Введение в тестирование ПО

Содержание

- Основы тестирования
- Модели жизненного цикла разработки
- Команда тестирования
- Типы и уровни тестирования
- Дефекты
- Портрет тестировщика ПО

Основы тестирования

Что такое тестирование?

Для начала мы ...

... удостоверяемся, все ли в порядке

Почему тестирование необходимо?

- Тестирование необходимо, потому что люди склонны ошибаться. Одни ошибки незначительны, другие же опасны и дорого обходятся.
- Поскольку ошибки допускают все люди, мы должны внимательно проверять результаты своей (и чужой ;-)) работы, всего, что мы делаем.

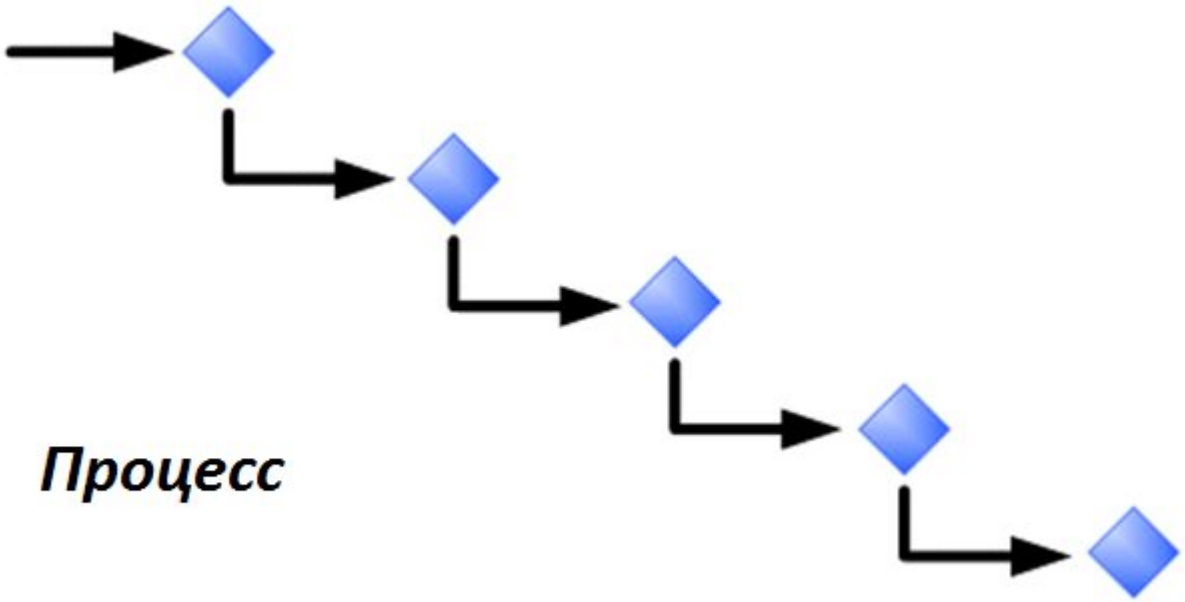
Что такое тестирование?

1) Процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для достижения заявленных целей, а также для нахождения дефектов.
2) Процесс выполнения программы с целью обнаружения ошибок.
3) Процесс поиска ошибок. Хороший набор тестов позволит найти больше ошибок, чем плохой, и ошибки эти будут более серьезными.
4) Наблюдение за функционированием ПО в специфических условиях с целью определения степени соответствия ПО требованиям к нему.
5) проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранным определенным образом.

Тестирование представляет собой процесс, который ставит своей целью признать точность, полноту, безопасность и качество разрабатываемого программного обеспечения.

Определение тестирования «по частям» 1/5

Во-первых, тестирование – это процесс, а не единичное действие



Определение тестирования «по частям» 2/5

Процесс тестирования включен во все активности жизненного цикла

Все активности жизненного цикла

Определение тестирования «по частям» 3/5

Тестирование ПО может быть статическим и динамическим

Статическое тестирование: Тестирование компонента или системы на уровне спецификации или реализации **без исполнения кода программного продукта**, например рецензирование или статический анализ кода.

Динамическое тестирование: Тестирование, проводимое **во время выполнения** программного обеспечения, компонента или системы.

