Оглавление

Жизненный цикл тестирования

• жизненный цикл тестирования — это **последовательность действий, проводимых в процессе тестирования, с помощью которых гарантируется качество программного обеспечения и его соответствие требованиям.**

Цикл тестирования:

1. Общее планирование и анализ требований
2. Уточнение критериев приёмки
3. Уточнение стратегии тестирования
4. Разработка тест-кейсов
5. Выполнение тест-кейсов
6. Фиксация найденных дефектов
7. Анализ результатов тестирования
8. Отчётность

Виды тестирования:

1. Функциональные
2. Нефункциональные
3. Связанные с изменениями

Функциональные виды тестирования

Функциональные тесты.

1. Функциональные
2. Нефункциональные
3. Связанные с имзенениями

Тестирование безопасности.

1. Конфиденциальность
2. Целостность
3. Доступность

Нефункциональные виды тестирования

Тестирование производительности

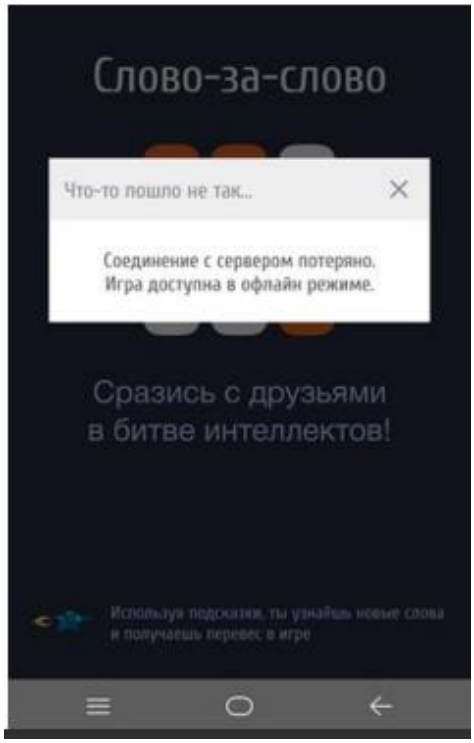
- Измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций.
- Определение количества пользователей, одновременно работающих с приложением.
- Определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций).
- Исследование производительности на высоких, предельных, стрессовых нагрузках.

Стрессовое тестирование

Объемное тестирование

- Измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций.
- Может производиться определение количества пользователей, одновременно работающих с приложением.

Тестирование стабильности или надежности



Тестирование Установки

- Процесс получения списка файлов проводится до, а проверка файлов — после установки программы;
- Проверка корректности регистрации библиотек и служебных записей;
- Корректность регистрации расширений для новых файлов инсталлированной веб-программы внутри операционной системы;

- Проверять учетную запись пользователя и моментально информировать о потенциально возможных проблемах с пользовательскими правами;
- Проверка на возможные конфликты в отношении системных доступов к общим файловым ресурсам при одновременной установке сразу нескольких программ;
- Тестирование на обратную совместимость – при выполнении обновлений, все ранее инсталлированные файлы и папки должны правильно открываться и выполнять свои непосредственные системные «обязанности»;
- Если продукт запущен, клиент должен получить специальное уведомление о том, что выполнить обновление продукта не получится, поскольку программа уже функционирует;
- Процедура удаления продукта в процессе выполнения инсталляции (удалять можно как библиотеки в системном каталоге, так и регистрационные записи внутри системного реестра);
- Тестирование проверки удаления реестров, ссылки на системные библиотеки;
- Тест на сохранность данных при взаимодействии с продуктом.
- Проверка поведения инсталлятора, при условии, что каталог продукта закрыт для процесса удаления по правам доступа;
- Если пользователь не авторизирован для деинсталляции продукта, он должен получить специальное уведомление и процесс удаления должен быть остановлен.

Тестирование удобства пользования

- **Производительность, эффективность (efficiency)** - сколько времени и шагов понадобится пользователю для завершения основных задач приложения.
- **Правильность (accuracy)** - сколько ошибок сделал пользователь во время работы с приложением .
- **Активизация в памяти (recall)** – как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени.
- **Эмоциональная реакция (emotional response)** – как пользователь себя чувствует после завершения задачи - растерян, испытал стресс? Порекомендует ли пользователь систему своим друзьям?

- Готовый продукт - тестирование черного ящика (black box testing),
- Интерфейсы приложения (API) - тестирование белого ящика (white box testing).

- «пока-йока» или fail-safe, «защита от дурака»
- цикл Демминга (Plan-Do-Check-Act) :

- Тестирование пользовательского интерфейса = Тестирование удобства пользования
- Тестирование удобства пользования можно провести без участия эксперта

Тестирование на отказ и восстановление

- Целью данного вида тестирования является проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

- Отказ электричества на компьютере-сервере.
- Отказ электричества на компьютере-клиенте.
- Незавершенные циклы обработки данных (прерывание работы фильтров данных, прерывание синхронизации).
- Объявление или внесение в массивы данных невозможных или ошибочных элементов.
- Отказ носителей данных.

- Симулировать внезапный отказ электричества на компьютере (обесточить компьютер).
- Симулировать потерю связи с сетью (выключить сетевой кабель, обесточить сетевое устройство).
- Симулировать отказ носителей (обесточить внешний носитель данных).
- Симулировать ситуацию наличия в системе неверных данных (специальный тестовый набор или база данных).

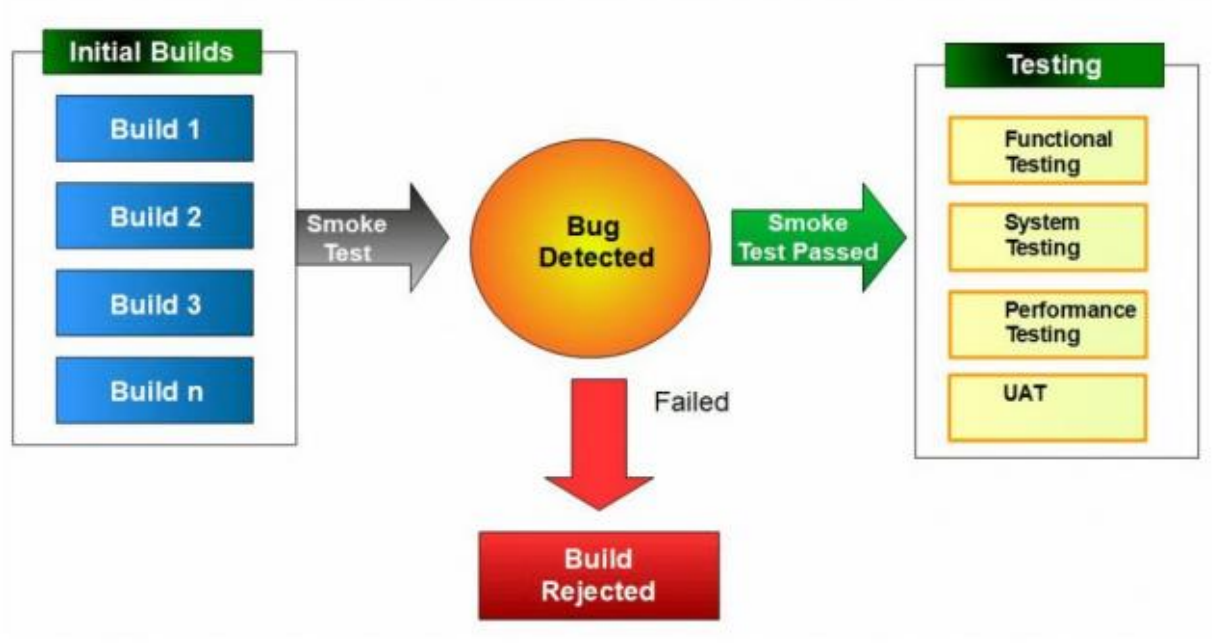
Конфигурационное тестирование

- Проект по профилированию работы системы. Цель Тестирования: определить оптимальную конфигурацию оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы.
- Проект по миграции системы с одной платформы на другую. Цель Тестирования: проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, операционными системами и программными продуктами третьих фирм.

Связанные с изменениями виды тестирования

Дымовое тестирование

Дымовое тестирование – это тестирование на наличие явных дефектов.



Регрессионное тестирование

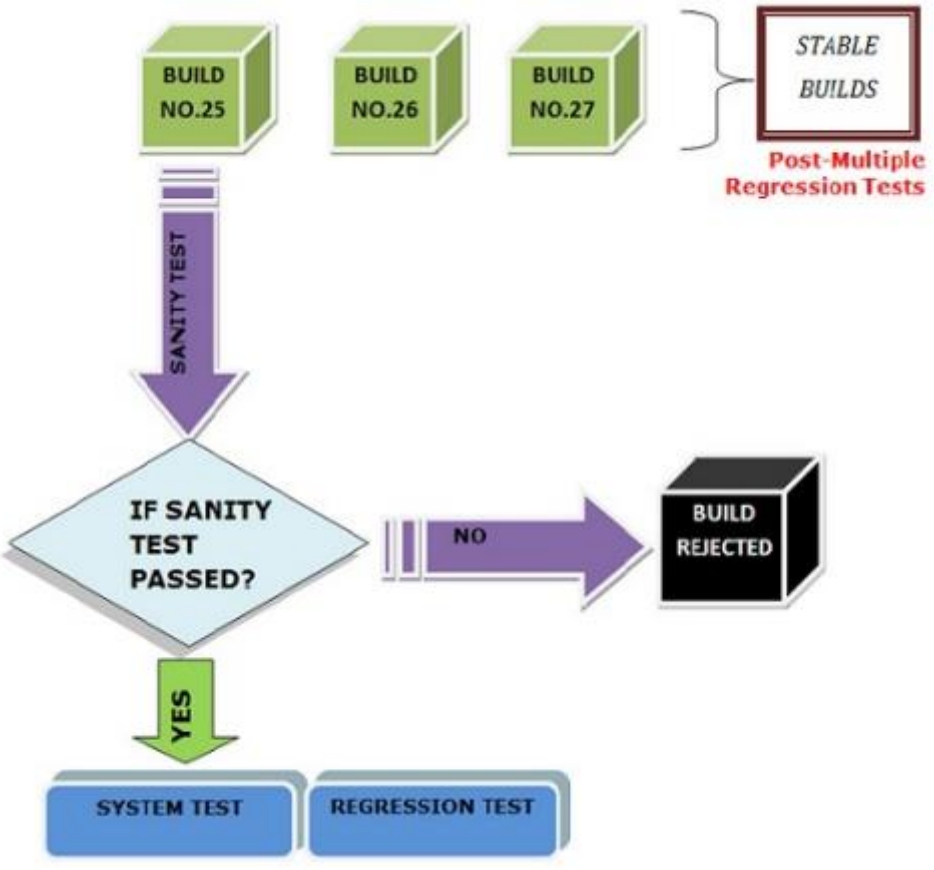
- Регрессия багов (Bug regression) - попытка доказать, что исправленная ошибка на самом деле не исправлена.

- Регрессия старых багов (Old bugs regression) - попытка доказать, что недавнее изменение кода или данных сломало исправление старых ошибок, т.е. старые баги стали снова воспроизводиться.
- Регрессия побочного эффекта (Side effect regression) - попытка доказать, что недавнее изменение кода или данных сломало другие части разрабатываемого приложения.

Тестирование сборки

- Тестовые примеры проверки сборки должны охватывать все важные функциональные возможности программного обеспечения.

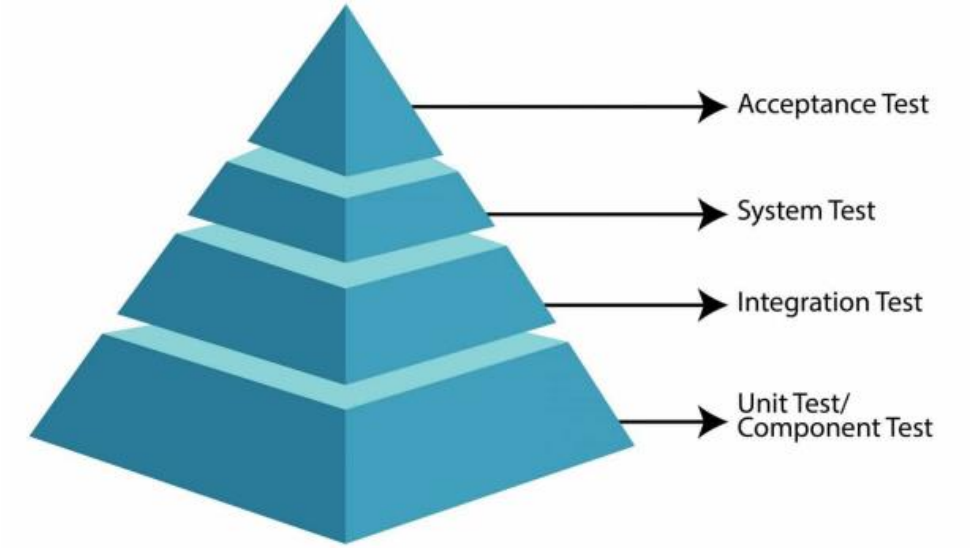
Санитарное тестирование



Санитарное тестрование – это тестирование конкретного стабильного билда.

