Верификация и валидация программного обеспечения

Тестирование ПО

Тестирование ПО – запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности системы.

Тестирование – это динамический метод ***верификации*** и ***валидации*** (*аттестации*), так как применяется к исполняемой системе.

Верификация и валидация ПО

Верификацией и **валидаций** - называют процессы проверки и анализа, в ходе которых проверяется соответствие ПО своей спецификации и требованиям заказчика.

Верификация и **аттестация** охватывают полный жизненный цикл ПО, они начинаются на этапе анализа требований и завершаются проверкой программного кода на этапе тестирования готовой программной системы.

Верификация и валидация ПО

Верификацией и **валидацией** - называют процессы проверки и анализа, в ходе которых проверяется соответствие ПО своей спецификации и требованиям заказчика.

Верификация и **валидация** охватывают полный жизненный цикл ПО, они начинаются на этапе анализа требований и завершаются проверкой программного кода на этапе тестирования готовой программной системы.

Верификация и валидация ПО

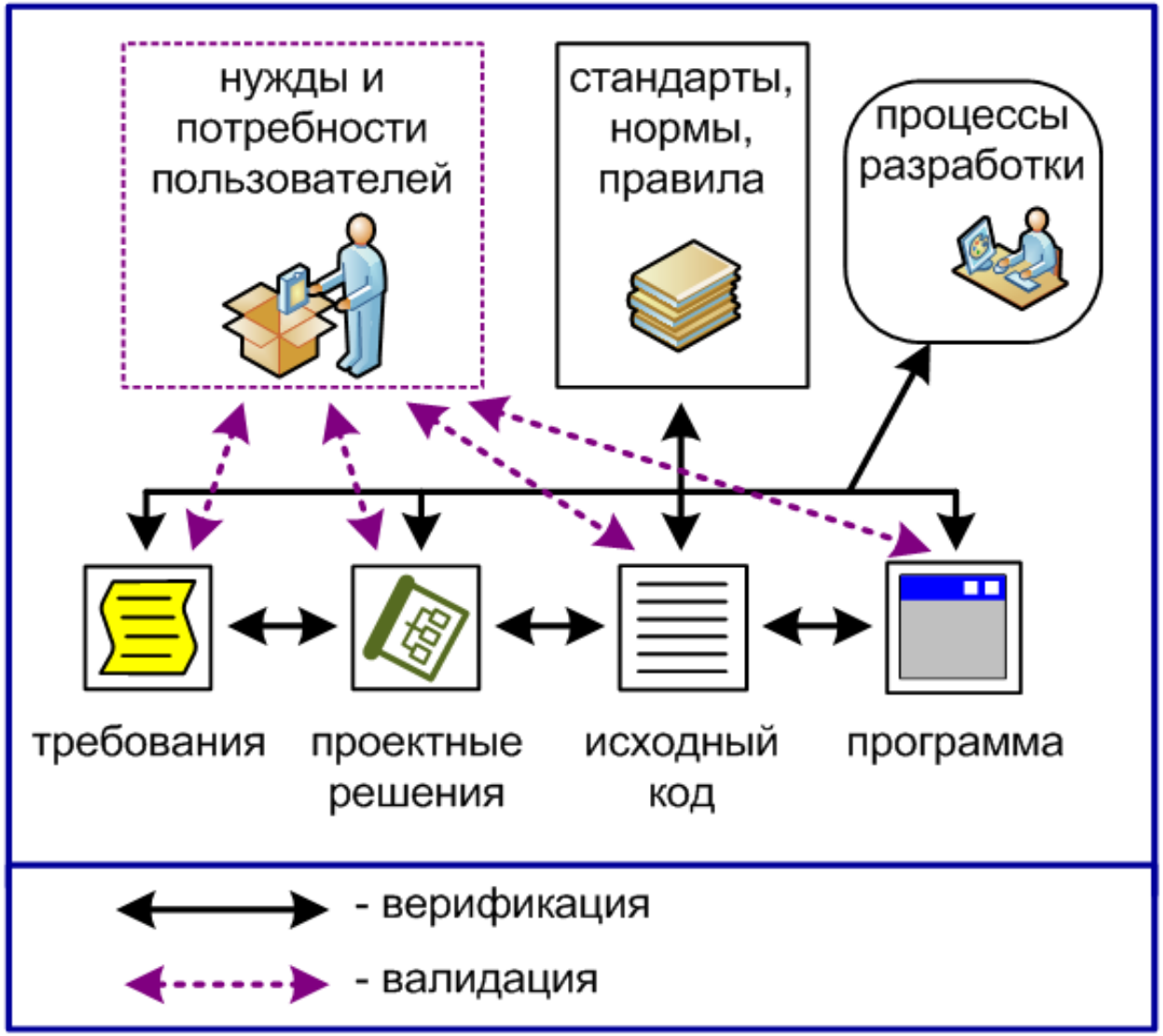
Верификация отвечает на вопрос, правильно ли создана система.