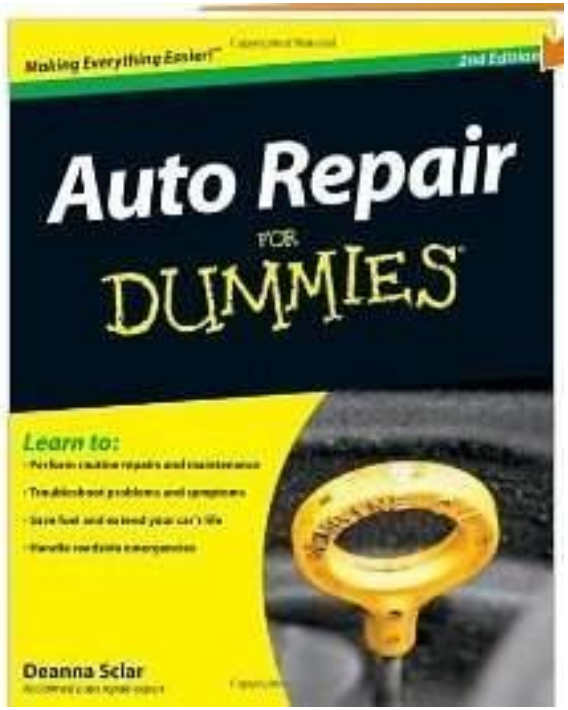
ISTQB International Software Testing Qualifications Board

Определение тестирования «по частям» 4/5

- Планирование
- Подготовка
- Оценка

Определение тестирования «по частям» 5/5

Тестированию подлежит программный продукт и связанные с ним рабочие продукты



Цели тестирования

- Предоставление информации для принятия решений
- Повышение уверенности в уровне качества
- Обнаружение дефектов
- Предотвращение дефектов

Тестирование помогает уменьшить общий уровень риска в системе после обнаружения и устранения дефектов и порождает уверенность в качестве ПО

Определение тестирования: сравнение как ключевое понятие

Тестирование всегда предполагает сравнение.

Что с чем сравнивается?

1. Объект тестирования (что сравнивается)
2. Базис тестирования (с чем сравнивается)

Терминология

Объект тестирования: Компонент или система, которые должны быть протестированы.

Базис тестирования: Документ, на основании которого определяются требования к компоненту или системе. Документация, на которой базируются тестовые сценарии.

Если правка данного документа может быть осуществлена только в процессе формальной процедуры внесения изменения, то такой базис тестирования называется **замороженным базисом тестирования.**