Уровни Тестирования

Компонентное или Модульное тестирование



Разработка от тестирования (test-driven development) подход тестирования вначале (test first approach)

Интеграционное тестирование

- Компонентный интеграционный уровень (Component Integration testing) проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.
- Системный интеграционный уровень (System Integration Testing) - проверяется взаимодействие между разными системами после проведения системного тестирования.

- Снизу вверх (Bottom Up Integration): Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования.
- Сверху вниз (Top Down Integration): Сначала тестируются все высокоуровневые модули, затем

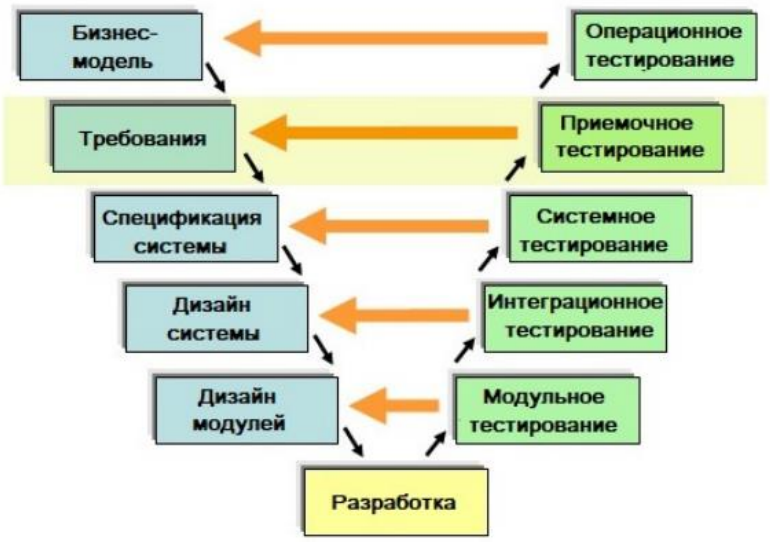
постепенно, один за другим, добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем, по мере готовности, они заменяются реальными активными компонентами.

- Большой взрыв ("Big Bang" Integration): Все (или практически все) разработанные модули собираются вместе в виде законченной системы или ее основной части, а затем проводится интеграционное тестирование.

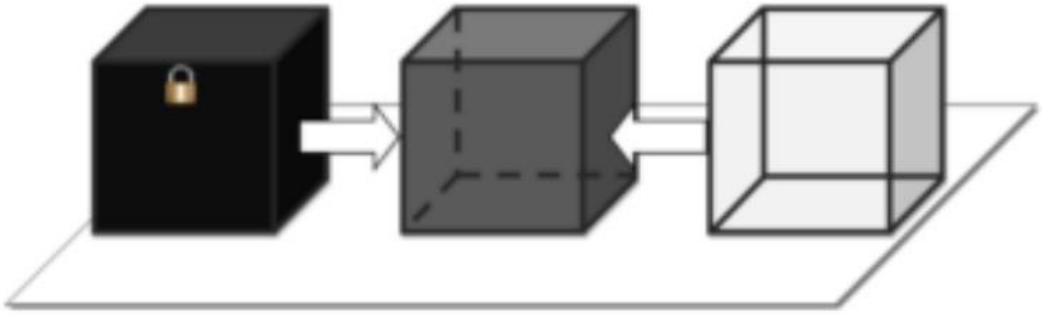
Системное тестирование

- на базе требований (requirements based) - для каждого требования пишутся тестовые случаи (test cases), проверяющие выполнение данного требования.
- на базе случаев использования (use case based) - на основе представления о способах использования продукта создаются случаи использования системы (Use Cases).

Приемочное тестирование. Операционное тестирование



Типы тестирования



Black Box

Преимущества

1. Тестирование производится с позиции конечного пользователя и может помочь обнаружить неточности и противоречия в спецификации;
2. Тестировщику нет необходимости знать языки программирования и углубляться в особенности реализации программы;
3. Тестирование может производиться специалистами, независимыми от отдела разработки, что помогает избежать предвзятого отношения;
4. Можно начинать писать тест-кейсы, как только готова спецификация.

Недостатки

1. Тестируется только очень ограниченное количество путей выполнения программы;
2. Без четкой спецификации (а это скорее реальность на многих проектах) достаточно трудно составить эффективные тест-кейсы;
3. Некоторые тесты могут оказаться избыточными, если они уже были проведены разработчиком на уровне модульного тестирования.

White Box

- Тестирование, основанное на анализе внутренней структуры компонента или системы;
- Тест-дизайн - составление тест-кейсов на основе анализа внутреннего устройства системы или компьютера.

Преимущества

1. Тестирование может производиться на ранних этапах: нет необходимости ждать создания пользовательского интерфейса;
2. Можно провести более тщательное тестирование с покрытием большого количества путей выполнения программы.

Недостатки

1. Для выполнения тестирования белого ящика необходимо большое количество специальных знаний;
2. При использовании автоматизации тестирования на этом уровне поддержка тестовых скриптов может оказаться достаточно накладной, если программа часто изменяется.

Сравнение Black Box и White Box

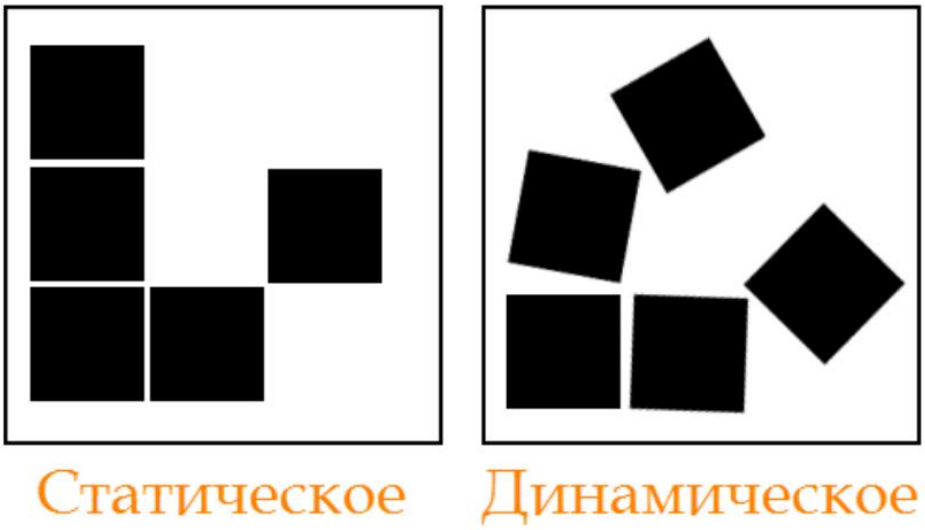
Критерий	Black Box	White Box
Определение	тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы	тестирование, основанное на анализе внутренней структуры компонента или системы
Уровни, к которым применима техника	В основном: • Приемочное тестирование • Системное тестирование	В основном: • Юнит-тестирование • Интеграционное тестирование
Кто выполняет	Как правило, тестировщики	Как правило, разработчики
Знание программирования	Не нужно	Необходимо
Знание реализации	Не нужно	Необходимо
Основа для тест-кейсов	Спецификация, требования	Проектная документация

Grey Box

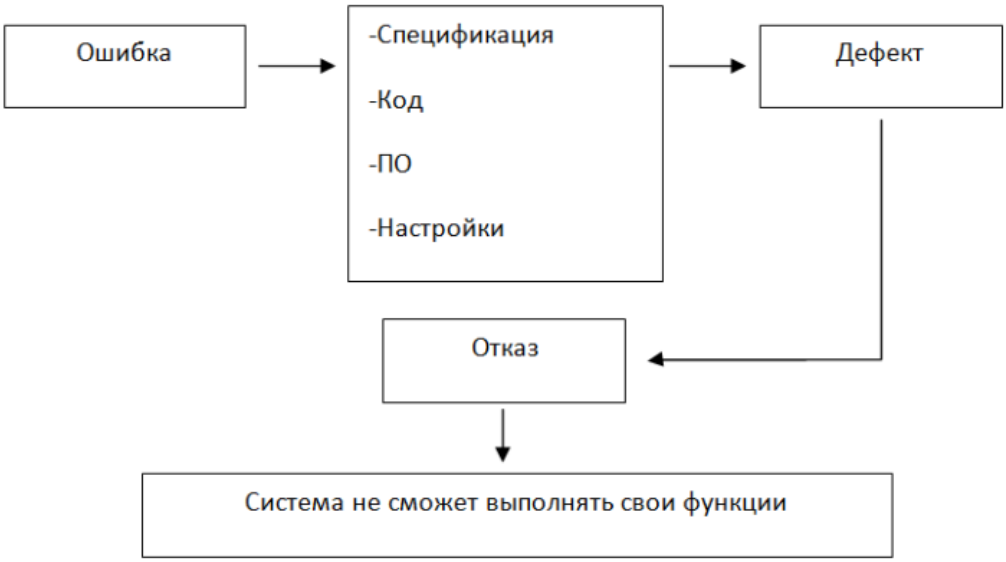
Статическое и динамическое тестирование

Статическое тестирование проводится без исполнения кода. Сюда относится корректура, проверка, ревизия кода (при наблюдении за работой другого / парном программировании), критический анализ, инспекции и так далее.

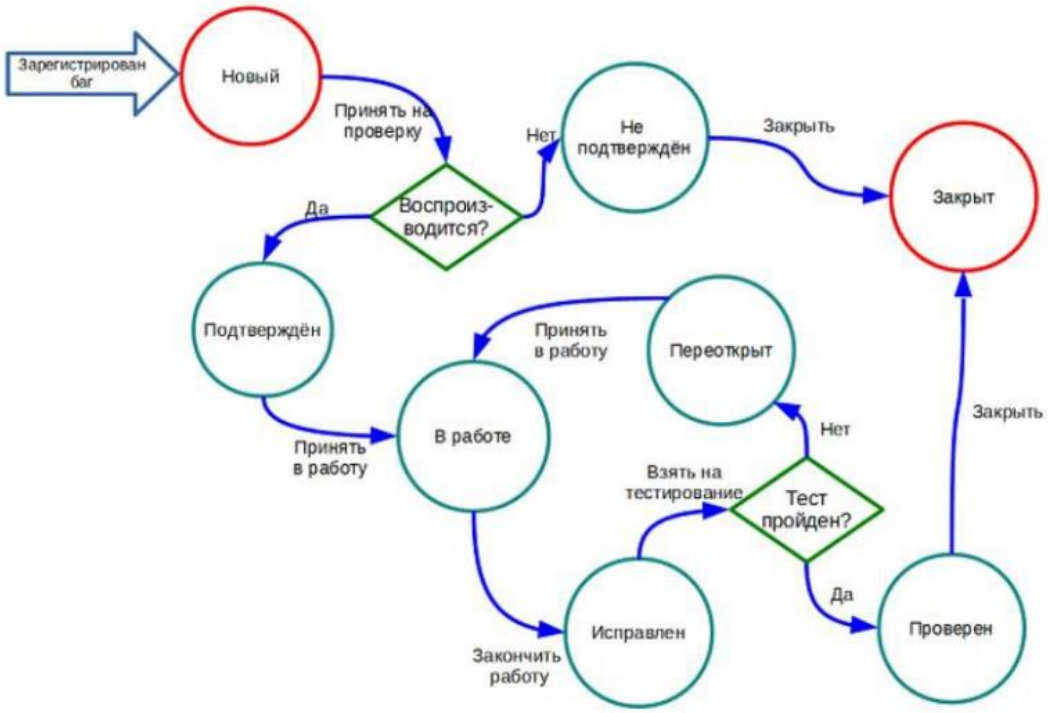
Динамическое тестирование для получения корректных результатов требует исполнять код. Например, для модульных тестов, интеграционных, системных, приёмочных и прочих тестов.