Валидация отвечает на вопрос, правильно ли работает система.

Верификация проверят соответствие ПО системной спецификации, в частности функциональным и нефункциональным требованиям.

Валидация – более общий процесс, во время проведения необходимо убедиться, что программный продукт соответствует ожиданиям заказчика, проводится после верификации.

Верификация и валидация ПО



Тестирование. Верификация. Валидация.

Тестирование отвечает на вопрос

- **«Как это сделано?»**
- **«Соответствует ли поведение разработанной программы требованиям?»**.

Верификация отвечает на вопрос

- ✓ **«Что сделано?»**
- ✓ **«Соответствует ли разработанная система требованиям?».**

Валидация отвечает на вопрос

- ❖ **«Сделано ли то, что нужно?»**
- ❖ **«Соответствует ли разработанная система ожиданиям заказчика?»**.

Верификация. Статические и динамические методы

Статические методы верификации выявляют неверные конструкции или неверные отношения объектов программы (ошибки формального задания) формальными методами анализа без выполнения тестируемой программы:

- ✓ С помощью специальных инструментов контроля кода
- ✓ Обзоры (*Reviews*)
- ✓ Инспекции (*Inspections*)
- ✓ Сквозные просмотры (*Walkthroughs*)
- ✓ Аудиты (Audits)
- ✓ Тестирование требований, спецификаций, документации.

Динамические методы верификации осуществляют выявление ошибок на выполняющейся программе.
Верификация заканчивается, когда выполнилось или "прошло" (pass) успешно достаточное количество тестов в соответствии с выбранным критерием тестирования.

Тестирование объектно-ориентированных ПС

Уровни тестирования

Применительно к объектно-ориентированным системам можно определить следующие уровни тестирования:

- *Тестирование отдельных методов (операций), ассоциированных с объектами*.
- *Тестирование отдельных классов объектов.*
- *Тестирование кластеров объектов.*
- *Тестирование сценариев и вариантов использования.*
- *Тестировании потоков*.
- *Тестирование взаимодействий между объектами*.

Тестирование отдельных методов

Обычно методы представляют собой функции или процедуры. Поэтому здесь можно использовать тестирование методами черного и белого ящиков.

Функциональное тестирование, или тестирование методом черного ящика базируется на том, что все тесты основываются на спецификации системы или ее компонентов. Поведение системы, как черного ящика, можно определить только посредством изучения ее входных и соответствующих им выходных данных, то есть проверяется не реализация ПО, а только ее выполняемые функции.

Метод структурного тестирования, так называемый метод белого ящика, предполагает создание тестов на основе структуры системы и ее реализации. Такой метод, как правило, применяется к относительно небольшим программным элементам, например к подпрограммам или методам, ассоциированными с объектами. При таком подходе тестировщик анализирует программный код и для получения тестовых данных использует знания о структуре компонента.