Рабочие продукты 1/2

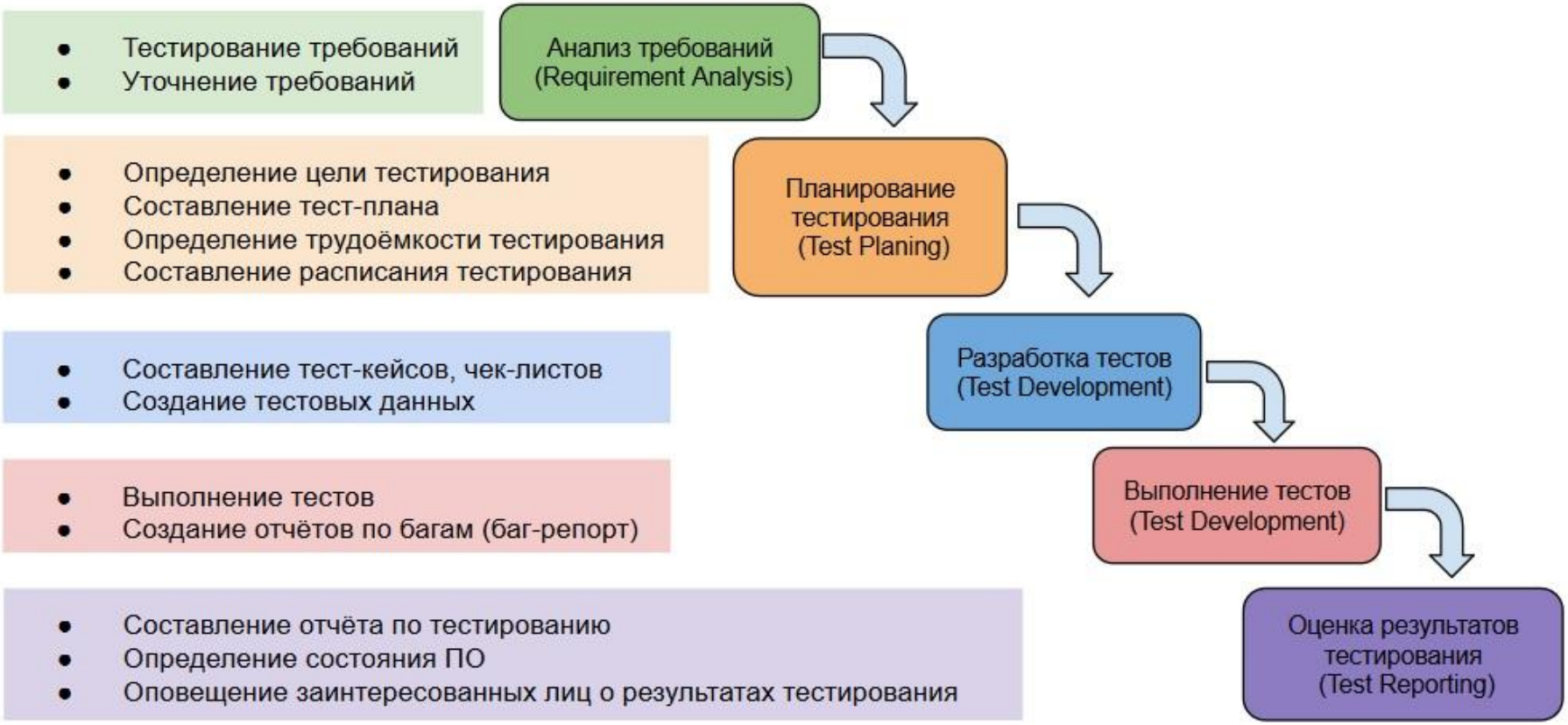
Рабочие продукты, поставляемые команде тестировщиков в качестве объектов тестирования, могут быть разными:

- отдельный модуль
- компонент (несколько модулей)
- подсистема
- система

Рабочие продукты 2/2

- документация с требованиями (маркетинговая, пользовательская, техническая)
- требования (функциональные , проектные, базы данных)
- модели, диаграммы, макеты
- сценарии использования
- код
- тестовые планы и сценарии
- проектная документация по автоматизации тестирования, код автоматизации

Описание процесса тестирования



Что такое дефект?

Дефект: Изъян в компоненте или системе, который может привести компонент или систему к невозможности выполнить требуемую функцию, например неверный оператор или определение данных. Дефект, обнаруженный во время выполнения, может привести к отказам компонента или системы.

Баг – синоним слова «дефект»

Как определить дефект перед нами или нет?

1. Программа не делает чего-то, что она должна делать согласно техническим требованиям.
2. Программа делает что-то, чего она не должна делать согласно техническим требованиям.
3. Программа делает что-то, о чем в требованиях не упоминалось.
4. Программа не делает чего-то, о чем не говорится в требованиях, однако подразумевается, что она должна делать это.
5. Программа трудна для понимания, неудобна в использовании

Связанные понятия: ошибка и отказ 1/2

Люди делают ошибки.

Если кто-то допустит ошибку в архитектуре или коде программы, то эта программа будет содержать дефект.

При исполнении программы любой дефект может привести к отказу.