Отчёты о дефектах



Жизненный цикл «бага»



Использование данных отчета о дефекте

- Что было не так?
- Когда была создана эта проблема? Какое именно действие при разработке явилось ее источником? Это была проблема в требованиях? Проектировании системы? Коде? Тестировании?
- Когда проблема была выявлена? Сколько она существовала до того, как мы ее обнаружили?
- Каким образом была найдена эта проблема?
- Можно ли было обнаружить ее раньше?
- Сколько стоило устранение этой проблемы?
- Какого рода была эта проблема?

Атрибуты отчёта о дефекте

- **Идентификатор (identifier)** представляет собой уникальное значение, позволяющее однозначно отличить один отчёт о дефекте от другого и используемое во всевозможных ссылках.

- **Краткое описание (summary)** должно в предельно лаконичной форме давать исчерпывающий ответ на вопросы «Что произошло?», «Где это произошло?», «При каких условиях это произошло?»
- **Подробное описание (description)** представляет в развёрнутом виде необходимую информацию о дефекте, а также (обязательно!) описание фактического результата, ожидаемого результата и ссылку на требование (если это возможно).
- **Шаги по воспроизведению (steps to reproduce)** описывают действия, которые необходимо выполнить для воспроизведения дефекта.
- **Воспроизводимость (reproducibility)** показывает, при каждом ли прохождении по шагам воспроизведения дефекта удаётся вызвать его проявление. Это поле принимает всего два значения: всегда (always) или иногда (sometimes).
- **Важность (severity)** показывает степень ущерба, который наносится проекту существованием дефекта.
    - Критическая
    - Высокая
    - Средняя
    - Низкая
- **Срочность (priority)** показывает, как быстро дефект должен быть устранён.
    - Наивысшая
    - Высокая
    - Обычная
    - Низкая
- **Фактический результат (actual result)** - результат, полученный после прохождения шагов к воспроизведению.

- **Ожидаемый результат (expected result)** - описывает ожидаемое поведение ПО после прохождения шагов к воспроизведению.
- **Симптом (symptom)** — позволяет классифицировать дефекты по их типичному проявлению.
    - Косметический дефект
    - Повреждение/потеря данных
    - Проблема в документации
    - Некорректная операция
    - Проблема инсталляции
    - Ошибка локализации
    - Нереализованная функциональность
    - Проблема масштабируемости
    - Низкая производительность
    - Крах системы
    - Неожиданное поведение
    - Недружественное поведение
    - Расхождени с требованиями
    - Предложение по улучшению
- **Комментарий (comments, additional info)** — может содержать любые полезные для понимания и исправления дефекта данные.
- **Приложения (attachments)** — список прикреплённых к отчёту о дефекте приложений (копий экрана, вызывающих сбой файлов и т.д.)

Свойства качественных отчётов о дефектах

- Тщательное заполнение всех полей точной и корректной информацией.
- Правильный технический язык.
- Специфичность описания шагов.

- Отсутствие лишних действий и/или их длинных описаний.
- Отсутствие дубликатов.
- Очевидность и понятность.
- Прослеживаемость.
- Отдельные отчёты для каждого нового дефекта.
- Соответствие принятым шаблонам оформления и традициям.

Обеспечение качества и тестирование ПО

Обеспечение качества ПО

- **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.

Набор стандартов ISO 9126

Верификация и валидация

- **Верификация (verification)** - это процесс оценки системы или её компонентов с целью определения, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. Т.е. выполняются ли наши цели, сроки, задачи по разработке проекта, которые были определены в начале текущей фазы.
- **Валидация (validation)** - это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

№	Верификация	Валидация
1.	Делаем ли мы продукт *правильно?*	Делаем ли мы *правильный* продукт?
2.	Реализована ли *вся* функциональность?	*Правильно ли* реализована функциональность?
3.	Верификация происходит раньше и включает проверку правильности написания документации, кода и т.д.	Валидация происходит после верификации и, как правило, отвечает за оценку продукта в целом.
4.	Производится разработчиками	Производится тестировщиками
5.	Включает статический анализ – инспектирование кода, сравнение требований и т.п.	Включает динамический анализ – выполнение программы для сравнения ее реальной работы с установленными требованиями
6.	Основывается на объективной оценке соответствия реализованных функций	Субъективный процесс, включающий личную оценку качества работы ПО

Откуда берутся ошибки в ПО?

- **Ошибка (error)** – это действие человека, которое порождает неправильный результат.
- **Дефект, Баг (Defect, Bug)** – недостаток компонента или системы, который может привести к отказу определенной функциональности. Дефект, обнаруженный во время исполнения программы, может вызвать отказ отдельного компонента или всей системы.
- **Сбой (failure)** – несоответствие фактического результата (actual result) работы компонента или системы ожидаемому результату (expected result).

Причины появления дефектов в программном коде

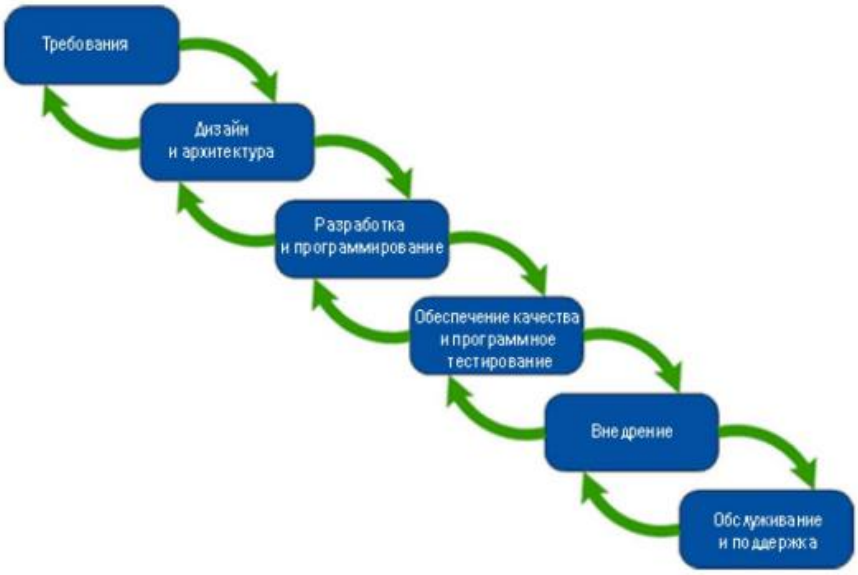
1. Недостаток или отсутствие общения в команде.
2. Сложность программного обеспечения.
3. Изменения требований.
4. Плохо документированный код.

5. Средства разработки ПО.

Принципы тестирования

1. Тестирование показывает наличие дефектов.
2. Исчерпывающее тестирование невозможно.
3. Раннее тестирование.
4. Скопление дефектов.
5. Парадокс пестицида (Если одни и те же тесты будут прогоняться много раз, в конечном счете этот набор тестовых сценариев больше не будет находить новых дефектов. Чтобы преодолеть этот “парадокс пестицида”, тестовые сценарии должны регулярно рецензироваться и корректироваться, новые тесты должны быть разносторонними, чтобы охватить все компоненты программного обеспечения, или системы, и найти как можно больше дефектов.).
6. Тестирование зависит от контекста.
7. Заблуждение об отсутствии ошибок.
    - Тестирование должно производиться независимыми специалистами;
    - Привлекайте лучших профессионалов;
    - Тестируйте как позитивные, так и негативные сценарии;
    - Не допускайте изменений в программе в процессе тестирования;
    - Указывайте ожидаемый результат выполнения тестов.

Жизненный цикл ПО



SO / IEC / IEEE 12207: 2017 Systems and Software development — software lifecycle processes

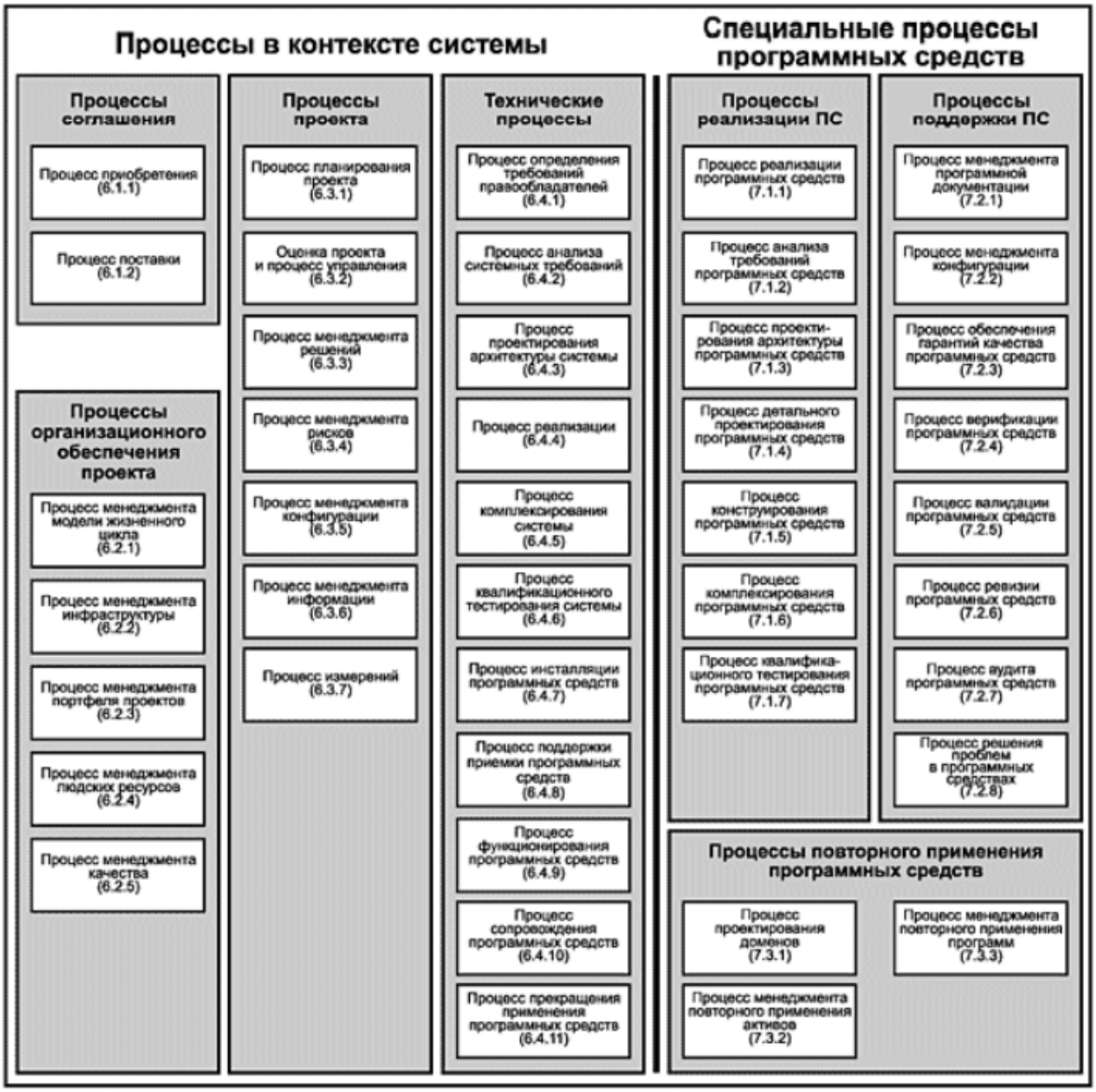
ГОСТ Р ИСО/МЭК 12207-2010 НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная технология Системная и программная инженерия ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ

1. Устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии.
2. Определяет процессы, виды деятельности и задачи, которые используются при приобретении программного продукта или услуги, а также при поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов.
3. Устанавливает процесс, который может использоваться при определении, управлении и совершенствовании процессов жизненного цикла программных средств.
4. Разработан для сторон, приобретающих системы, программные продукты и услуги, а также для поставщиков, разработчиков, операторов,

сопровожденцев, менеджеров (в том числе, менеджеров по качеству) и пользователей программных продуктов.

