестирования, такие как тестовое окружение, инструменты тестирования, офисное окружение и процедуры

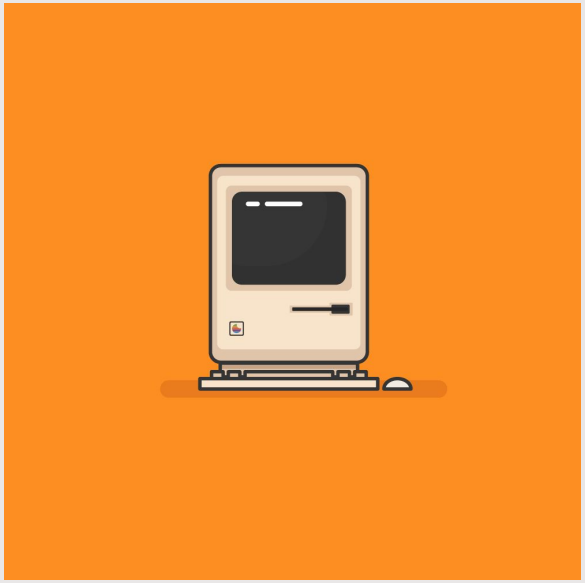
Тестовое окружение (test environment)

Окружение, включающее в себя аппаратное обеспечение, измерительную аппаратуру, имитаторы, программный инструментарий и прочие инструменты, необходимые для проведения теста. [IEEE 610]

Первый баг

9 сентября 1947 года официально был зарегистрирован **первый** в истории баг.

День тестировщика — профессиональный день тестировщиков



Основные определения

- Разработки (dev)
- Тестовая (test)
- Промежуточная (stage)
- Демо (demo)
- Продакшен (prod)

Каждой версии программы присваивают номер. Например, 9.3 и 10.0

Основные определения

Тестирование ПО – процесс исследования, имеющий своей целью проверку соответствия между реальным поведением программы и ее ожидаемым поведением на **конечном наборе** тестов, выбранных определенным образом

Основные определения

Тестирование ПО должно быть направлено на **предоставление информации** о программном продукте
и нахождение максимально возможного числа дефектов **на возможно ранних этапах** процесса разработки
при заданных ограничениях стоимости и графика разработки.

Цели процесса тестирования

1. Предоставление информации о качестве элемента тестирования и любых остаточных рисках относительно того, до какой степени элемент тестирования был проверен;
2. Обнаружение дефектов в элементе тестирования до его передачи в эксплуатацию;
3. Смягчение рисков получения продукта низкого качества заинтересованными сторонами.

ISO/IEC/IEEE 29119-1:2013 - ГОСТ Р 56920-2016

Жизненный цикл ПО

Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации

Цикл разработки облегчает проектирование, создание и выпуск программного обеспечения.

Жизненный цикл ПО

1) Анализ требований
2) Проектирование
3) Кодирование (программирование)
4) Тестирование и отладка
5) Внедрение
6) Эксплуатация и поддержка

Модель жизненного цикла ПО

Модель жизненного цикла ПО —
структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении жизненного цикла.

Виды моделей жизненного цикла ПО

- Code and fix — модель кодирования и устранения ошибок;
- Waterfall – каскадная модель;
- V-model;
- Iteration model - итеративная модель;
- Incremental model– инкрементальная модель;
- Spiral – спиральная модель;

Code-and-Fix

Code-and-Fix или Build-and-Fix, кодируй и фиксируй.

- Модель проб и ошибок.
- Строим продукт заново каждый раз до тех пор, пока клиент не будет доволен, не будет удовлетворен.
- Сколько раз придется нам как разработчикам этот цикл разработки повторять? Сколько раз нам придется это делать до того, пока заказчик не будет удовлетворен? Существенная сложность и существенная проблема данного подхода.

Каскадная модель

Каскадная модель (waterfall model) — модель процесса разработки программного обеспечения, жизненный цикл которой выглядит как поток, последовательно проходящий фазы анализа требований, проектирования. реализации, тестирования, интеграции и поддержки.

Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом

Каскадная модель

V-model

V образная модель (V-model) – является развитием водопадной модели

Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять.

V-model


Планирование проекта и требований
Производство, эксплуатация и сопровождение
Анализ требований продукта и спецификаций
Системное и приемочное тестирование
Разработка архитектурного проекта на высшем уровне
Интеграция и тестирование
Детализированная разработка проекта
Модульное тестирование
Кодирование

V-model


Бизнес-требования
Приемочное тестирование
Функциональные требования
Функциональное тестирование
Архитектура системы
Интеграционное тестирование
Архитектура компонентов
Модульное тестирование
Разработка

Итеративная модель

Итеративный подход (*iteration* - «повторение»)
- выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы.

Проект при этом подходе в каждой фазе развития проходит повторяющийся цикл PDCA:

Планирование — Реализация — Проверка — Оценка (*plan-do-check-act cycle*).