Тестирование отдельных классов объектов

Подход к тестовому покрытию систем требует, чтобы все операторы в программе выполнялись хотя бы один раз, а также, чтобы выполнялись все ветви программы. При тестировании объектов полное тестовое покрытие включает:

- раздельное тестирование всех методов, ассоциированных с объектом;

- проверку всех атрибутов, ассоциированных с объектом;

- проверку всех возможных состояний объекта, для чего необходимо моделирование событий, приводящих к изменению состояния объекта.

Использование наследования усложняет разработку тестов для классов объектов. Если класс представляет методы, унаследованные от подклассов, то необходимо протестировать все подклассы со всеми унаследованными методами.

Тестирование кластеров объектов

Следует применять методы тестирования, основанные на сценариях.

В объектно-ориентированных системах нет непосредственного эквивалента тестированию модулей. Однако считается, что группы классов, которые совместно предоставляют набор сервисов, следует тестировать вместе. Такой вид тестирования называется **тестирование кластеров**.

Тестирование сценариев и вариантов использования

Следует применять методы тестирования, основанные на сценариях.

Варианты использования (*Use Case*), или сценарии, описывают какой-либо один режим работы системы. Тестирование может базироваться на описании этих сценариев и кластеров объектов, реализующих данный Use Case.

Тестирование потоков

Этот подход основывается на проверке системных откликов на ввод данных или группу входных событий. Объектно-ориентированные системы, как правило, событийно-управляемые, поэтому для них особенно подходит данный вид тестирования. При использовании этого подхода необходимо знать, как в системе проходит обработка основных и альтернативных потоков событий.

Тестирование потоков

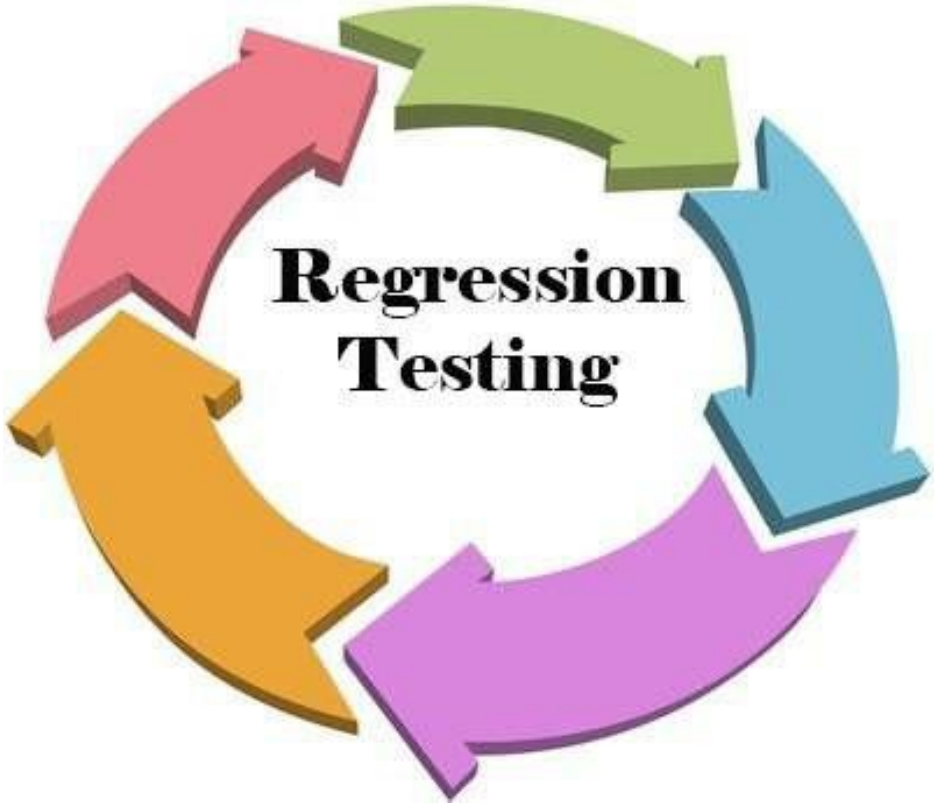
Этот подход основывается на проверке системных откликов на ввод данных или группу входных событий. Объектно-ориентированные системы, как правило, событийно-управляемые, поэтому для них особенно подходит данный вид тестирования. При использовании этого подхода необходимо знать, как в системе проходит обработка основных и альтернативных потоков событий.