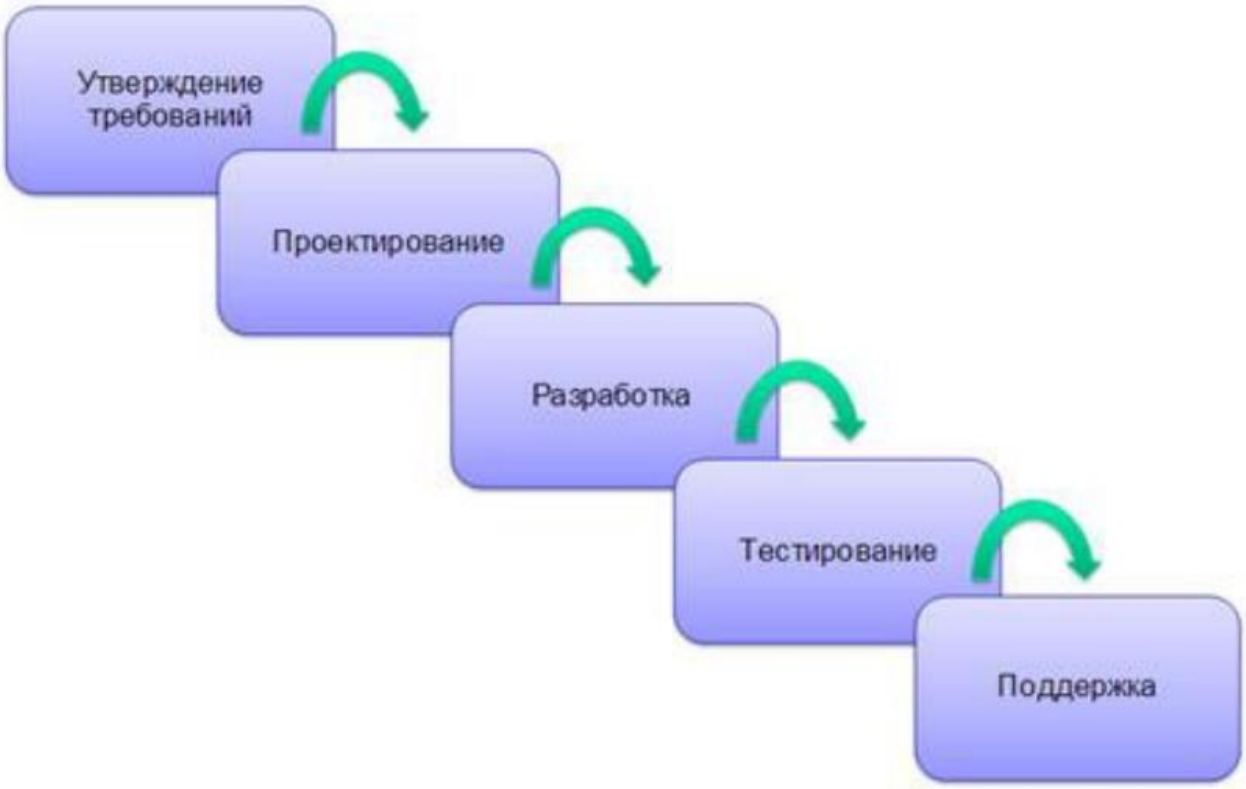
Группы процессов жизненного цикла



Модели разработки ПО

- Каскадная
- v-образная
- Итерационная
- Инкрементальная
- Спиральная
- Гибкая

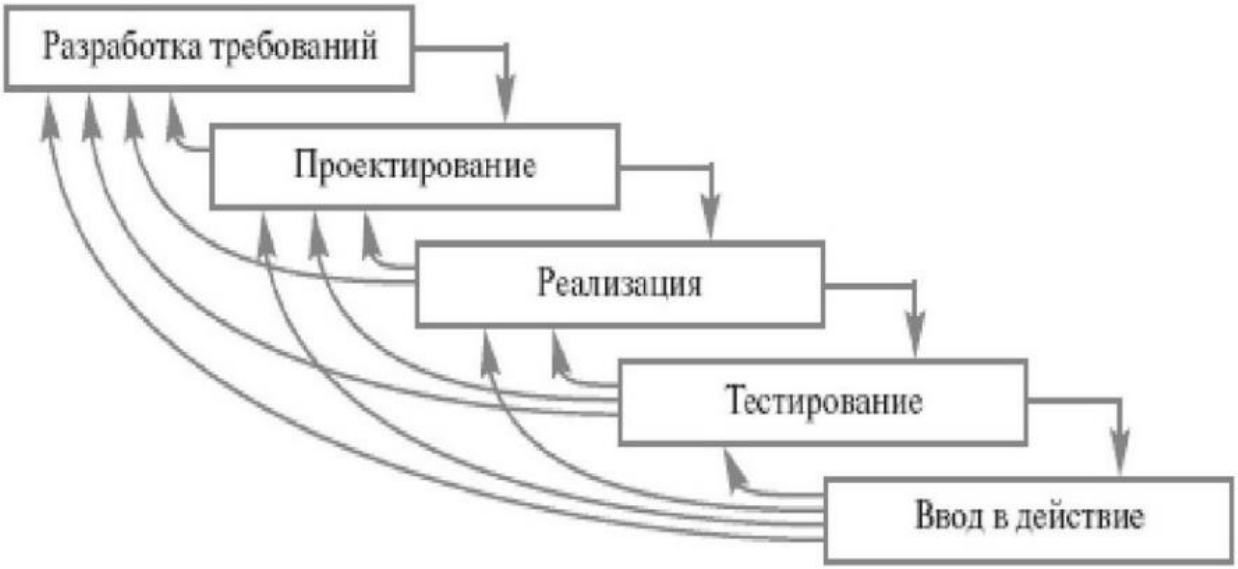
Каскадная модель (waterfall)



Особенности каскадной модели

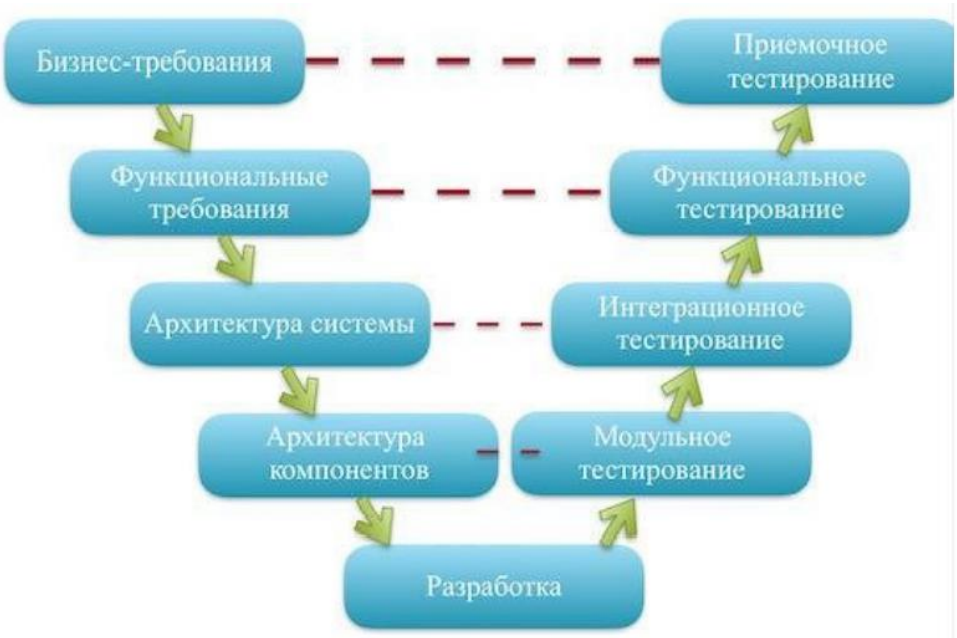
- высокий уровень формализации процессов;
- большое количество документации;
- жесткая последовательность этапов жизненного цикла без возможности возврата на предыдущий этап.

Итеративные или инкрементальные модели



Спиральная модель

V – модель



Agile «гибкая» модель

12 принципов Agile

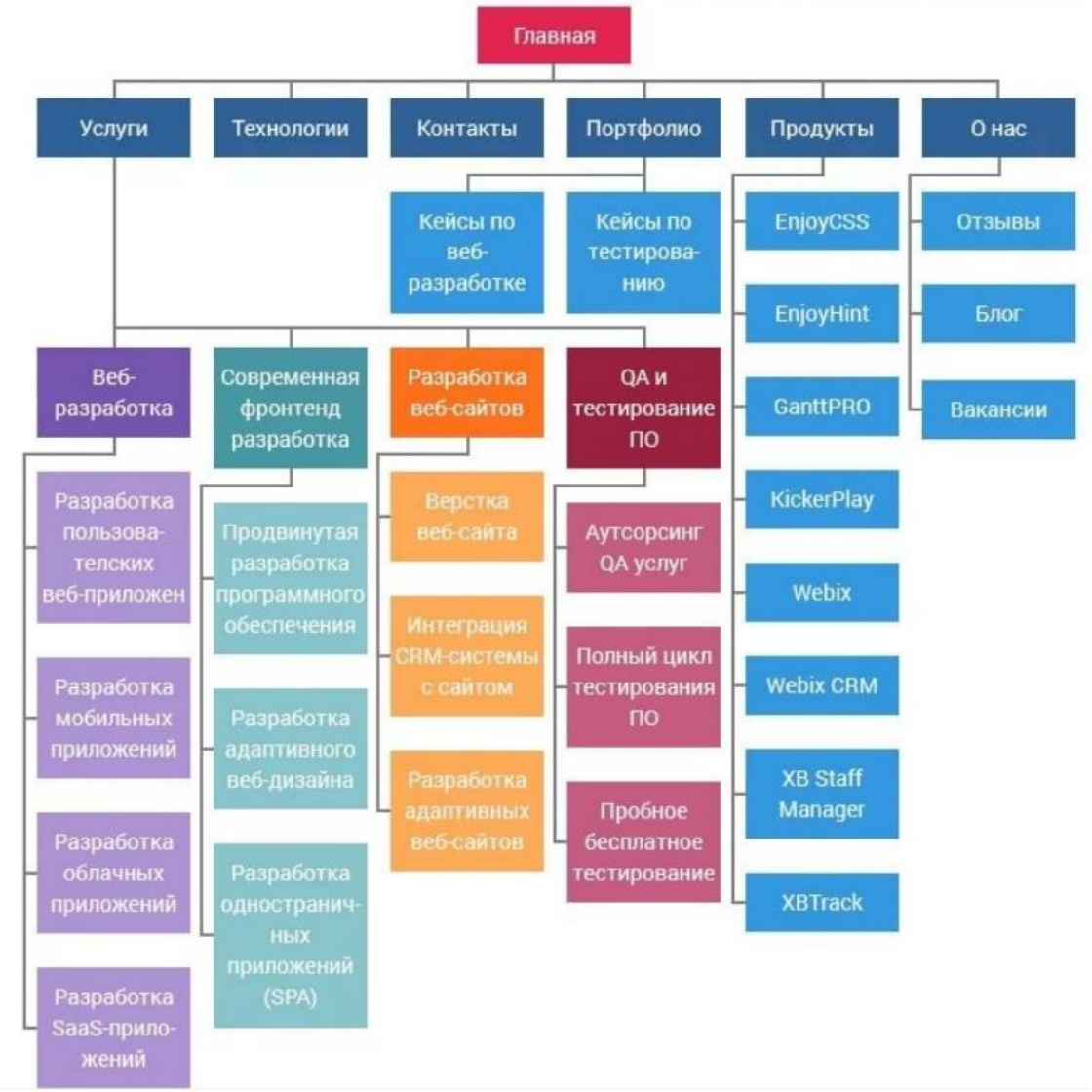
1. Регулярно и как можно раньше удовлетворять потребности заказчика, предоставляя ему программное обеспечение.
2. Учитывать, что требования могут измениться на любом этапе разработки.
3. Выпускать версии готовой программы как можно чаще.
4. Ежедневно вместе работать над проектом — разработчикам и заказчикам.
5. Поручить работу мотивированным профессионалам.
6. Общаться напрямую.
7. Считать главным показателем прогресса работающий продукт.
8. Поддерживать постоянный ритм работы.
9. Уделять пристальное внимание техническому совершенству и качеству проектирования.
10. Минимизировать лишнюю работу.
11. Стремиться к самоорганизующейся команде.

12. Всем участникам команды — постоянно искать способы повышать эффективность работы.

Анализ юзабилити сайта

Улучшение юзабилити посредством фокус-группы

Юзабилити структуры проекта и его навигации



Основные этапы разработки проблемно-центрированного дизайна

- Анализ задач и пользователей;
- Выбор репрезентативных задач;
- Заимствование;
- Черновое описание дизайна;
- Обдумывание дизайна;
- Создание макета или прототипа;
- Тестирование дизайна с пользователями;
- Итерирование;
- Реализация;

- Отслеживание эксплуатации;
- Изменение дизайна.