Итеративная модель

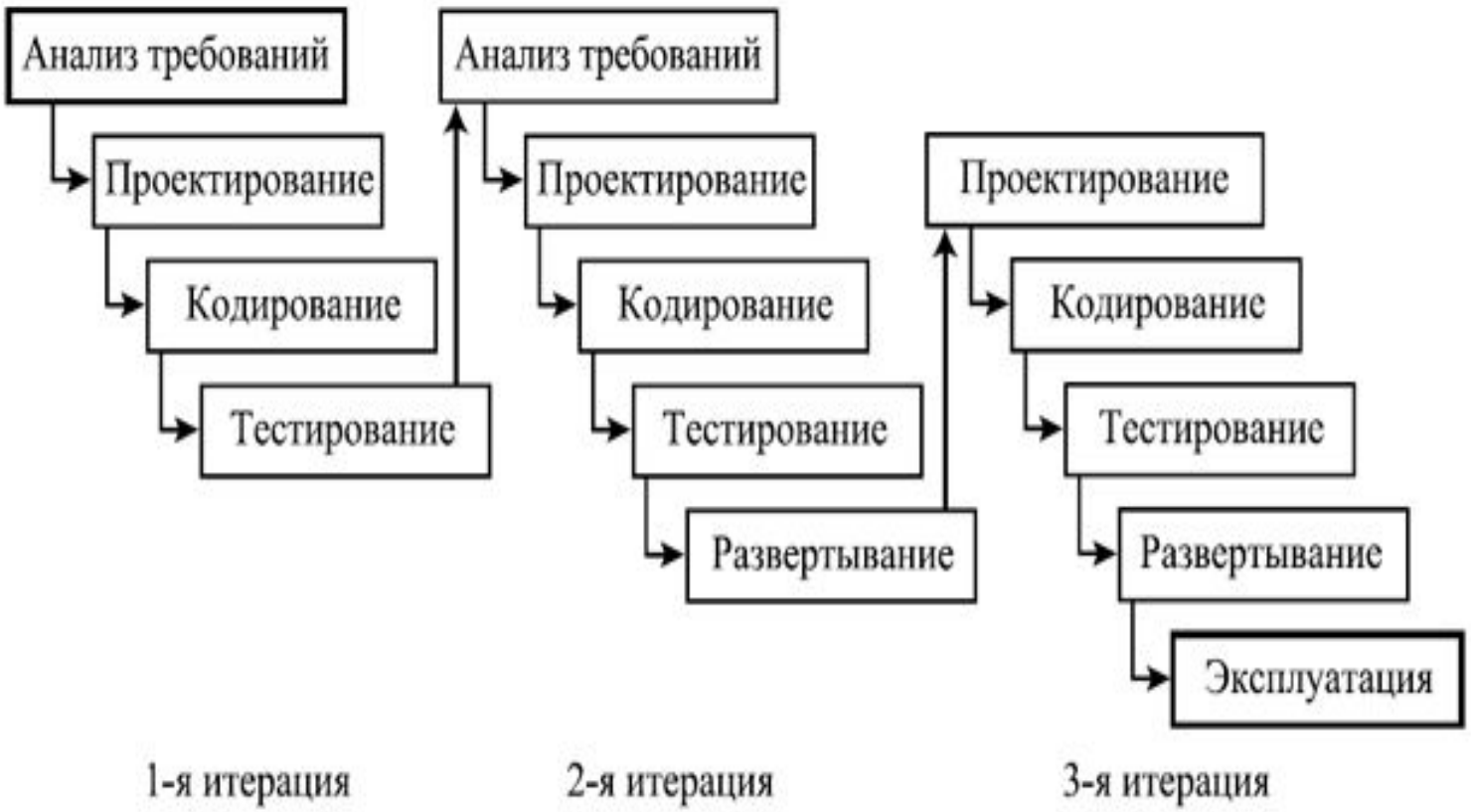
Цикл: планирование — действие — проверка — корректировка

Инкрементная модель

Инкрементная разработка представляет собой процесс **частичной** реализации всей системы и медленного наращивания функциональных возможностей.