Связанные понятия: ошибка и отказ 2/2

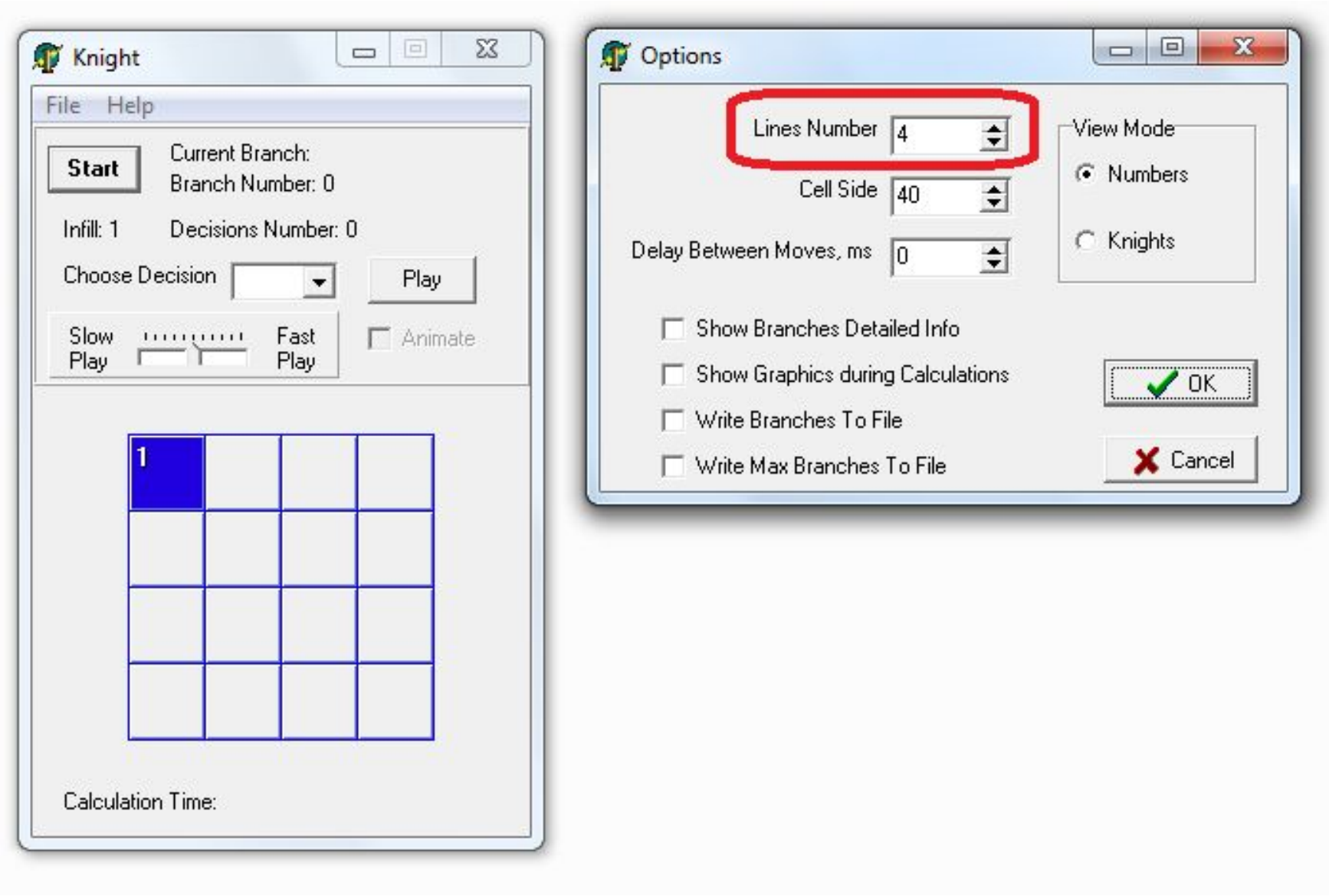
Ошибка: Действие человека, которое приводит к неправильному результату .

Отказ: Отклонение компонента или системы от ожидаемого выполнения, эксплуатации или результата.

Демонстрация дефекта - Требования

- На примере программы TestKnight
- Фрагмент требований:
  - **Диалоговое окно Опций**
    - **Lines Number** – размер шахматной доски (3…10)
    - **Cell Side** – размер стороны клетки в пикселях
    - **Delay Between Moves, ms** – пауза между движениями в процессе вычислений (0…5000). Используется для понимания принципов работы программы. Данную опцию следует использовать вместе с опцией
    - **Show …**
    - **…..**