Кто и зачем собирается использовать разрабатываемую систему?

- Уровень знаний пользователя
- Область деятельности
- Общие характеристики

- Система должна запрашивать от пользователя информацию в порядке, который кажется ему естественным,
- Система должна давать возможность простой коррекции ошибок при вводе данных,
- Выбранные для организации интерфейса аппаратные средства должны вписываться в рабочую среду пользователя и быть эргономичными для его действий.

- Задачи, которые пользователи описали разработчикам
- Реальные задачи, с которыми сталкиваются пользователи
- Задачи должны достаточно полно покрывать всю функциональность системы
- Смесь простых и более сложных задач

Найти существующие интерфейсы, с помощью которых пользователи могут выполнить требуемую работу, и затем строить идеи новой системы на их базе

- Какие программы ваши пользователи используют сейчас?

- Найти существующие интерфейсы, с помощью которых пользователи могут выполнить требуемую работу, и затем строить идеи новой системы на их базе
- Чаще всего наилучшим вариантом будет придерживаться правилам старой системы

Черновое (грубое) описание разрабатываемой вами системы должно быть положено на бумагу (обязательно). Это позволяет задуматься о многих вещах. Но это описание не следует оформлять в виде компьютерной программы (пока), даже если вы умеете пользоваться какими-либо системами автоматизации разработки. Такие системы вынуждают вас прикрепляться к конкретным решениям, которые ещё слишком рано делать.

- ✓ Стоимость построения законченного пользовательского интерфейса и его тестирование с достаточным количеством пользователей для выявления всех проблем очень высока.
- ✓ Существует несколько структурных подходов, которые можно использовать, чтобы исследовать сильные и слабые стороны интерфейса до его программного воплощения.

CWT - позволяет находить места в дизайне, где пользователь может делать ошибки.

GOMS - оценка трудоёмкости выполнения задач по времени и выявление задач, требующих слишком много шагов.

«Думать вслух»

Тестирование

проанализировать результаты тестирования, соизмеряя **стоимость** корректировок с **серьёзностью** возникших

проблем, затем доработать интерфейс и протестировать его снова.

цель тестирования состоит не в том, чтобы доказать правильность интерфейса, а в том, чтобы улучшить его

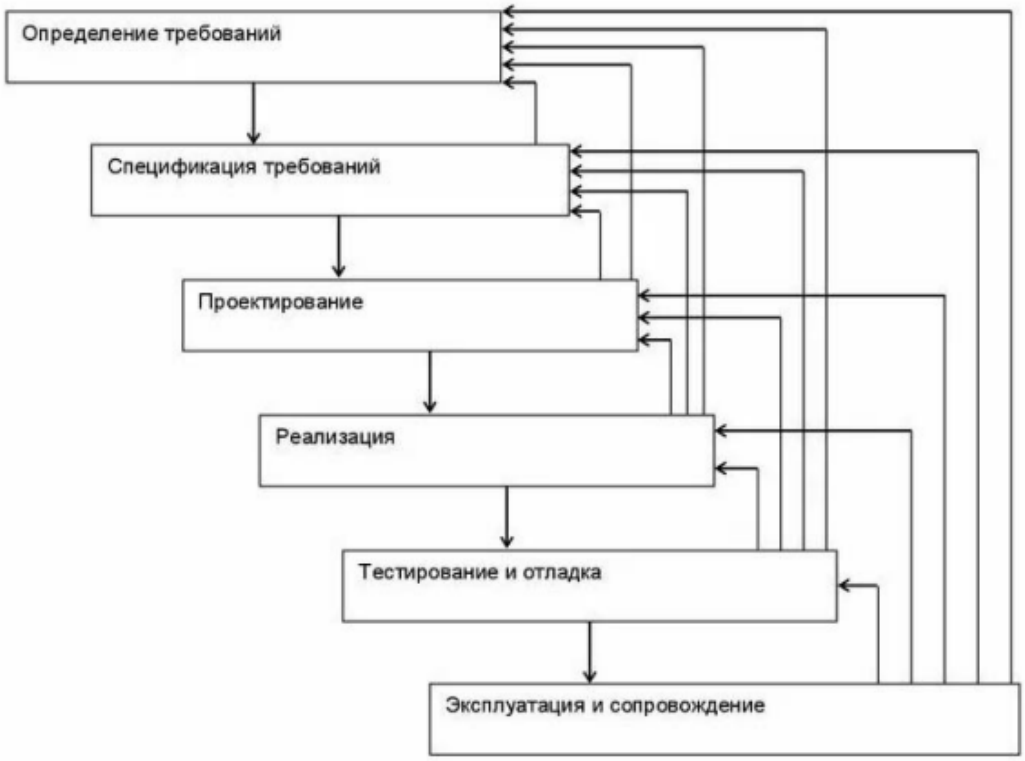
пользовательский интерфейс часто занимает более половины кода коммерческого продукта

команда разработчиков не должна быть изолирована от всей остальной деятельности, связанной с функционированием системы

Независимо от того, насколько удачно система была спроектирована первоначально, с большой вероятностью она будет **терять адекватность** с течением лет. Меняются и задачи и пользователи. **Приёмы** работы меняются из-за нового оборудования и программных продуктов. Пользователи приобретают **новые навыки** и ожидаемые реакции. Разработчики должны стоять вровень с этими изменениями, не только отслеживая состояние той **рабочей среды**, для которой была предназначена их система, но и **развитие** всего общества, технологий и методов, потребностей.

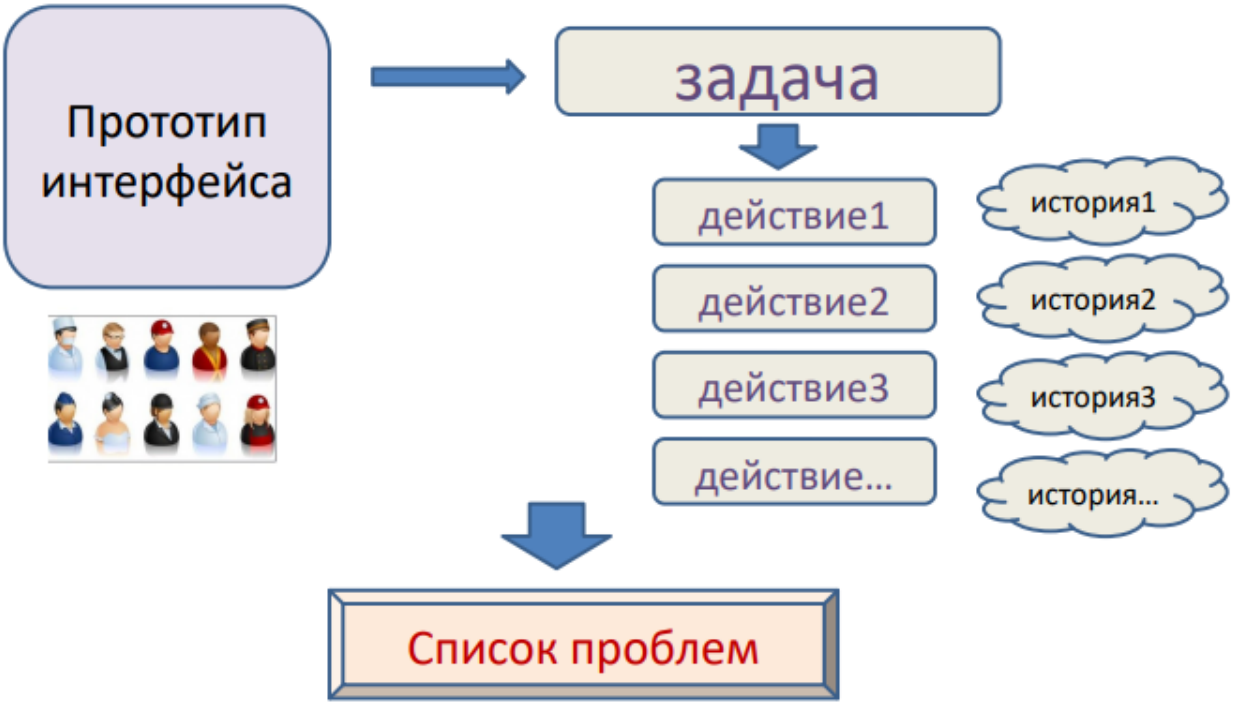
Метод водопада

Проблемно-центрированный подход



- ✓ Требования практичности – это целевые значения для таких характеристик, как скорость выполнения репрезентативных задач и допустимое количество ошибок.
- ✓ Эти показатели могут использоваться, чтобы мотивировать разработчиков и обосновывать решения по распределению ресурсов.

- ✓ Целевые значения могут быть выбраны так, чтобы побить конкурентов или обеспечить функциональные нужды для хорошо определённых задач.



- ✓ Он может поставить под сомнение ваши первоначальные и не вполне обоснованные предположения о том, как мыслит пользователь (вначале поставить диск, а потом запустить программу, или наоборот? кнопка с изображением дискеты означает копирование или сохранение?).
- ✓ Он может выявлять элементы управления, которые очевидны для разработчика, но могут быть скрыты от пользователя (список форматов выходного файла).
- ✓ Он может выявлять затруднения с надписями и подсказками (неудачное предупреждение "No media present").
- ✓ Он может обнаруживать неадекватную обратную связь, что может заставить пользователя сомневаться в результате и повторять всё с начала, хотя всё было

сделано правильно (отсутствие индикации пути, по которому сохраняется файл).

✓ Он может показывать недостатки в текущем описании интерфейса (слова "Start copying" вместо графического изображения кнопки в руководстве пользователя).

✓ Будут ли пользователи пытаться произвести тот или иной эффект, который даёт действие?

✓ Видят ли пользователи элемент управления (кнопку, меню, переключатель и т.д.) для осуществления действия?

✓ Если пользователи нашли элемент управления, поймут ли они, что он производит тот эффект, который им нужен?

✓ После того как действие сделано, будет ли понятен пользователям тот отклик, который они получают, чтобы перейти к следующему действию с уверенностью?

Операции в GOMS – это элементарные действия, которые нельзя разложить на более мелкие.

Шаг метода: нажатие на кнопку

- визуально определить местонахождение кнопки (мыслительная операция);
- навести на кнопку указатель мыши (внешняя операция);
- щелкнуть кнопкой мыши (внешняя операция).

K – нажатие клавиши;

B – клик кнопкой мыши;

P – наведение указателя мыши;

R – ожидание ответной реакции компьютера;

H – перенос руки с клавиатуры на мышь или наоборот;

D – проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

M – мыслительная подготовка (к осуществлению одной из перечисленных операций).

K 0.2 с

B 0.2 с

P 1.1 с

R 0.25 с (или больше)

H 0.4 с

D 2 с (или больше)

M 1.35 с

Время нажатия на клавишу:

К = 0.2 с (естественной последовательности)

К = 0.5 с (случайной последовательности)

К = 0.75 с (сложные коды)

Извлечения простого элемента знания из долговременной памяти 1.2 с

Извлечения простого элемента знания из краткосрочной памяти 0.6 с

Золотые правила построения интерфейсов

Правила Нильсена-Молиха (Nielsen, Molich)

1. **Простой и естественный диалог**:

- не должно присутствовать не относящейся к теме или редко используемой информации;
- "лучше меньше да лучше»;

- информация, которая выводится на экран, должна появляться в порядке, соответствующем ожиданиям пользователя

2. **Простой и естественный диалог**:

- не должно присутствовать не относящейся к теме или редко используемой информации;
- "лучше меньше да лучше»;
- информация, которая выводится на экран, должна появляться в порядке, соответствующем ожиданиям пользователя

3. **Минимизируйте загрузку памяти пользователя**:

- не заставляйте пользователя помнить вещи от одного действия к следующему;
- оставляйте информацию на экране до тех пор, пока она не перестанет быть нужной;
- хорошим стилем считается делать только один ряд закладок.

4. **Будьте последовательны**:

- у пользователей должна быть возможность изучить действия в одной части системы и применить их снова, чтобы получить похожие результаты в других местах.

5. **Обеспечьте обратную связь**:

- дайте пользователю возможность видеть, какой эффект оказывают его действия на систему.

6. **Обеспечьте хорошо обозначенные выходы**:

- Если пользователь попадает в часть системы, которая его не интересует, у него всегда должна быть возможность быстро выйти оттуда, ничего не повредив.

7. **Обеспечьте быстрые клавиши и ярлыки**:

- элементы быстрого доступа могут помочь опытным пользователям избегать длинных диалогов и информационных сообщений, которые им не нужны.