Тестирование взаимодействий между объектами

Этот метод тестирования групп взаимодействующих объектов. Этот промежуточный уровень тестирования сборки системы основан на определении путей «метод - сообщение», отслеживающих последовательности взаимодействий между объектами.

Диаграммы последовательности UML можно использовать для определения тестируемых операций и для разработки тестовых сценариев.

Регрессионное тестирование



Тестирование ПО в процессе разработки

Как ПО изменяется в процессе разработки?

Тестирование ПО в процессе разработки

Как ПО изменяется в процессе разработки?

Поэтапно, небольшими независимыми шагами:

- Изменение уже существующего кода
- Исправление ошибок
- Добавление новой функциональности
- Адаптация имеющихся компонентов к новым задачам

Тестирование ПО в процессе разработки

Как ПО изменяется в процессе разработки?

Поэтапно, небольшими независимыми шагами:

- Изменение уже существующего кода
- Исправление ошибок
- Добавление новой функциональности
- Адаптация имеющихся компонентов к новым задачам

Любые (*даже самые незначительные*) изменения могут серьезно повлиять на качество ПО

Тестирование ПО в процессе разработки

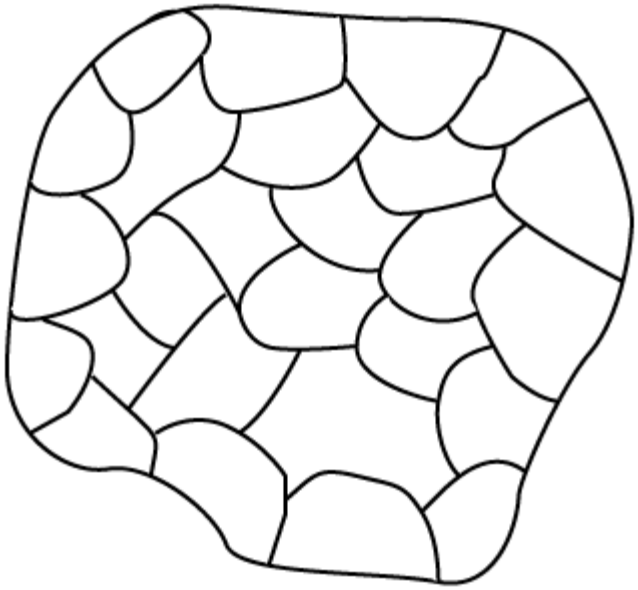
- После любого изменения требуется проверить, что в программе не появилось новых ошибок
- Для этого выполняют все имеющиеся тесты и проверяют, что все они успешно завершаются

Основной вид тестирования в процессе разработки ПО – это **регрессионное тестирование**

Регрессионное тестирование

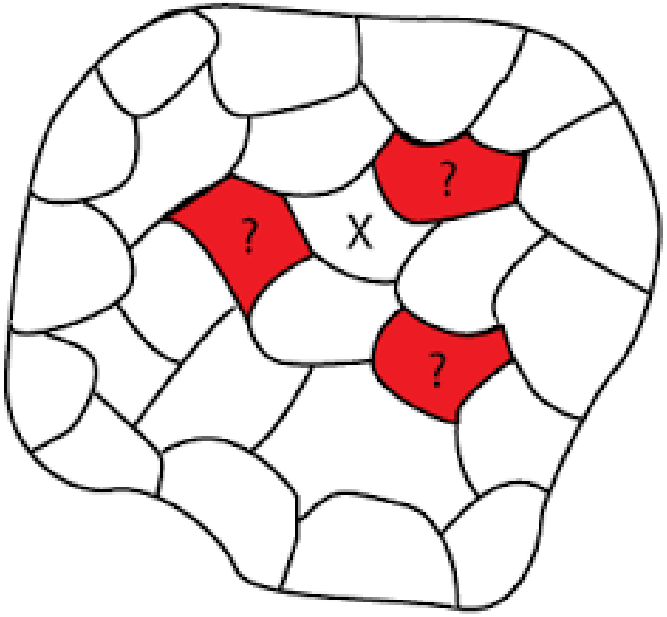
- Суть регрессионного тестирования в том, чтобы найти проблемы, возникшие в результате изменений продукта.