Демонстрация дефекта - Программа



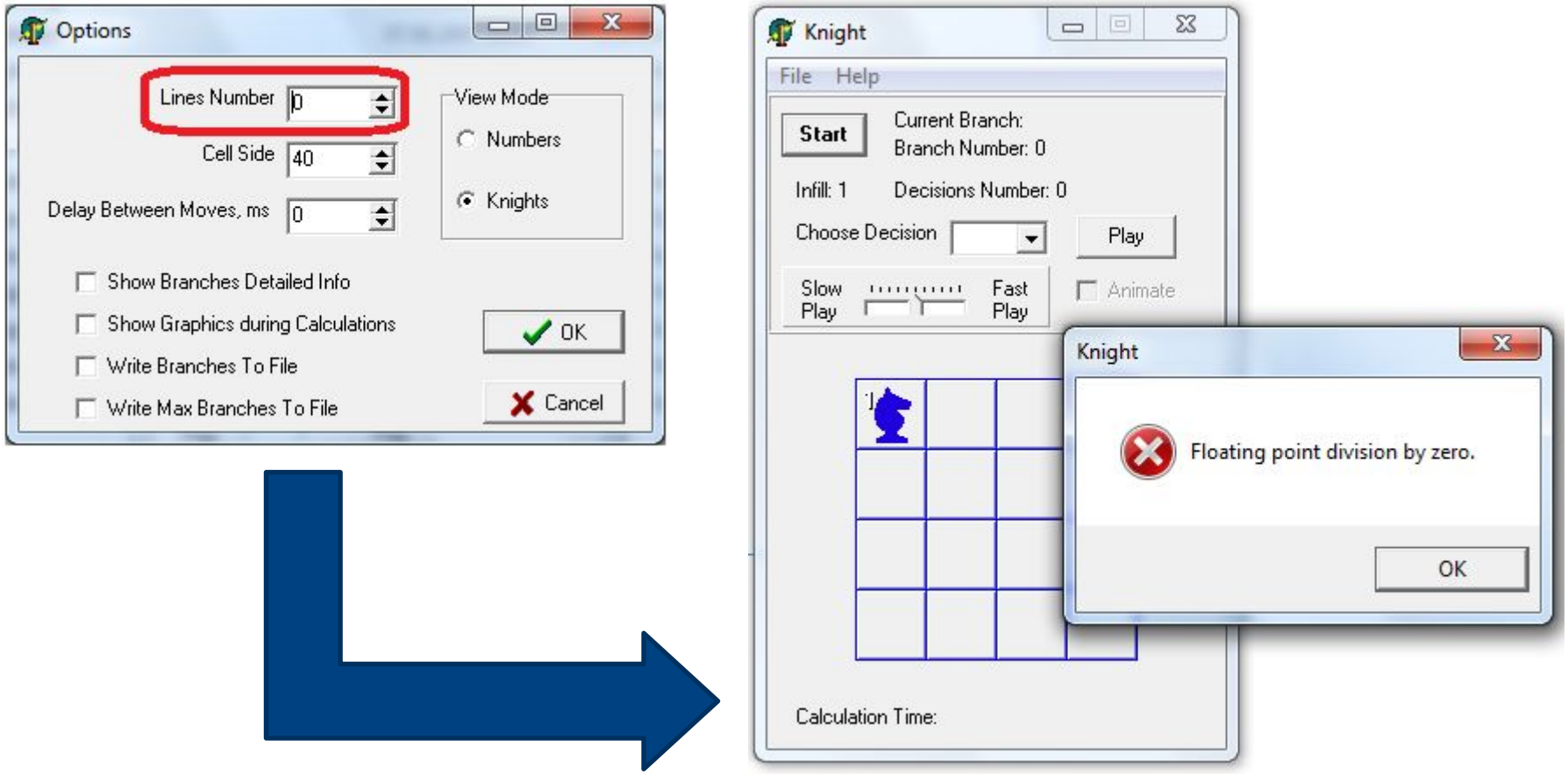
Демонстрация дефекта – Ошибка кодирования