8. **Хорошие сообщения об ошибках**
- Хорошее сообщение об ошибке помогает пользователю понять, в чём проблема и как это исправить
9. **Предотвращайте ошибки**
- Всегда, когда вы пишете сообщение об ошибке, вы должны спросить себя, можно ли избежать этой ошибки?

Принципы организации графического интерфейса

1. **Принцип кластеризации**:
- организуйте экран в виде визуально разделённых блоков с похожими элементами управления, предпочтительно с названием для каждого блока;
- подобные команды должны быть в одном меню: это позволяет им быть визуально близко и идти под одним заголовком;
- команды, относящиеся к некоторой конкретной области функциональности, могут также быть показаны в диалоговых боксах, опять-таки в визуально определимых блоках.
2. **Принцип "видимость отражает полезность"**:
- делайте часто используемые элементы управления заметными, видимыми и легко доступными;
- прячьте или сжимайте редко используемые элементы;
3. **Принцип интеллектуальной последовательности**:
- используйте похожие экраны для похожих функций;
- экраны не должны выглядеть одинаково, если в действительности они должны отражать совершенно другие вещи;
- предупреждение о критической ошибке в системе реального времени должно иметь вид, значительно

отличающийся от экрана помощи или информационного сообщения.

4. **Принцип "цвет как приложение":**

- не полагайтесь на цвет как носитель информации;
- используйте цвет умеренно, чтобы лишь акцентировать информацию, передаваемую другими средствами;
- обязательно дайте дополнительные ключи для пользователей, не способных воспринимать изменения цвета;
- помните, что многие пользователи могут, и часто это делают, изменить цвет окон, подсветок и других системных объектов. Стройте ваш продукт так, чтобы он работал с пользователем, а не боролся с ним.

5. **Принцип уменьшения беспорядка:**

- не помещайте на экран слишком много всего;
- не пытайтесь наделить каждое меню собственным шрифтом или работать с большим набором размеров. Как правило, пользователи заметят не столько различия, сколько беспорядок.

Тест-дизайн

Тестовое покрытие

- Покрытие требований (Requirements Coverage) - оценка покрытия тестами функциональных и нефункциональных требований к продукту, путем построения матриц трассировки (traceability matrix).
- Покрытие кода (Code Coverage) - оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.
- Тестовое покрытие на базе анализа потока управления - оценка покрытия основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Техники тест-дизайна

- Эквивалентное Разделение
- Анализ Граничных Значений
- Причина / Следствие
- Предугадывание ошибки
- Исчерпывающее тестирование
- Парное тестирование

Использование техники анализа классов эквивалентности

1. Необходимо определить класс эквивалентности.
2. Затем нужно выбрать одного представителя от каждого класса.
3. Нужно выполнить тесты. На этом шаге мы выполняем тесты от каждого класса эквивалентности.

Пример: функция подсчета комиссии при отмене бронирования авиабилетов. Размер комиссии зависит от времени до вылета, когда совершена отмена

1. **Определим классы эквивалентности:**
   1 класс: время до вылета > 5 суток.
   2 класс: 24 часа < время до вылета < 5 суток.
   3 класс: 0 часов < время до вылета < 24 часа.
   4 класс: время до вылета < 0 часов (вылет уже состоялся).
2. **Выберем представителя от каждого класса**:

- время до вылета = 10 суток (тест из 1-го класса).
- время до вылета = 3 суток (тест из 2-го класса).
- время до вылета = 12 часов (тест из 3-го класса).
- время до вылета = -30 мин (тест из 4-го класса).

3. Выполним тесты:

- Отменим бронь за 10 суток до вылета и проверим, что комиссия составила 0%.
- Отменим бронь за 3 суток до вылета и проверим, что комиссия составила 50%.
- Отменим бронь за 12 часов до вылета и проверим, что комиссия составила 75%.
- Отменим бронь через 30 мин после вылета и проверим, что комиссия составила 100%.

Плюсы и минусы техники анализа классов эквивалентности

Плюс: Заметное сокращение времени и улучшение структурированности тестирования

Минус: При неправильном использовании техники, мы рискуем потерять баги

Использование техники анализа граничных значений

1. Выделить классы эквивалентности.
2. Определить граничные значения этих классов.
3. Понять, к какому классу будет относиться каждая граница.
4. Для каждой границы провести тесты по проверке значения до границы, на границе, и сразу после границы.

Пример: функция подсчета комиссии при отмене бронирования авиабилетов. Размер комиссии зависит от времени до вылета, когда совершена отмена:

1. Выделим классы эквивалентности:

- 1 класс: время до вылета > 5 суток.
- 2 класс: 24 часа < время до вылета < 5 суток.
- 3 класс: 0 часов < время до вылета < 24 час.
- 4 класс: время до вылета < 0 часов (вылет уже состоялся).

2. Определим границы:

- 5 суток.
- 24 часа.
- 0 часов.

3. Определим, к какому классу относятся границы:

- 5 суток – ко 2-му классу.
- 24 часа – ко 2-му классу.
- 0 часов – к 4-му классу.

4. Протестируем значения на границах, до и после них:

1. Отменим бронь за 5 суток + 1 секунду до вылета (или как можно ближе к границе, но слева от нее) и проверим, что комиссия равна 0%.
2. Отменим бронь ровно за 5 суток до вылета и проверим, что комиссия равна 50%.
3. Отменим бронь за 5 суток – 1 секунду до вылета и проверим, что комиссия равна 50%.
4. Отменим бронь за 24 часа + 1 секунду до вылета и проверим, что комиссия равна 50%.
5. Отменим бронь ровно за 24 часа до вылета и проверим, что комиссия равна 50%.
6. Отменим бронь за 24 часа - 1 секунду до вылета и проверим, что комиссия равна 75%.
7. Отменим бронь за 1 секунду до вылета и проверим, что комиссия равна 75%.
8. Отменим бронь ровно во время вылета и проверим, что комиссия равна 100%.
9. Отменим бронь спустя 1 секунду после вылета и проверим, что комиссия равна 100%.

Плюсы и минусы техники анализа граничных значений

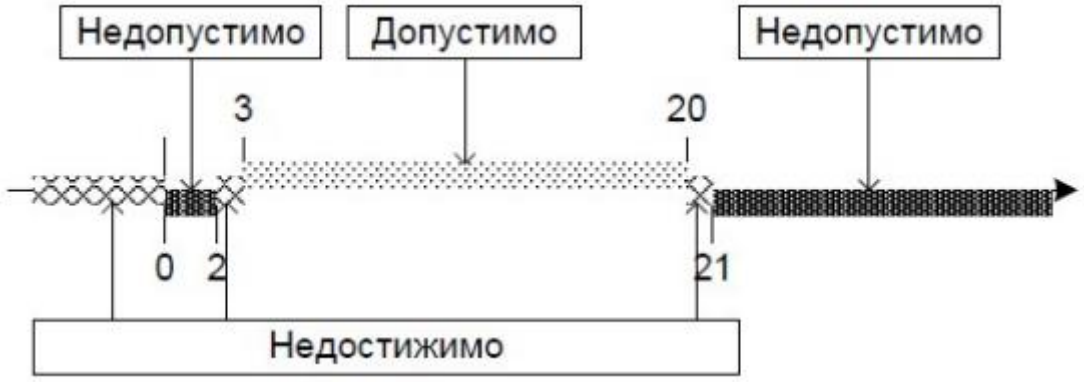
Плюсы:

- Эта техника добавляет в технику анализа классов эквивалентности ориентированность на конкретный тип ошибок.
- техника граничных значений ориентирована на обнаружение конкретной проблемы – возникновения ошибок на границах классов эквивалентности

Минусы:

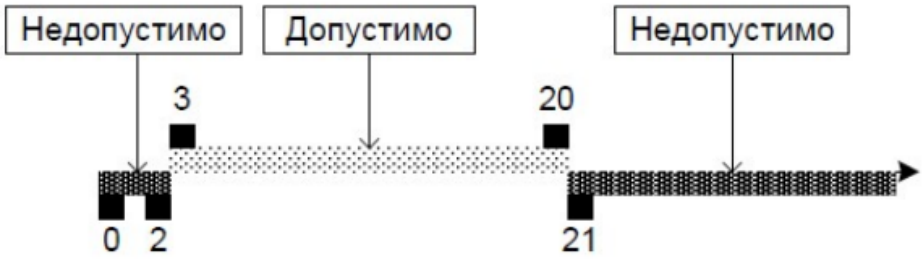
- эффективность техники анализа граничных значений зависит от правильности ее использования

Пример использования техники тестирования на основе классов эквивалентности и граничных условий



Итоговое разбиение на классы эквивалентности значений длины имени пользователя

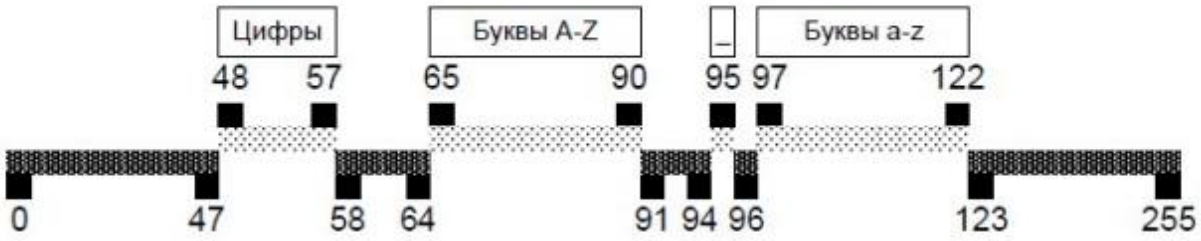
[0, 2] — недопустимая длина;
[3, 20] — допустимая длина;
[21, ∞] — недопустимая длина.



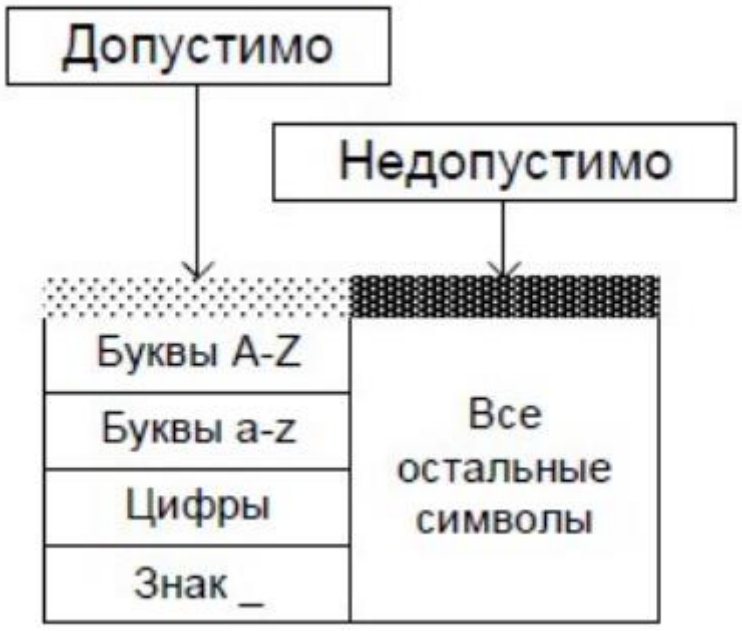
Значения входных данных для тест-кейсов (реакция на длину имени пользователя)

	Позитивные тест-кейсы		Негативные тест-кейсы		
Значение	AAA	123_zzzzzzzzzzzzzzz	AA	Пустая строка	1234_zzzzzzzzzzzzzzz
Пояснение	Строка мини-мальной допу-стимой длины	Строка максималь-ной допустимой длины	Строка недопусти-мой длины по нижней границе	Строка недопусти-мой длины, учтена для надёжности	Строка недопусти-мой длины по верх-ней границе

Неудачный способ поиска классов эквивалентности для наборов допустимых и недопустимых символов



Классы эквивалентных допустимых и недопустимых символов



Значения всех входных данных для тест-кейсов

	Позитивные тест-кейсы		Негативные тест-кейсы			
Значение	AAA	123_zzzzzzzzzzzzzz	AA	Пустая строка	1234_zzzzzzzzzzzzzz	#$%
Пояснение	Строка минимальной допустимой длины	Строка максимальной допустимой длины	Строка недопустимой длины по нижней границе	Строка недопустимой длины, учтена для надёжности	Строка недопустимой длины по верхней границе	Строка допустимой длины, недопустимые символы

Тестирование мобильных приложений

- Учесть все модели устройств
- Тестировать самую старую, самую новую ОС и браузеры
- Провести тест-кейсы
- Проверить удобство обновлений
- Проверить работу при слабом Wi-Fi
- Проверить взаимодействие с интерфейсом

Тестирование мобильных приложений. Размер экрана и touch-интерфейс

1. Все элементы должны быть такого размера, чтобы пользователь мог однозначно попасть по ним.
2. Отсутствие пустых экранов в приложении – пользователь не должен оказываться в ситуации, в которой не очевидно, что сейчас происходит и что делать.
3. Следует проверять многократное быстрое нажатие на кнопку – часто при этом может случиться падение приложения. Также следует проверять мультитач – нажатие на несколько кнопок одновременно.
4. Следует проверять наличие или отсутствие «нативных» жестов (pinch-to-zoom, doubletap) – если, например, поддерживается зум части приложения, то должен использоваться жест по умолчанию. А если нет

необходимости выделять картинку, то по даблтапу она не должна выделяться

Тестирование мобильных приложений. Ресурсы устройства

1. Утечки памяти - проявляется на окнах с большим количеством информации (длинные списки как пример), во время задач с длительным workflow (когда пользователь долго не выходит из приложения), при некорректно работающем кэшировании изображений.
2. Обработка ситуаций нехватки памяти для функционирования ОС, когда приложение активно или работает в фоне.
3. Недостаток места для установки или работы приложения.
4. Отсутствие в некоторых устройствах поддерживаемых приложением функций (3G, SD-карта).
5. Установка или перенос приложения на карту SD.