Каждая часть представляет собой готовый фрагмент итогового продукта

Инкрементная модель



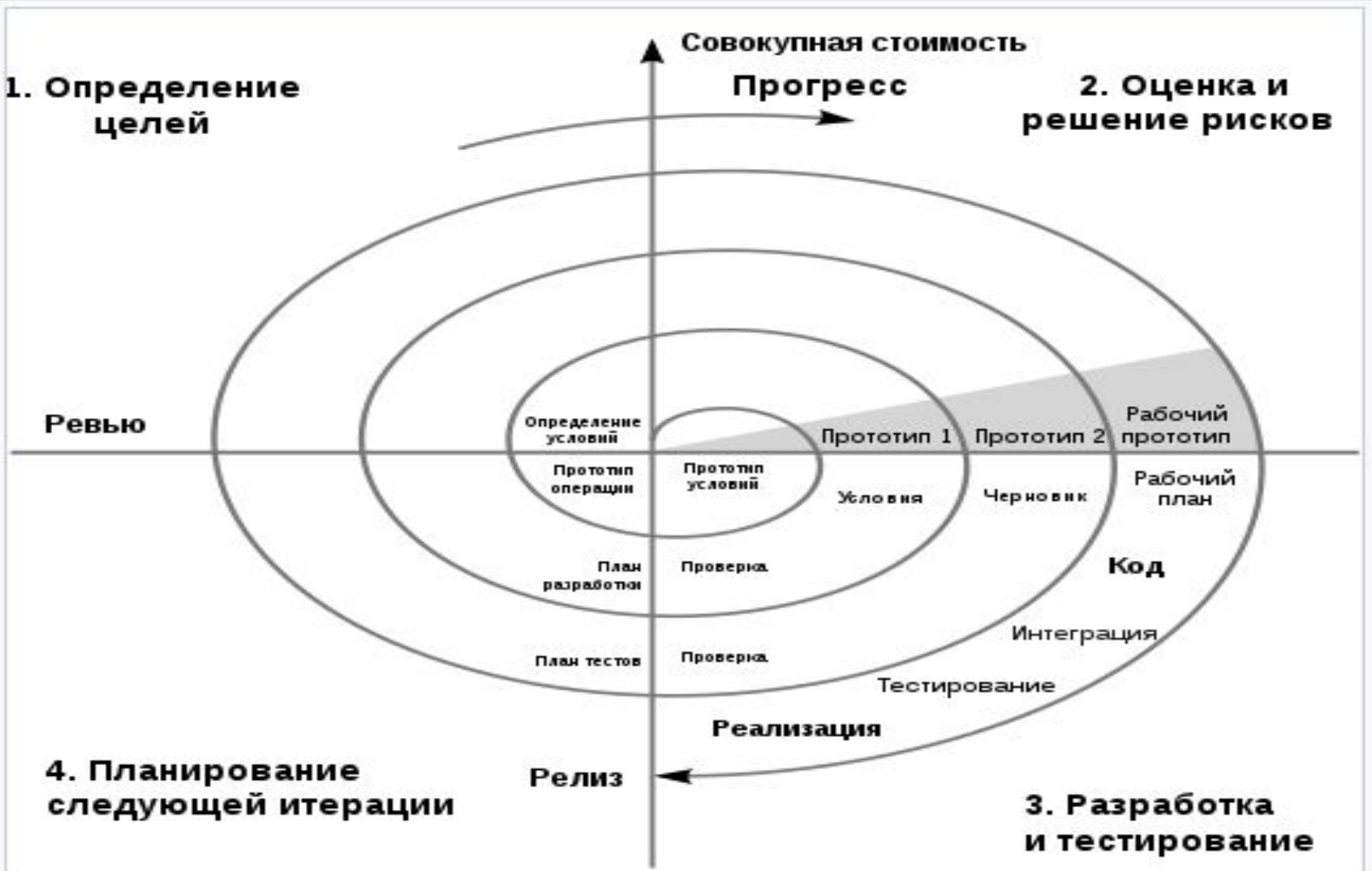
Отличия итеративной и инкрементальной модели

Спиральная модель

Спиральная модель (Spiral Model) представляет собой разновидность итерационно-инкрементальной модели, где на каждом витке спирали получается минимально готовая версия планируемого на данном витке продукта.

- Проработка целей
- Анализ рисков и прототипирование
- Разработка промежуточной версии продукта
- Планирование следующего витка (витка)

Спиральная модель



Методологии разработки ПО

- Под методологией разработки подразумевается набор методов и критериев оценки, которые используются для постановки задачи, планирования, контроля и в конечном итоге — для достижения поставленной цели.
- Сам процесс разработки описывается моделью, которая определяет последовательность наиболее общих этапов и получаемых результатов.

Гибкая методология разработки (Agile)

Гибкая модель (Agile model) - совокупность подходов к разработке ПО

Работа основывается на Agile манифесте (ценности):

- **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

Гибкая методология разработки (Agile)

- Практика организации труда небольших групп
- Минимизация рисков путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели.
- Каждая итерация включает все задачи, необходимые для релиза ПО (планирование, анализ, требований, проектирование, программирование, тестир ование и документирование).
- По окончании каждой итерации команда выполняет переоценку приоритетов разработки.
- Упор на непосредственном общении лицом к лицу
- Основной метрикой agile-методов является рабочий продукт.

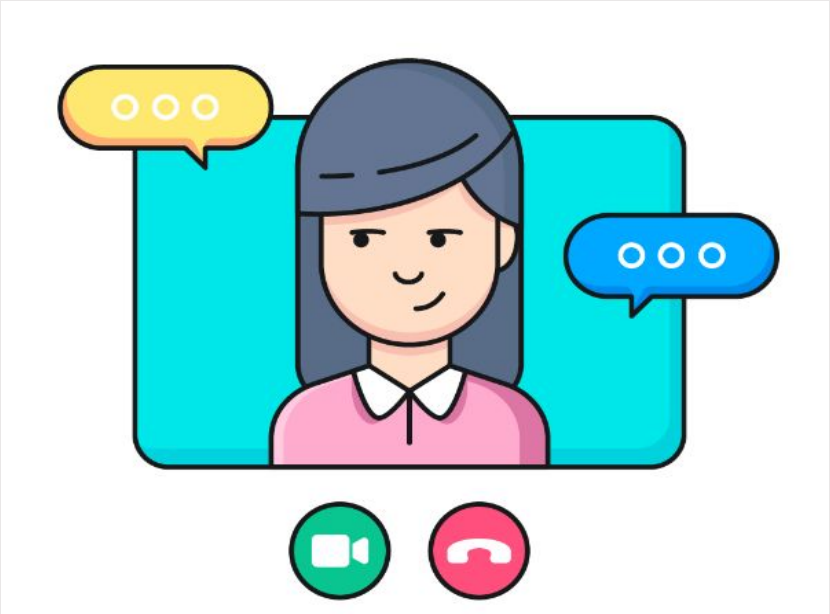
SCRUM

Скрам (Scrum) — метод (фреймворк) управления проектами, который помогает решать изменяющиеся в процессе работы задачи, чтобы поставлять клиентам продукты с максимально возможной ценностью.