- Нет проверки (забыли ;-)) на диапазон обозначенный в требований 3-10

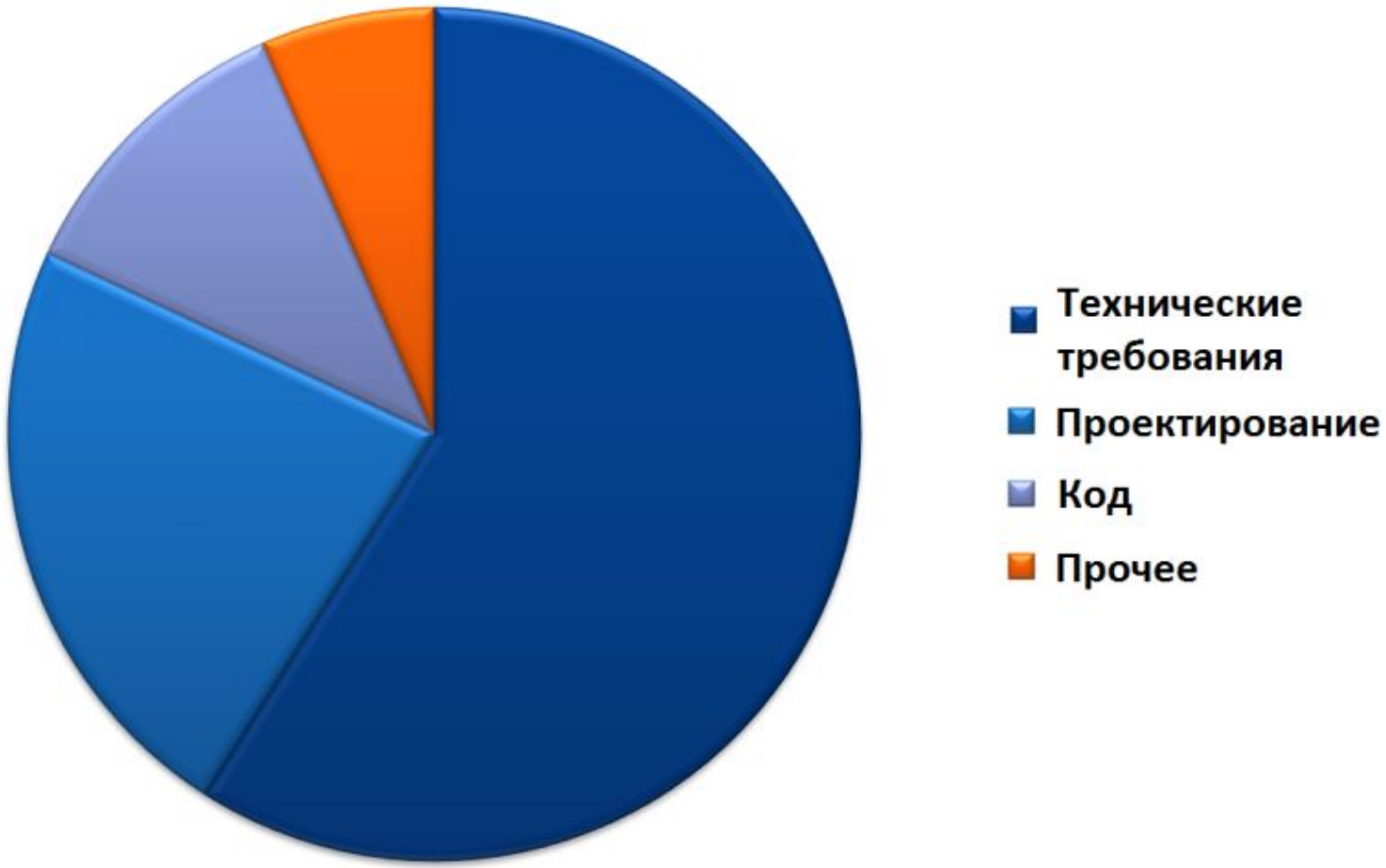
```
procedure TBoard.GetOptions;
{ Gets option significance from Frm_Options }
begin
  Frm_Knight.DeselectStartPosMode;

  SetOptions;

  Frm_Options.ShowModal;

  if (NumLines <> Frm_Options.Spin_NumLines.Value) or
    (NeedWriteToFile <> Frm_Options.Chk_WriteBranchesToFile.Checked) or
    (NeedWriteToFileMax <> Frm_Options.Chk_WriteMaxBranchesToFile.Checked) then
  begin
    NumLines := Frm_Options.Spin_NumLines.Value;

    NeedWriteToFile := Frm_Options.Chk_WriteBranchesToFile.Checked;
    NeedWriteToFileMax := Frm_Options.Chk_WriteMaxBranchesToFile.Checked;

    ChessBoardZeroing;
    Initialize;
    SetStartPos(1, 1);

    Frm_Knight.Cmb_Decisions.Items.Clear;
    DecisionsList.Clear;
    NumDecisions := 0;
  end;
```

Демонстрация дефекта – Сбой

- Вводим в параметрах значение 0
- Нажимаем Ок



Источники дефектов 1/2



Дефекты появляются по разным причинам, но, как правило, их источником являются технические требования (спецификация).