Предположим, есть продукт, состоящий из множества частей:



Регрессионное тестирование

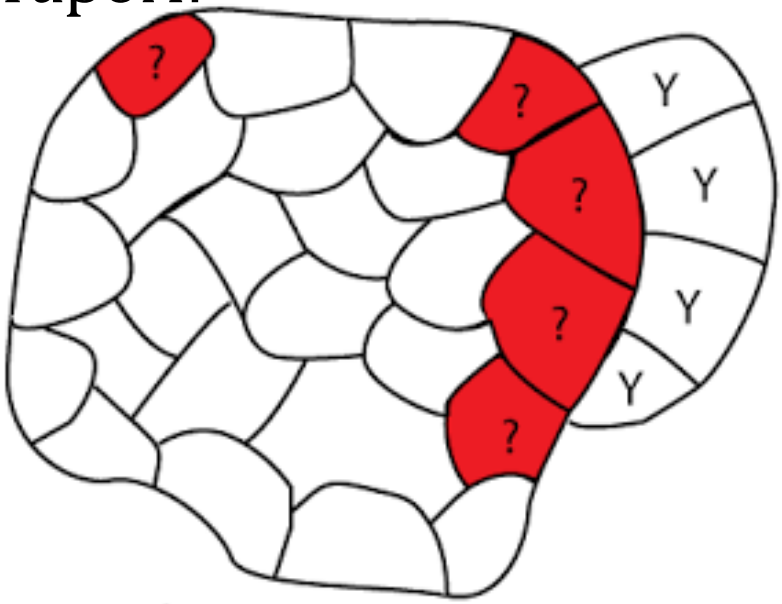
При изменении одной из его частей могут возникнуть проблемы в других частях:



X - изменение
? - возможные регрессионные проблемы

Регрессионное тестирование

- Либо добавление нового функционала приведет к ошибкам в старом:



Y - новые фичи
? - возможные регрессионные проблемы

Зачем проводить данный вид тестирования?
Одна из очевидных причин - минимизировать регрессионные риски. То есть, риски того, что при очередном изменении продукт перестанет выполнять свои функции.

Регрессионное тестирование

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.

Регрессионное тестирование. Цели

1. Удостовериться, что программа функционирует в соответствии со своей спецификацией, и что изменения не привели к внесению новых ошибок в ранее протестированный код. *(Эта цель всегда может быть достигнута повторным выполнением всех тестов регрессионного набора, но более перспективно отсеивать тесты, на которых выходные данные модифицированной и старой программы не могут различаться.)*
2. В соответствии с используемым критерием покрытия кода (например, критерием покрытия потока операторов или потока данных), гарантировать тот же уровень покрытия, что и при полном повторном тестировании программы. *(Для этого необходимо запускать тесты, относящиеся к измененным областям кода или функциональным возможностям.)*

Регрессионное тестирование.
Направления работы

Работа регрессионного тестирования ПО проводится по трем основным направлениям:

- **Багам**
- **Старым проблемам**
- **Побочным эффектам**

Регрессия багов

Попытка доказать, что исправленная ошибка на самом деле не исправлена.

К особенностям этого вида проверок относится необходимость контроля всех действий с конкретным объектом в разных комбинациях.

Прежде всего, тестируется соответствие реальности сообщения об избавлении от проблемы по механизму, который позволил ее обнаружить.

Регрессия старых ошибок

Попытка доказать, что недавнее изменение кода или данных сломало исправление старых ошибок, т.е. старые баги стали снова воспроизводиться. Другими словами, они вновь активны.