- Вся работа разбивается на итерации, которые называются Спринт (***Sprint***). Каждый Спринт длится от 2х до 4х недель.

SCRUM

- Product Owner
- Scrum Master
- Команда



SCRUM

- *Product Backlog*
- *Sprint Backlog*
- *Daily Scrum*
- *Demo (или Review)*
- *Ретроспектива*

SCRUM


Ежедневный скрам
8 часов
Беклог продукта
Беклог спринта
Готовый продукт с новой функциональностью
2-4 недели
Ретроспектива
Планирование спринта
Разработка
Мониторинг и управление
Тестирование и демонстрация владельцу продукта

DevOps

DevOps методология активного взаимодействия специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимная интеграция их рабочих процессов друг в друга для обеспечения качества продукта.

Предназначена для эффективной организации создания и обновления программных продуктов и услуг.

Основана на идее тесной взаимозависимости создания продукта и эксплуатации программного обеспечения, которая прививается команде как культура создания продукта.

DevOps

Цели DevOps охватывают весь процесс поставки программного обеспечения.

- Сокращение времени для выхода на рынок;
- Снижение частоты отказов новых релизов;
- Сокращение времени выполнения исправлений;
- Уменьшение количества времени на восстановления (в случае сбоя новой версии или иного отключения текущей системы).

Жизненный цикл тестирования

- Планирование и анализ требований
- Уточнение критериев приемки
- Уточнение стратегии тестирования
- Разработка тестовой документации
- Выполнение тест-кейсов и Фиксация найденных дефектов
- Анализ результатов тестирования и Отчетность

Жизненный цикл тестирования

Принципы тестирования

1) Тестирование демонстрирует наличие дефектов, а не их отсутствие
2) Исчерпывающее тестирование недостижимо
3) Раннее тестирование сохраняет время и деньги
4) Кластеризация дефектов
5) Парадокс пестицида
6) Тестирование зависит от контекста
7) Заблуждение об отсутствии ошибок

Команда тестирования

1. Head of QA department.
2. QA manager
3. QA specialist
   - junior
   - middle
   - senior
   - team lead

Модуль 1. Итоги

- Зачем нужно тестировать программы?
- Понятие качество ПО. Стандарты качества ПО.
- Атрибуты и характеристики качества ПО.
- Основные определения тестирования.
- Жизненный цикл ПО.
- Методологии разработки ПО.
- Жизненный цикл тестирования ПО.
- Принципы тестирования.
- Команда тестирования

Модуль 2

Методы и виды тестирования. Анализ требований к ПО

Содержание

- Уровни и методы тестирования.
- Классификация видов тестирования.
- Подходы в тестировании.
- Критерии тестового покрытия.
- Требования к ПО.
- Классификация требований.
- Анализ требований к программному обеспечению.
- Как задавать вопросы ?!?

Уровни тестирования

Модульное тестирование - направлено на проверку отдельных небольших частей приложения

Интеграционное тестирование - направлено на проверку взаимодействия между несколькими частями приложения

Уровни тестирования

Системное тестирование - направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях.

Приемочное тестирование - формализованное тестирование, направленное на проверку приложения с точки зрения конечного пользователя/заказчика и вынесения решения о том, принимает ли заказчик работу у исполнителя

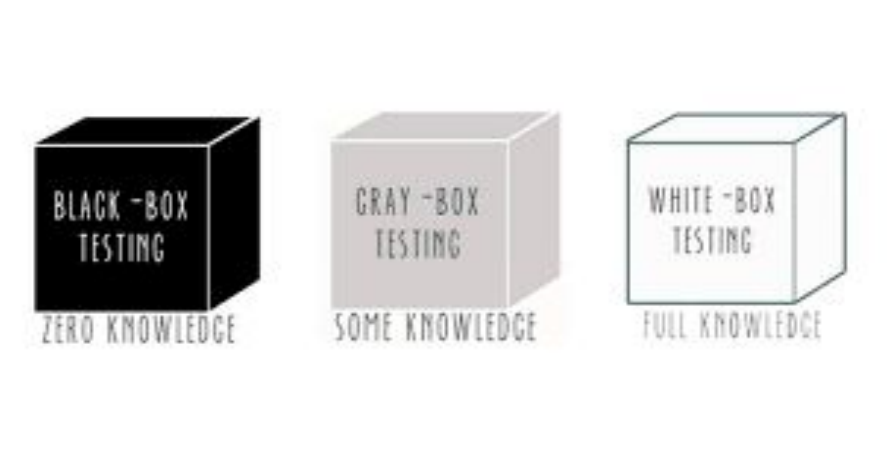
Методы тестирования

Метод белого ящика (white box testing) - есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

Метод чёрного ящика (black box testing) - либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования

Методы тестирования

Метод серого ящика (gray box testing) — комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет.