Источники дефектов 2/2

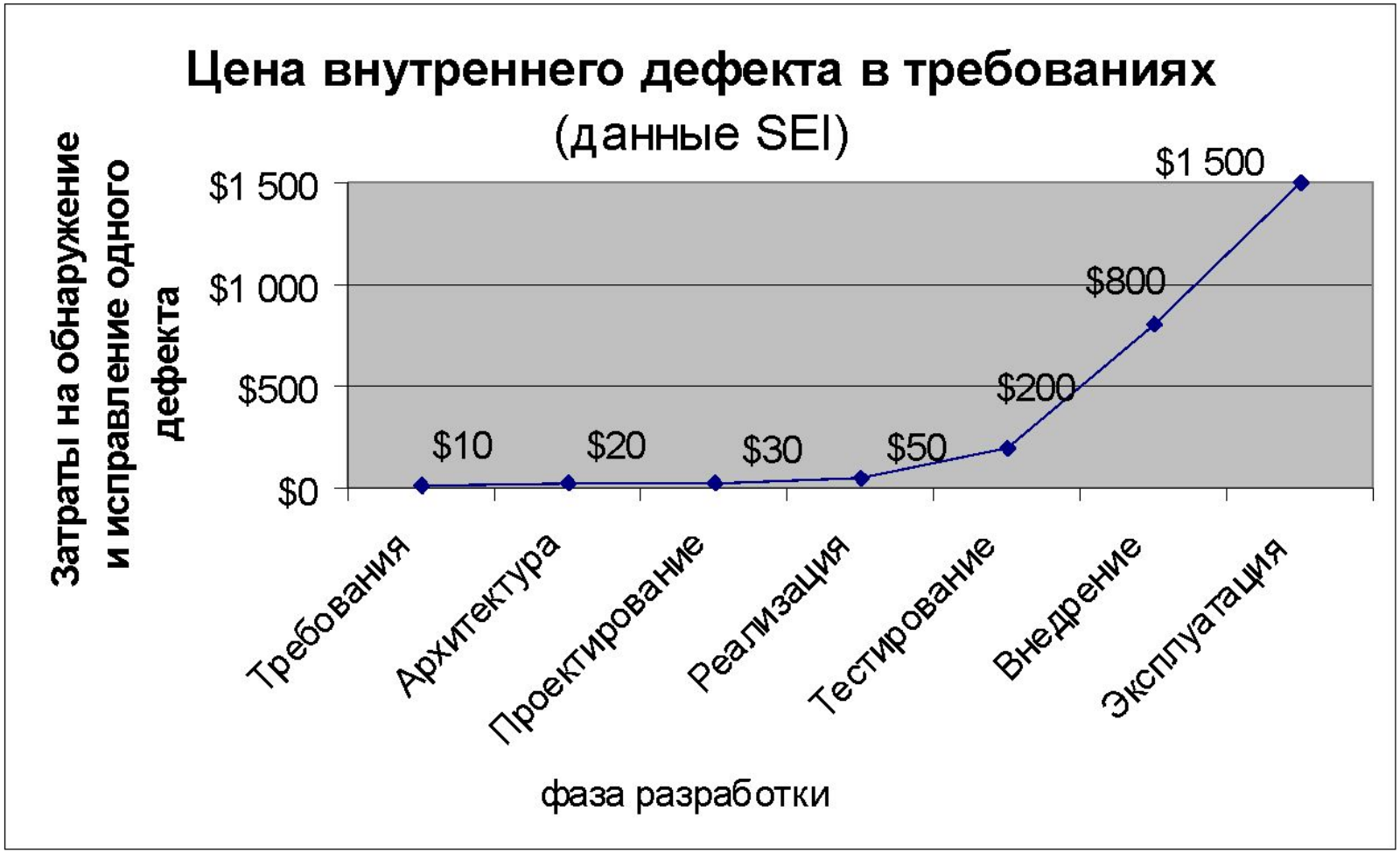
Source: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black "Foundations of Software Testing"

Цена дефектов 1/2

Обнаружение и исправление дефекта программы после поставки обходится в **100 раз** дороже, чем на стадии формирования требований и проектирования.

Как отмечал Боэм в 1987 г., «именно это понимание заставляло разработчиков уделять главное внимание тщательному анализу требований и проектированию, ранней верификации и валидации, а также моделированию и имитации, которые помогали избежать затратных послепродажных работ по устранению неисправностей».