Тестирование мобильных приложений. Разрешения экрана и версии ОС

1. Ретина и обычные экраны.
2. Адаптация приложения к портретной и альбомной ориентациям устройства.
3. Версии ОС.
4. Поддержка необходимых медиа-файлов данной моделью и ОС.
5. Соответствие используемых в приложении view их смысловому назначению и концепциям платформы.

Тестирование мобильных приложений. Реакция приложения на внешние прерывания

1. Входящие и исходящие SMS, MMS, звонки, оповещения других приложений.
2. Выключение устройства, изъятие аккумулятора, разрядка устройства.

3. Переход в режим ожидания (в том числе и с защитой паролем). Смена ориентации устройства в режиме ожидания.
4. Отключение и подключение провода.
5. Отключение и включение сети, Bluetooth, авиарежима, GPS.
6. Потеря связи с сервером или прокси (подключение есть, но пакеты не доходят).
7. Отключение и подключение SD-карты, дополнительных устройств вроде физической клавиатуры или гарнитуры.
8. Зарядка устройства, работа с физической клавиатурой.

Тестирование мобильных приложений. Платный контент внутри приложения

1. Соответствие цены и содержимого, заявленного в приложении.
2. Восстановление покупки (обновление приложения).

Тестирование мобильных приложений. Интернационализация

1. Проверка корректности перевода.
2. Проверка того, что все надписи входят в соответствующие формы, кнопки и т.п.
3. Проверка форматов дат, разделителей в числах, специфических особенностей локализации (вроде пробела перед знаком вопроса во французской, верхних индексов “o” и “a”, в порядковых числительных в испанской и других нетривиальных моментах).

Тестирование мобильных приложений. Обновления

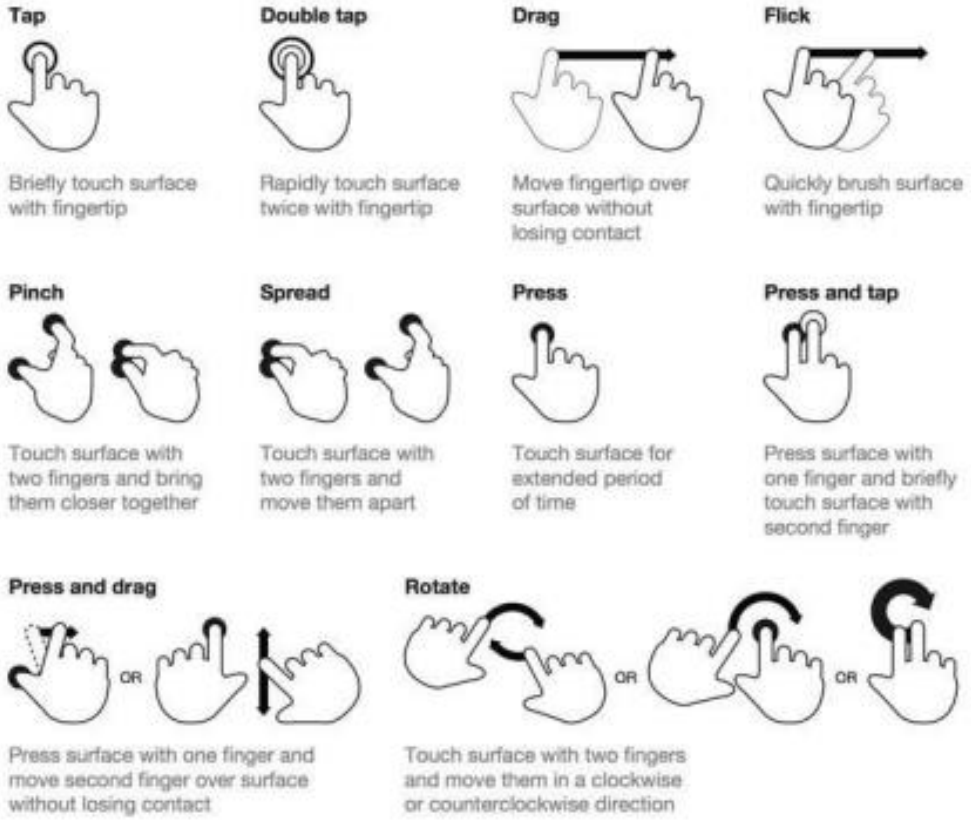
1. Убедиться, что поддерживаются те же версии ОС, что и предыдущая версия (если новая версия приложения использует новые возможности ОС, то для старых

поддерживаемых версий ОС необходимо создание урезанной версии приложения).
2. Проверка адекватного обновления (сохраняются все данные пользователя и т. п.).

Тестирование мобильных приложений. Постоянная обратная связь с пользователем

1. У всех нажимаемых элементов должно быть нажатое состояние (отклик на действие). В Android-приложениях у элементов может быть ещё одно состояние – focused.
2. Реакция кнопок на нажатие. Скорость отклика элементов должна быть достаточно высокой. Желательно использовать для проверки этого пункта самые слабые устройства среди поддерживаемых.
3. Сообщения при загрузке контента или прогресс-бар.
4. Сообщения при ошибке доступа к сети, GPS.
5. Наличие понятных сообщений при попытке удалить важную информацию.
6. Наличие экрана или сообщения при окончании процесса или игры.
7. Наличие и синхронность звуков или вибрации с уведомлениями и другими событиями на экране.

Тестирование мобильных приложений. Жесты.

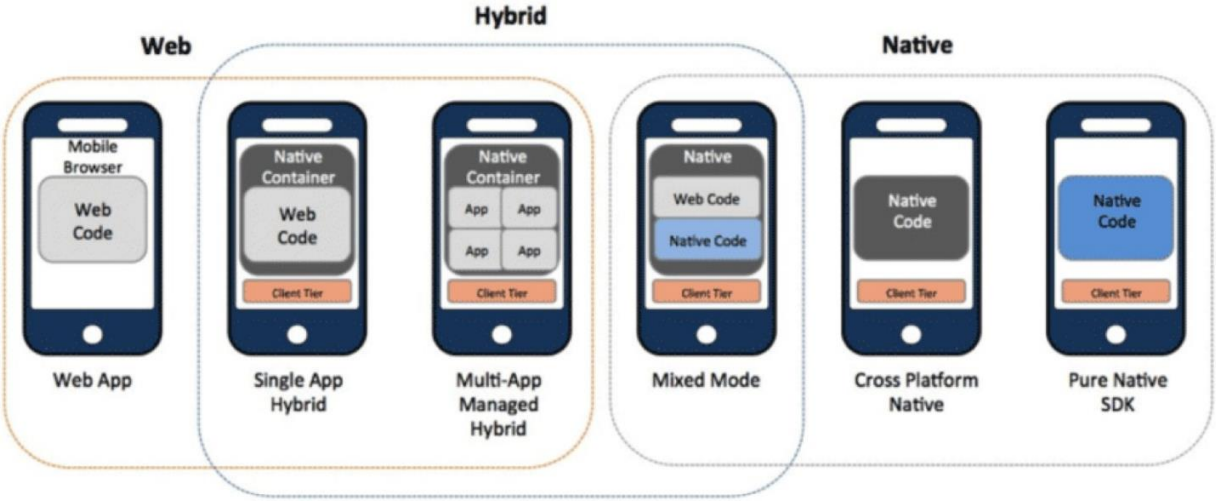


Типы мобильных приложений

Мобильные веб-приложения

Нативные приложения

Гибридные приложения



Инструменты для тестирования мобильных приложений

1. Эмуляторы устройств
2. DevTools
3. Сервисы TestFlight и Beta
4. Снифферы

Стандарты, регламентирующие процесс тестирования

- наличие целей тестирования на каждом уровне тестирования;
- любому процессу разработки соответствует свой процесс тестирования;
- возможность рецензирования документации тестировщиками начиная с самых первых версий;

- анализ требований и написание тестов начинается одновременно с деятельностью разработчиков или раньше.

- IEEE 12207/ISO/IEC 12207-2008 Software Life Cycle Processes – описывает жизненный цикл программного обеспечения и место различных процессов в нём;

- ISO/IEC 9126-1:2001 Software Engineering – Software Product Quality – описывает характеристики качества программных продуктов;

- IEEE 829-1998 Standard for Software Test Documentation – описывает виды документов, служащих для подготовки тестов;

- IEEE 1008-1987 (R1993, R2002) Standard for Software Unit Testing – описывает организацию модульного тестирования;

- ISO/IEC 12119:1994 Information Technology. Software Packages – Quality Requirements and Testing (аналог AS/NZS 4366:1996 и ГОСТ Р-2000, также принят IEEE под номером IEEE 1465-1998) – описывает требования к процедурам тестирования программных систем;

- ISO/IEC 20000:2005. Процессы, сертификация ISO 20000 – описывает требования к управлению и обслуживанию IT-сервисов

IEEE 12207/ISO/IEC 12207-2008 Software Life Cycle Processes

Под **жизненным циклом программного обеспечения (ЖЦ ПО)** понимается цикл создания и развития ПО, который начинается с момента принятия решения о необходимости создания программного продукта и завершается в момент изъятия программного продукта из эксплуатации.

Группировка процессов жизненного цикла ПО согласно IEEE 12207

Фактор	Атрибуты
Процессы соглашения	•Поставка; •приобретение
Процессы организационного обеспечения проекта	•Процесс менеджмента модели жизненного цикла; •процесс менеджмента инфраструктуры; •процесс менеджмента портфеля проектов; •процесс менеджмента людских ресурсов; •процесс менеджмента качества

Фактор	Атрибуты
Процессы проекта	•Процессы менеджмента проекта: ✓ процесс планирования проекта; ✓ процесс управления и оценки проекта; •процессы поддержки проекта: ✓процесс менеджмента решений; ✓процесс менеджмента рисков; ✓процесс менеджмента конфигурации; ✓процесс менеджмента информации; ✓процесс измерений

Фактор	Атрибуты
Технические процессы	•Определение требований правообладателей; •анализ системных требований; •проектирование архитектуры системы; •процесс реализации; •процесс комплексирования системы; •процесс квалификационного тестирования системы; •процесс инсталляции программных средств; •процесс поддержки приемки программных средств; •процесс функционирования программных средств; •процесс сопровождения программных средств; •процесс изъятия из обращения программных средств

Фактор	Атрибуты
Процессы реализации программных средств	•Процесс анализа требований к программным средствам; •процесс проектирования архитектуры программных средств; •процесс детального проектирования •программных средств; •процесс конструирования программных средств; •процесс комплексирования программных средств; •процесс квалификационного тестирования программных средств

Фактор	Атрибуты
Процессы поддержки программных средств	•Процесс менеджмента документации программных средств; •процесс менеджмента конфигурации программных средств; •процесс обеспечения гарантии качества программных средств; •процесс верификации программных средств; •процесс валидации программных средств; •процесс ревизии программных средств; •процесс аудита программных средств; •процесс решения проблем в программных средствах

Фактор	Атрибуты
Процессы повторного применения программных средств	• Процесс проектирования доменов; • процесс менеджмента повторного применения активов; • процесс менеджмента повторного применения программ

ISO/IEC 9126-1:2001 Software Engineering – Software Product Quality

- точку зрения разработчиков, которые воспринимают внутреннее качество ПО;
- точку зрения руководства и аттестации ПО на соответствие сформулированным к нему требованиям, в ходе которой определяется внешнее качество ПО;
- точку зрения пользователей, ощущающих качество ПО при использовании.