Классификация видов тестирования

- Статическое и динамическое
- По степени автоматизации
- По целям: функциональное, нефункциональное
- По фокусировке на уровне архитектуры
- По важности тестируемых функций
- Связанные с изменениями
- По времени проведения

По запуску кода на исполнение

Статическое тестирование (static testing)
тестирование без запуска кода на исполнение.

Динамическое тестирование (dynamic testing)
Тестирование, проводимое во время выполнения программного обеспечения, компонента или системы.

Статическое и динамическое тестирование. Отличия

- **Статическое**

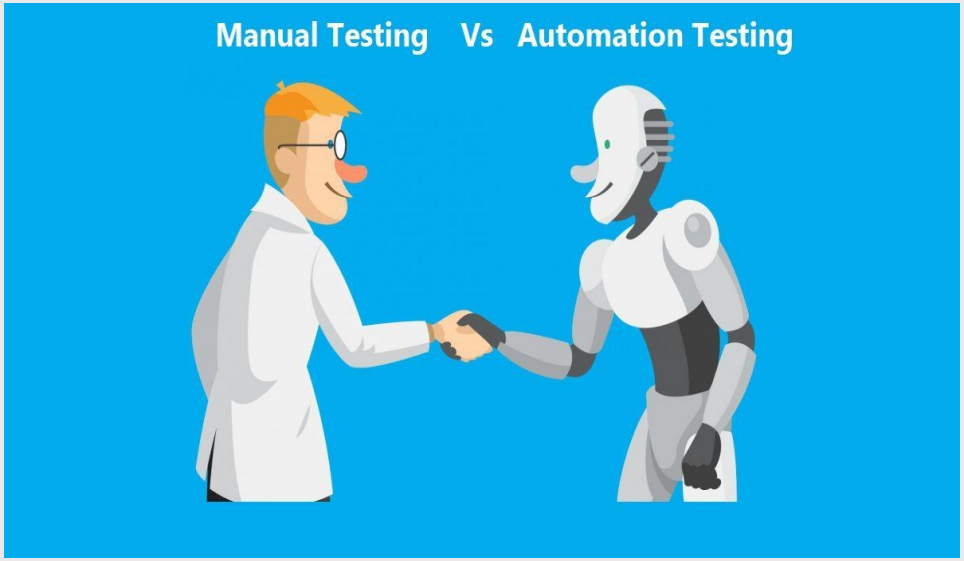
1. Без запуска кода
2. Верификация
3. Предотвращение дефектов
4. Оценка кода и документации
5. Чек-листы
6. Цена дефекта меньше

- **Динамическое**

1. Запуск кода
2. Валидация
3. Поиск и исправление дефектов
4. Ошибки и узкие места в ПО
5. Тест-кейсы
6. Цена дефекта выше

По степени автоматизации

- Ручное тестирование
- Автоматизированное тестирование



Функциональное тестирование

ISO 9126 Модель качества

1. *Функциональное тестирование (Functional testing)*
2. *Тестирование безопасности (Security and Access Control Testing)*
3. *Тестирование взаимодействия (Interoperability Testing)*

Тестирование взаимодействия (Interoperability Testing) – это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости (compatibility testing) и интеграционное тестирование

Нефункциональное тестирование

Нефункциональное тестирование - вид тестирования, направленный на проверку нефункциональных особенностей приложения

- Уровень производительности
- Совместимость
- Удобство пользования
- Надежность
- Защищенность
- Сопровождаемость
- Переносимость

Нефункциональное тестирование

Тестирование производительности - Тип тестирования, проводимого для оценки степени, в которой элемент тестирования выполняет свои определенные функции при заданных ограничениях времени и других ресурсах.

Исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке.

Пример: нагрузочное тестирование, стресс-тестирование

Нефункциональное тестирование

Нагрузочное тестирование — обычно проводится для того, чтобы оценить поведение приложения под заданной ожидаемой нагрузкой.

Пример: ожидаемое количество одновременно работающих пользователей приложения, совершающих заданное число транзакций за интервал времени.

Нефункциональное тестирование

Стресс-тестирование позволяет проверить, насколько приложение и система в целом работоспособны в условиях стресса, а также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию, после прекращения воздействия стресса.

Пример: повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера.

Также, одной из задач при стрессовом тестировании может быть оценка деградации производительности

Нефункциональное тестирование

Тестирование совместимости - тестирование, направленное на проверку способности приложения работать в указанном окружении.

- Совместимость с аппаратной платформой, операционной системой и сетевой инфраструктурой (конфигурационное тестирование)
- Совместимость с браузерами и их версиями (кросс-браузерное тестирование
- Совместимость с мобильными устройствами

Нефункциональное тестирование

Тестирование надежности - тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени или заданного количества операций.

Основная задача — наблюдая за потреблением ресурсов, выявить утечки памяти и проследить чтобы скорость обработки данных и/или время отклика приложения в начале теста и с течением времени не уменьшалась.