Таким образом, во время внесения изменений в код программного обеспечения следует с самого начала выполнять процессы тестирования.

Регрессия побочных эффектов

Попытка доказать, что недавнее изменение кода или данных сломало другие части разрабатываемого приложения.

В виде указания о присутствии подобных проблем представлено отсутствие работоспособности в каких-то частях программы.

Задачей тестирования в данном случае является определение всех проблемных мест.

Регрессионное тестирование.

Как выглядит одна итерация регрессионного тестирования?

1. Модифицируется программа P и получают программу P’
2. Из всего множества тестов T выбирают набор тестов T’, который необходимо выполнить на P’
3. Для новой функциональности разрабатывают новые тесты T’’
4. Полученный набор тестов T’ + T’’ запускается на P’
5. Результаты выполнения анализируются с последующей возможной модификацией как программы, так и набора тестов

Регрессионное тестирование.

Проблема №1

Как выбрать набор тестов T’ после изменения в программе?

Консервативный подход

1. Выбирают все имеющиеся тесты
2. Проводят полное регрессионное тестирование

Регрессионное тестирование.

Проблема №1

Как выбрать набор тестов T’ после изменения в программе?

Консервативный подход

1.
2.

Экономичный подход

1. Выбирают все тесты, каким-либо образом связанные по требованиям/ спецификации/ другими изменениями в коде
2. Проводят «выборочное» регрессионное тестирование

Регрессионное тестирование.

Проблема №1

Как выбрать набор тестов T’ после изменения в программе?

Консервативный подход

1.
2.

Экономичный подход

Умный подход

1. Выбирают все тесты, которые затрагивают при выполнении измененные части программы
2. Проводят «выборочное» регрессионное тестирование

Регрессионное тестирование

Каким свойствам должно удовлетворять выборочное регрессионное тестирование?

1. **Полнота.** Способность выбирать те тесты, которые могут обнаружить ошибки, связанные с изменениями в коде.
2. **Точность**. Способность пропускать такие тесты, которые не изменяют своего поведения на модифицированной программе.
3. **Эффективность**. Способность выполняться быстрее, чем полное регрессионное тестирование.
4. **Универсальность**. Применимость в большинстве практических ситуаций.

Управление регрессионными тестами

Проблема №2

Как управлять набором регрессионных тестов?

1. Когда и как добавлять в набор новые тесты?
2. Когда можно удалять старые тесты?

Управление регрессионными тестами

❶ Когда и как добавлять в набор новые тесты?

- Когда в ПО появилась новая функциональность
- Когда в ПО была исправлена ошибка
- Когда хотят улучшить тестовое покрытие ПО
- Когда можно позволить добавить новый неповторяющийся тест

Управление регрессионными тестами

❷ Когда можно удалять старые тесты?

- Никогда

Управление регрессионными тестами

❷ Когда можно удалять старые тесты?

- Никогда, но… :
- Когда тест дублирует другие тесты
- Когда тест не улучшает тестовое покрытие
- Когда тест ни разу не обнаружил ошибки за все время тестирования
- Когда тест обнаруживает такие же тесты, как и другие тесты

Приоритизация регрессионных тестов

Проблема №3

Как запускать регрессионные тесты?

Взяли имеющийся набор тестов и запустили их…

Приоритизация регрессионных тестов

Проблема №3

Как запускать регрессионные тесты?

Взяли имеющийся набор тестов и запустили их…

Такой подход может не всегда нас устраивать.
Что можно сделать / изменить в этом случае?

Приоритизация регрессионных тестов

Можно изменить порядок, в котором запускаются регрессионные тесты, когда:

- Чем раньше станет известно, что в ПО появилась регрессионная ошибка, тем скорее можно приступить к её исправлению
- Часто причиной непрохождения различных (напрямую не связанных друг с другом) тестом является одна и та же ошибка ПО
- Иногда время на тестирование является ограниченным, и необходимо найти наибольшее число ошибок с учетом всех ограничений

Как можно приоритизировать регрессионные тесты?