Факторы и атрибуты внешнего и внутреннего качества ПО согласно ISO 9126

Функциональность (functionality) – способность ПО в определенных условиях решать задачи, нужные пользователям. Определяет, что именно делает ПО

1. Функциональная пригодность (suitability) – способность решать нужный набор задач
2. Точность (accuracy) – способность выдавать нужные результаты
3. Способность к взаимодействию, совместимость (interoperability) – способность взаимодействовать с нужным набором других систем
4. Соответствие стандартам и правилам (compliance) – соответствие ПО имеющимся стандартам, нормативным и законодательным актам, другим регулирующим нормам
5. Защищенность (security) – способность предотвращать неавторизованный и неразрешенный доступ к данным, коммуникациям и другим элементам ПО

Надежность (reliability) – способность ПО поддерживать определенную работоспособность в заданных условиях

1. Зрелость, завершенность (maturity) – величина, обратная частоте отказов ПО. Определяется средним временем работы без сбоев и величиной, обратной вероятности возникновения отказа за данный период времени
2. Устойчивость к отказам (fault tolerance) – способность поддерживать заданный уровень работоспособности при отказах и нарушениях правил взаимодействия с окружением
3. Способность к восстановлению (recoverability) – способность восстанавливать определенный уровень

работоспособности и целостность данных после отказа в рамках заданных времени и ресурсов
4. Соответствие стандартам надежности (reliability compliance)

Удобство использования (usability), или практичность – способность ПО быть удобным в обучении и использовании, а также привлекательным для пользователей

1. Понятность (understandability) – показатель, обратный усилиям, которые затрачиваются пользователями на восприятие основных понятий ПО и осознание способов их использования для решения своих задач
2. Удобство обучения (learnability) – показатель, обратный усилиям, затрачиваемым пользователями на обучение работе с ПО
3. Удобство работы (operability) – показатель, обратный трудоемкости решения пользователями задач с помощью ПО
4. Привлекательность (attractiveness) – способность ПО быть привлекательным для пользователей
5. Соответствие стандартам удобства использования (usability compliance)

Производительность (efficiency), или эффективность, – способность ПО при заданных условиях обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам

1. Временная эффективность (time behaviour) – способность ПО решать определенные задачи за отведенное время
2. Эффективность использования ресурсов (resource utilisation) – способность решать нужные задачи с использованием заданных объемов ресурсов

определенных видов. Имеются в виду такие ресурсы, как оперативная и долговременная память, сетевые соединения, устройства ввода и вывода и пр.
3. Соответствие стандартам производительности (efficiency compliance)

Удобство сопровождения (maintainability) – удобство проведения всех видов деятельности, связанных с сопровождением программ

1. Анализируемость (analyzability), или удобство проведения анализа – удобство проведения анализа ошибок, дефектов и недостатков, а также удобство анализа необходимости изменений и их возможных последствий
2. Удобство внесения изменений (changeability) – показатель, обратный трудозатратам на выполнение необходимых изменений
3. Стабильность (stability) – показатель, обратный риску возникновения неожиданных эффектов при внесении необходимых изменений
4. Удобство проверки (testability) – показатель, обратный трудозатратам на проведение тестирования и других видов проверки того, что внесенные изменения привели к нужным результатам
5. Соответствие стандартам удобства сопровождения (maintainability compliance)

Переносимость (portability) – способность ПО сохранять работоспособность при переносе из одного окружения в другое, включая организационные, аппаратные и программные аспекты окружения

1. Адаптируемость (adaptability) – способность ПО приспосабливаться к различным окружениям без

проведения для этого действий, помимо заранее предусмотренных
2. Удобство установки (installability) – способность ПО быть установленным или развернутым в определенном окружении
3. Способность к сосуществованию (coexistence) – способность ПО сосуществовать в общем окружении с другими программами, деля с ними ресурсы
4. Удобство замены (replaceability) другого ПО данным – возможность применения данного ПО вместо других программных систем для решения тех же задач в определенном окружении.
5. Соответствие стандартам переносимости (portability compliance)

Факторы оценки качества ПО с точки зрения пользователей согласно ISO 9126

Фактор	Описание
Эффективность (effectiveness)	Способность решать задачи пользователей с необходимой точностью при использовании в заданном контексте
Продуктивность (productivity)	Способность предоставлять определенные результаты в рамках ожидаемых затрат ресурсов
Безопасность (safety)	Способность обеспечивать необходимо низкий уровень риска нанесения ущерба жизни и здоровью людей, бизнесу, собственности или окружающей среде
Удовлетворение пользователей (satisfaction)	Способность приносить удовлетворение пользователям при использовании в заданном контексте

IEEE 829-1998 Standard for Software Test Documentation

Артефакт	Описание
Описания тестовых процедур (test procedure specifications)	Тестовые процедуры могут быть представлены в виде скриптов или программ, автоматизирующих запуск тестовых сценариев (автоматизированное тестирование), или в виде инструкций для человека, следуя которым можно выполнить те же сценарии (ручное тестирование)
Отчеты о нарушениях (test incident reports, или bug reports)	Сценарии проведения отдельных тестов. Каждый тестовый сценарий предназначен для проверки определенных свойств некоторых компонентов системы в определенной конфигурации
Итоговый отчет о тестировании (test summary report)	Отчет, аккумулирующий общую информацию по результатам тестов, включающий достигнутое тестовое покрытие и общую оценку качества компонентов тестируемой системы

IEEE 1008-1987 (R1993, R2002) Standard for Software Unit Testing

- планирование – определение используемых методов тестирования необходимых ресурсов, критериев полноты тестов и критерия выхода;
- определение проверяемых требований и ограничений;
- уточнение, корректировка и детализация планов;
- разработка набора тестов;
- выполнение тестов;
- проверка достижения критерия выхода с последующей оценкой полноты тестирования по специально выбранным критериям;
- оценка затрат ресурсов и качества протестированных модулей

ISO/IEC 12119:1994 Information Technology. Software Packages – Quality Requirements and Testing

1) общие требования к содержанию;
2) обозначения и указания;
3) функциональные возможности;
4) надёжность;

5) практичность;
6) эффективность;
7) сопровождаемость и мобильность

ISO/IEC 20 000:2005. Процессы, сертификация ISO 20 000

1) **ISO 20 000-1:2005 «Information technology – Service management. Part 1: Specification»**

a) **процессы предоставления сервисов (Service delivery processes)** – группа процессов управления уровнем сервисов, предоставлением отчётности по предоставлению сервисов, составлением бюджета и статистики затрат;

b) **процессы управления взаимодействием (Relationship processes)** – описываются вопросы коммуникации между компанией, заказчиком и различными подрядчиками;

c) **процессы разрешения (Resolution processes)** – управление проблемами и инцидентами;

d) **процессы контроля (Control processes)** – описываются особенности контроля и управления изменениями в компании;

e) **процессы управления релизами (Release processes)** – описываются активности, связанные с внедрением новых и уже имеющихся решений.

2) **ISO 20000-2:2005 «Information technology – Service man-agement. Part 2: Code of Practice»**

- планирование (PLAN) – подразумевает ответы на вопросы «что?», «когда?» нужно сделать, а также «кто?» и «с помощью чего?» должен это сделать, т. е. на данном этапе проектируются или корректируются процессы;
- выполнение (DO) – включает выполнение запланированных на первом этапе работ;

- проверка (CHECK) – собираются метрики, готовится отчётность, согласно которой выявляется, какой результат дало выполнение работ;
- действие (ACT) – планы корректируются согласно результатам предыдущего этапа, проводятся требующиеся изменения.
- В стандарте также определяются требования к мерам ответственности руководителей компании, которая занимается разработкой, т. е. предоставляет ITсервисы, к компетенции персонала и управлению документацией

Преимущества механизма сигналов и слотов

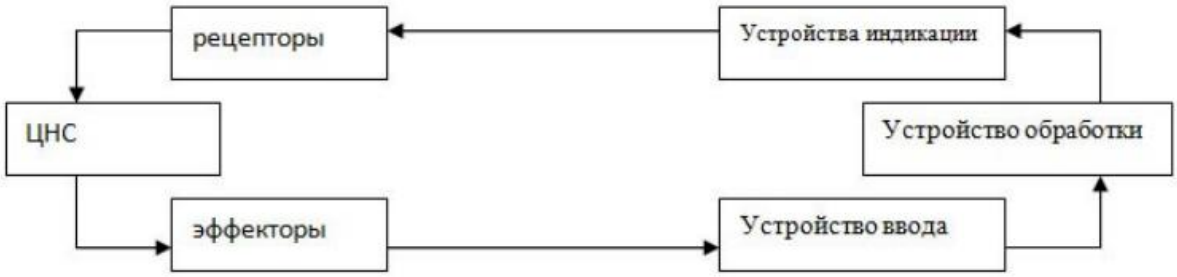
- каждый класс, унаследованный от QObject, может иметь любое количество сигналов и слотов;
- сообщения, посылаемые посредством сигналов, могут иметь множество аргументов любого типа;
- сигнал можно соединять с различным количеством слотов. Отправляемый сигнал поступит ко всем подсоединенным слотам;
- слот может принимать сообщения от многих сигналов, принадлежащих разным объектам;
- соединение сигналов и слотов можно производить в любой точке приложения;
- сигналы и слоты являются механизмами, обеспечивающими связь между объектами. Более того, эта связь может выполняться между объектами, которые находятся в различных потоках;
- при уничтожении объекта происходит автоматическое разъединение всех сигнально-слотовых связей. Это гарантирует, что сигналы не будут отправляться к несуществующим объектам;

Недостатки механизма сигналов и слотов

- сигналы и слоты не являются частью языка C++,;
- отправка сигналов происходит немного медленнее, чем обычный вызов функции, который осуществляется при использовании механизма функций обратного вызова;
- существует необходимость в наследовании класса QObject;
- в процессе компиляции не производится никаких проверок.

Основные вопросы человеко-машинного взаимодействия

Пользовательский интерфейс программы - это совокупность элементов, позволяющих пользователю программы управлять ее работой и получать требуемые результаты.



Правила создания диалоговых окон

Стремитесь к тому, чтобы диалоговое окно не содержало ничего лишнего и было как можно **проще**. В диалоговом окне настроек программы желательны только основные кнопки например **Ok, Cancel, Apply**

Объединяйте виджеты в **логические группы**, снабжая их прямоугольной рамкой и подписью. Используйте горизонтальные и вертикальные линии для разделения.

Никогда не делайте содержимое диалогового окна **прокручивающимся**. Если окно содержит много элементов, то постарайтесь разбить их на группы и разместить их **с помощью вкладок**.

Нежелательно, чтобы вкладки в диалоговом окне занимали **более одного ряда** – это усложняет поиск

Избегайте создания диалоговых окон с **неизменяемыми размерами**. Пользователь всегда должен иметь возможность увеличить или уменьшить размеры окна по своему усмотрению.

Сложные диалоговые окна лучше снабжать дополнительной **кнопкой Help (?),** при нажатии на которую должно открываться окно контекстной помощи.

Команды меню, вызывающие диалоговые окна, **должны оканчиваться многоточием**, например, Open… Settings… Это делается для того, чтобы пользователь заранее знал, что

нажатие команды меню приведет к открытию диалогового окна.

Старайтесь **не добавлять меню** в диалоговые окна. Меню должны использоваться в окне основной программы.

По возможности используйте стандартные виджеты, **хорошо знакомые пользователям**. Не забывайте, что для освоения новых элементов управления может понадобиться дополнительное время

Для показа настроек избегайте использования цвета. В большинстве случаев **текст – лучшая альтернатива**

Необходимо снабдить все элементы окна клавишами быстрого вызова, которые позволят, нажав букву совместно с клавишей Alt, установить фокус на нужном элементе.

Практика

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

```h
#ifndef FORM_H
#define FORM_H
#include <QtGui>
#include <QApplication>
#include <QWidget>

namespace Ui {
class Form;
}

class Form : public QWidget
{
    Q_OBJECT

public:
    explicit Form(QWidget *parent = 0);
    ~Form();

private:

    Ui::Form *ui;
};

#endif // FORM_H
```