В противном случае вероятны сбои в работе продукта и перезагрузки системы.

Нефункциональное тестирование

Тестирование безопасности - тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.

1. Конфиденциальность
2. Целостность
3. Неподдельность
4. Отслеживаемость
5. Подлинность

Нефункциональное тестирование

Открытый проект по обеспечению безопасности веб-приложений (OWASP) представляет собой некоммерческий, образовательный, благотворительный фонд, помогающий организациям начать проектировать, разрабатывать, приобретать, использовать и поддерживать безопасное ПО.

Все инструменты, документы, форумы и отделения OWASP являются бесплатными и открытыми для тех, кто заинтересован в улучшении безопасности приложений.

Информацию можно найти по адресу www.owasp.org

Нефункциональное тестирование

Тестирование сопровождаемости - Тип тестирования, проводимого для оценки степени эффективности и продуктивности возможных изменений элемента тестирования.

Сопровождаемость: результативность и эффективность, с которыми продукт или система могут быть модифицированы предполагаемыми специалистами по обслуживанию

Нефункциональное тестирование

Тестирование удобства использования - тестирование, направленное на исследование того, насколько конечному пользователю понятно, как работать с продуктом, а также на то, насколько ему нравится использовать продукт

1. Определяемость пригодности
2. Изучаемость
3. Управляемость
4. Защищенность от ошибки
5. Эстетика пользовательского интерфейса
6. Доступность

Нефункциональное тестирование

Тестирование переносимости - Тип тестирования, проводимого для оценки простоты переноса элемента тестирования из одних аппаратных средств или программной среды в другие, включая уровень его изменений, необходимых для выполнения в средах различных типов.

Нефункциональное тестирование

Тестирование установки направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения (проверка установки приложений и программ, как десктопных, так и мобильных)

- Невозможность установить ПО
- Потеря данных после установки новой версии
- Невозможность откатиться до предыдущей версии

Нефункциональное тестирование

Тестирование на отказ и восстановление (**Failover and Recovery Testing**) проверяет с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками ПО, отказами оборудования или проблемами связи (например, отказ сети).

Цель: проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

Тестирование на отказ и восстановление очень важно для систем, работающих по принципу “24x7”.

По фокусировке на уровне архитектуры

Уровень представления - сконцентрировано на той части приложения, которая отвечает за взаимодействие с «внешним миром» (как пользователями, так и другими приложениями).
Уровень бизнес- логики - отвечает за проверку основного набора функций приложения и строится на базе ключевых требований к приложению, бизнес-правил и общей проверки функциональности.

Уровень данных сконцентрировано на той части приложения, которая отвечает за хранение и некоторую обработку данных

По степени важности тестируемых функций

Дымовое - направлено на проверку самой главной, самой важной, самой ключевой функциональности

Критического пути - направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности

Расширенное - направлено на исследование всей заявленной в требованиях функциональности — даже той, которая низко проранжирована по степени важности.

Связанные с изменениями

- Тестирование сборки (Build Verification Test)
- Дымовое тестирование (Smoke Testing)
- Повторное тестирование (Re-testing)
- Санитарное тестирование или проверка согласованности/исправности (Sanity Testing
- Регрессионное тестирование (Regression Testing)

Связанные с изменениями

Тестирование сборки (Build Verification Test)

Тестирование, направленное на определение соответствия выпущенной версии критериям качества для начала тестирования. По своим целям является аналогом дымового тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

Связанные с изменениями

Дымовое тестирование *Smoke testing* - минимальный набор тестов на явные ошибки. Пример, ошибки инсталляции

Часто такой набор тестов автоматизируется

Часто рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что, после сборки кода (нового или исправленного), устанавливаемое приложение стартует и выполняет основные функции.

Связанные с изменениями

Повторное тестирование
(Re-testing) Повторное выполнение тестов, для которых ранее был получен результат «сбоя», для оценки эффективности произведенных корректирующих действий. Примечание — Используется также термин «тестирование подтверждения»

- Повторное тестирование часто выполняется простым повтором теста, который привел к неожидаемому результату и который в свою очередь привел к идентификации первичной проблемы.
- Однако для эффективного повторного тестирования могут быть определены и проанализированы новые тестовые условия, а также разработаны новые тесты.

Связанные с изменениями

Санитарное тестирование - это узконаправленное тестирование, достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям. Является подмножеством регрессионного тестирования.

Используется для определения работоспособности определенной части приложения

В отличии от дымового (Smoke testing), санитарное тестирование (Sanity testing) направлено вглубь проверяемой функции, в то время как дымовое - направлено вширь, для покрытия тестами как можно большего функционала в кратчайшие сроки.

Связанные с изменениями

Регрессионное тестирование - это вид тестирования, направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб-сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

Регрессионными могут быть как функциональные, так и нефункциональные тесты.

- Re-testing — проверяется исправление багов
- Regression testing — проверяется то, что исправление багов, а также любые изменения в коде приложения, не повлияли на другие модули ПО и не вызвало новых багов.