Приоритизация регрессионных тестов

1. При помощи интуиции

- Подход работает, если у Вас хорошая интуиция
- Кроме интуиции можно использовать имеющийся опыт разработки ПО

Приоритизация регрессионных тестов

1. При помощи интуиции
2. На основе знаний о тестовом покрытии ПО
   - Сперва выполняются тесты, которые имеют наибольшее покрытие ПО
   - Сперва выполняются тесты, которые покрывают более важные компоненты ПО

Приоритизация регрессионных тестов

1. При помощи интуиции
2. На основе знаний о тестовом покрытии ПО
3. На основе истории разработки

- Приоритет отдается тестам, которые чаще других обнаруживали регрессионные ошибки
- Первыми выполняются тесты, проверяющие корректность работы наиболее «проблемных» компонентов ПО

Приоритизация регрессионных тестов

1. При помощи интуиции
2. На основе знаний о тестовом покрытии ПО
3. На основе истории разработки
4. Случайным образом

- Подход перекликается со случайным ВРТ
- Если можно случайным образом поменять порядок выполнения тестов, то почему это не сделать ?

Приоритизация регрессионных тестов

1. При помощи интуиции
2. На основе знаний о тестовом покрытии ПО
3. На основе истории разработки
4. Случайным образом
5. На основе характеристик тестов

- Первыми выполняются тесты с наименьшим временем выполнения
- Приоритет отдается тестам, которые наиболее активно работают с окружением программной системы

Анализ результатов регрессионного тестирования

Проблема №4

Что делать с результатами регрессионного тестирования?

- ✓ Если все тесты проходят – всё хорошо
- ✓ Если тест не проходит – то все зависит от того, по какой причине он не проходит…

Анализ результатов регрессионного тестирования

- В тестируемом модуле есть ошибка
  - Ошибку необходимо локализовать и исправить
  - После исправления требуется выполнить повторное РТ
- Тест больше не является корректным
  - Например, была изменена спецификация модуля
  - Необходимо либо обновить тест, либо удалить его

Виды регрессионных тестов

1. **Верификационные тесты.** Проводятся для проверки исправления обнаруженного и открытого ранее бага.
2. **Тестирование верификации версии.** Содержит принципы **smoke test** и тестирование сборки: проверка работоспособности основной функциональности программы в каждой новой сборке.
3. **Непосредственно само регрессионное тестирование.** Повторное выполнение всех тестов, которые были написаны и проведены ранее. Они выполняются по уже существующим тест-кейсам независимо от того, были в ходе их прохождения найдены баги, или нет.
4. **Тестирование в новом билде уже исправленных багов в старых билдах.** Это выполняется для того, чтобы проверить, не возобновило ли обновление билда старых дефектов.

Регрессионное тестирование.
Организация процесса

1. **Сбор информации.** Изучают продукт и его окружение. Собирают информацию о релизах, о типичных изменениях в продукте, о критериях качества, о пропущенных в прошлом регрессионных багах.
2. **Формирование стратегии.** Принимают решения по стратегии регрессионного тестирования, которая является общей для всех релизов.
3. **Сбор информации о конкретном релизе.** Изучают изменения в конкретном релизе.
4. **Составление тест плана по регрессии.** Принимают решения по тестированию конкретного релиза.
5. **Выполнение регрессии.** Во время выполнения регрессионных тестов следят за процессом и анализируют найденные проблемы (или отсутствие проблем). Полученная информация используется для корректировки плана регрессии
6. **Работа над ошибками.** После проведения тестирования анализируют регрессионные проблемы, которые выявились, делают выводы. Если есть регрессионная библиотека тестов, то обновляют её с учетом последних изменений продукта.

Контрольные вопросы:

1. Что такое верификация и валидация.
2. Основные методы верификации.
3. Перечислить уровни тестирования объектно-ориентированного ПО.
4. Что представляет собой регрессионное тестирование. Цели РТ.
5. Описать направления РТ.
6. Каким свойствам должно удовлетворять выборочное РТ?
7. Виды регрессионных тестов.