По времени проведения

Альфа - Моделируемое или действительное эксплуатационное тестирование потенциальными пользователями/заказчиками или независимой командой тестирования на стороне разработчиков, но вне разрабатывающей организации. Альфа-тестирование часто применяется к коробочному программному обеспечению в качестве внутреннего приёмочного тестирования.

Бета - интенсивное использование почти готовой версии продукта с целью выявления максимального числа ошибок в его работе для их последующего устранения перед окончательным выходом продукта на рынок, к массовому потребителю. Выполняется вне организации-разработчика с активным привлечением конечных пользователей заказчиков

Подходы в тестировании

На основе тест- кейсов (script-based) формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов

Исследовательское (exploratory testing) частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию

Свободное (ad hoc testing) полностью неформализованный подход. Тестировщик полностью опирается на свой профессионализм и интуицию

Подходы в тестировании

Позитивное тестирование - направлено на исследование приложения в ситуации, когда все действия выполняются строго по инструкции без каких бы то ни было ошибок, отклонений, ввода неверных данных и т.д.

Негативное тестирование - направлено на исследование работы приложения в ситуациях, когда с ним выполняются (некорректные) операции и/или используются данные, потенциально приводящие к ошибкам

Тестовое покрытие

Тестовое покрытие (test coverage): Степень, выраженная в процентах, в которой специфицированные элементы тестового покрытия были проверены тест-кейсами

Критерии тестового покрытия

Критерий тестового покрытия – метрика для оценки качества тестирования

- Критерий покрытия измеряет долю классов, ситуаций, представители которых попали в тестовый набор.
- Чем больше уровень тестового покрытия, тем больше классов, ситуаций покрыто, тем больше ошибок можно обнаружить.

Покрытие требований

Покрытие требований (Requirements Coverage) - оценка покрытия тестами функциональных и нефункциональных требований к продукту

- Требование считается «покрытым», если на него ссылается хотя бы один тест-кейс
- Покрытие требований = количество требований, покрытых хотя бы одним тест-кейсом/ общее количество требований.
- Полностью покрыты: входные данные, комбинации входных данных, последовательности комбинаций входных данных

Покрытие требований

Traceability matrix — Матрица соответствия требований — это двумерная таблица, содержащая соответствие функциональных требований (functional requirements) продукта и подготовленных тестовых сценариев (test cases). В заголовках колонок таблицы расположены требования, а в заголовках строк — тестовые сценарии.

Тест-кейсы	Требование 1	Требование 2
1.1		+
1.2	+	

Покрытие кода

Покрытие кода (Code Coverage) - оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.

- Чаще всего используется при тестировании методом белого ящика
- Полностью покрыты: строки кода программы, ветви в коде программы, пути в коде программы

Требования к ПО

Условия или возможности, необходимые пользователю для решения определенных задач или достижения определенных целей , которые должны быть достигнуты для выполнения контракта, стандартов, спецификации или других формальных документов (IEEE 610)

На основе требований ведётся проектирование, разработка и тестирование ПО

Требования к ПО

1. условия или возможности, необходимые пользователю для решения проблем или достижения целей;
2. условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам;
3. документированное представление условий или возможностей для пунктов 1 и 2.

Уровни требований

1) Бизнес- требования
2) Пользовательские требования
3) Функциональные требования

Бизнес-требования

Бизнес-требования (business requirements)

Выражают цель ради которой создавался продукт

Содержат высокоуровневые цели организации или заказчиков системы.

Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга.

Пример: Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену

Пользовательские требования

Требования пользователей описывают задачи, которые пользователь может выполнять с помощью разрабатываемого продукта, а также способы (сценарии) их решения в системе (реакция продукта на действия пользователя, сценарии работы пользователя).

Пользовательские требования представлены в виде:

- вариантов использования (uses cases)
- пользовательских историй (user stories)
- пользовательских сценариев (user scenarios)

Пользовательские требования

Структура user stories

Как, <роль/персонаж юзера>, я <что-то хочу получить>, <с такой-то целью>

Пример user stories

Как потребителю мне удобно искать книги по жанрам, чтобы быстро найти те, которые я люблю читать.

Функциональные требования

Функциональные требования (functional requirements) определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований.

Они содержат положения с традиционным «должен» или «должна»: «Система должна по электронной почте отправлять пользователю подтверждение о заказе». Или не должна..

Пример: В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе

Виды требований

- Функциональные
- Нефункциональные
- Ограничения проектирования

Нефункциональные требования

Нефункциональные требования (Non-functional Requirements) - охватывают свойства системы (удобства использования, надежность, масштабируемость), которыми она должна обладать при реализации своего поведения.

Нефункциональные требования в основном влияют на архитектуру продукта.

Пример:

- Время ответа для транзакции: среднее, максимальное
- Емкость: сколько пользователей или транзакций может обслужить система

Ограничения проектирования

Ограничения проектирования налагаются на проект системы или процессы, с помощью которых система создается.

Они не влияют на внешнее поведение системы, но должны выполняться для удовлетворения технических, деловых или контрактных обязательств.

Ограничения проектирования

Источники ограничений проектирования

- Операционные среды: Программы пишутся на Visual Basic.
- Совместимость с существующими системами: Приложение должно выполняться как на новой, так и на прежней платформе.
- Прикладные стандарты: Использовать библиотеку классов из Developer's Library на корпоративном сервере
- Корпоративные практические наработки и стандарты: Использовать стандарты программирования C++
- Инструкции и стандарты, которым подчиняется разработка проекта.

Источники требований

1) Законодательство
2) Нормативное обеспечение организации
3) Представления и ожидания пользователей
4) Конкурирующие программные продукты

Метод выявления требований

1) Мозговой штурм
2) Интервью, опросы, анкетирование
3) Анализ документации
4) Анализ конкурентных продуктов
5) Анализ статистики использования предыдущих версий

Свойства требований

- полнота,
- ясность,
- корректность,
- согласованность,
- *верифицируемость*,
- необходимость,

Свойства требований

- полезность при эксплуатации,
- осуществимость,
- модифицируемость,
- трассируемость,
- упорядоченность по важности и стабильности,
- наличие количественной метрики.

Цель тестирования требований

Проверка удовлетворения требований заданным критериям качества для раннего обнаружения и исправления ошибки при проектировании требований.

Ошибки в требованиях значительно увеличивают затраты на их исправление во время разработки продукта. (исправление самой ошибки, переработка артефактов: архитектура проекта, внедрение, тест-кейсы).

Анализ требований с точки зрения пригодности к тестированию

Тестопригодные требования – степень выраженности требований в терминах, допускающих начало работы над разработкой тестов (и в последствии над тестовыми сценариями) и выполнение тестов для определения соответствия заявленным требованиям [IEEE 610]

Анализ требований с точки зрения пригодности к тестированию

Система должна постоянно функционировать с максимальной мощностью, за исключением особых ситуаций, в которых она должна обеспечивать до 125% мощности при условии, что особая ситуация длится не более 15 минут, в противном случае мощность должна быть снижена до 105%, но если можно обеспечить только 95%, то система должна активировать исключительный режим работы при сниженной мощности и сохранять уровень мощности в пределах 10% отклонений от заданных значений в течение как минимум 30 минут.

Анализ требований с точки зрения пригодности к тестированию

Система должна предоставлять возможности обработки текстов, должна быть удобной для неподготовленного персонала и работать в среде локальной сети Ethernet, реализованной аппаратными средствами в виде системы воздушных кабельных каналов с интегрированными сетевыми адаптерами, устанавливаемыми в каждую систему, с добавлением дополнительных модулей памяти при необходимости.

Анализ требований с точки зрения пригодности к тестированию

Время отклика системы с точки зрения конечного пользователя (end-to-end) во время продуктивной нагрузки (50 пользовательских сессий в режиме «менеджер» / 15 пользовательских сессий в режиме «аналитик») при загруженности пропускного канала от клиентской системы до сервера приложений в пределах 50% для сети 100 Mb/sec и утилизации ресурсов сервера приложений (CPU, RAM) в рамках 70-80%, а клиентской машины в рамках 40-60%, не должно превышать 1 секунды для операций создания записи (сущности) и 3 секунд для операций поиска.

Анализ требований с точки зрения пригодности к тестированию

1. Задавать вопросы
2. Создавать чек-листы
3. Рисовать
4. Взаимный пересмотр
   - Неформальный
   - Технический
   - Формальная инспекция

Три взгляда на требования

1. с точки зрения данных;
2. с точки зрения поведения; (нормальное, альтернативное, исключения)
3. с точки зрения функциональности;

Аспекты качества требований

1. **Содержание** (полнота, трассируемость, корректность и адекватность, согласованность, проверяемость, необходимость)
2. **Документация** (понятность, однозначность, соответствие правилам документирования)
3. **Соглашение** (согласовано, согласовано после внесения изменений, конфликты разрешены)

Как задавать вопросы

Вы задаете вопросы, чтобы помочь команде создать лучший продукт

- Открытые вопросы
- Закрытые вопросы
- Подготовка к вопросу
- Метод “пять почему”
- Метод “Кто, что, где, когда, почему и как”
- Конструкции вопросов

Как задавать вопросы

- Открытые вопросы (Как вы думаете почему..? Можете описать как..?)
- Закрытые вопросы фокусируются на фактах и предполагают ответы Да/Нет (Когда дедлайн?)
- Контекст вопроса (почему вы задаете вопрос и какая информация вам нужна, для чего вам это)
- Метод "Пять почему" помогает обнаружить скрытые мотивы и убеждения людей

Как задавать вопросы

- Метод "Кто, что, где, когда, почему, как" помогает понять контекст проблемы
- Конструкции вопросов (Что, если..? Каким образом это изменится, если..? Предположим что...? Каким другим способом мы могли бы..?)

Модуль 2. Итоги

- Уровни и методы тестирования
- Классификация видов тестирования
- Подходы в тестировании
- Критерии тестового покрытия
- Требования к ПО
- Классификация требований
- Анализ требований к программному обеспечению
- Как задавать вопросы

Итоги работы

Модуль 1. Введение в тестирование ПО

Модуль 2. Методы и виды тестирования. Анализ требований к ПО

Выбирайте Центр «Специалист» – крупнейший учебный центр России!

info@specialist.ru
+7 (495)
232-32-16