Цена дефектов 2/2



Чем раньше дефект обнаружен, тем дешевле обходится его исправление

Терминология: «верификация» vs. «валидация» 1/3

Верификация: Доказанное объективными результатами исследования подтверждение того, что определенные требования были выполнены

Терминология: «верификация» vs. «валидация» 2/3

Валидация - определение соответствия разрабатываемого ПО **ожиданиям и потребностям пользователей**

ISTQB International Software Testing Qualifications Board

Терминология: «верификация» vs. «валидация» 3/3

«Вы создаете **продукт** **правильно** ?»

«Вы создаете **правильный** **продукт**?»

Тестирование и качество 1/9

Что такое качество?

«Качество – это ценность для индивидуума...»

Дж. Вайнберг (1992)

Тестирование и качество 2/9

Вопрос:

Отвечает ли тестировщик за качество?

Думаем – 2 мин.

Тестирование и качество 3/9

В IT-индустрии широко используется два понятия, которые напрямую связаны с тестированием программных продуктов:

- обеспечение качества (QA)
- контроль качества (QC)

Тестирование и качество 4/9

Тестирование и качество 5/9

В контроль качества входят:

- Тестирование
- Рецензирование кода
- Статический анализ кода
- Внешняя оценка и аудит

В обеспечение качества входят :

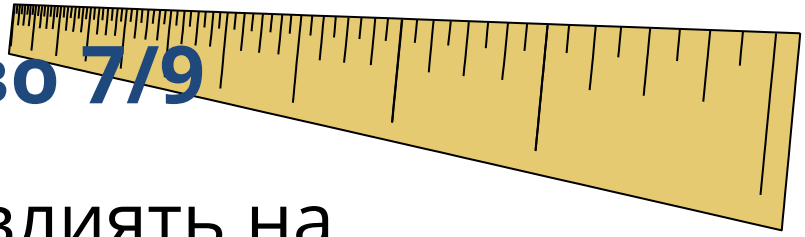
- Усовершенствование процессов
- *Контроль качества*
- Управление изменениями

Тестирование и качество 6/9

Тестирование выполняется для сбора информации.

Поэтому тестирование – это лишь один из информационных сервисов.

Тестирование и качество 7/9



Как тестировщик может повлиять на качество?

Тестирование - это возможный способ оценки качества программного обеспечения в терминах найденных дефектов, исполненных тестов и протестированных систем. Это может быть сделано как для функциональных требований, так и для нефункциональных требований и характеристик программного обеспечения.

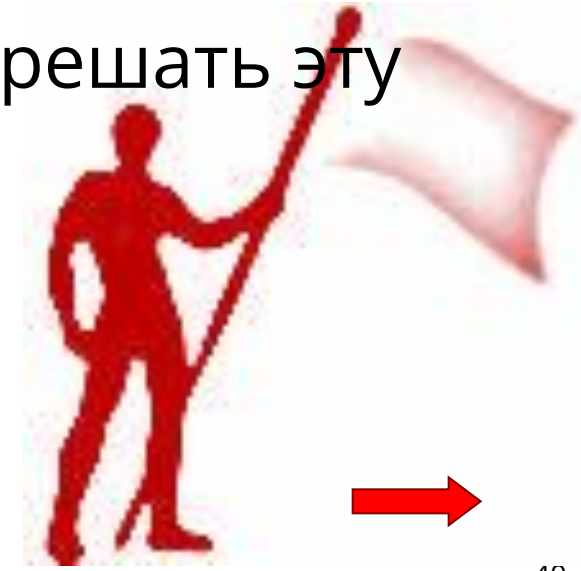
Когда во время тестирования находятся ошибки, качество систем программного обеспечения повышается, если эти дефекты

Тестирование и качество 8/9

Можно думать о себе, как о гаранте качества, но вы не создаете качество и не можете лишить продукт его.

Качество должно закладываться создателями продукта и зачастую для них это становиться неподъемной ношей.

Тестировщик призван помочь им решать эту задачу более эффективно.



Тестирование и качество 9/9

Любой проект похож на езду по дороге. Проекты бывают легкие и типовые, но большинство напоминают заснеженную горную трассу. В этих проектах не обойтись без света фар.

Как тестировщик, *вы освещаете дорогу*.

7 принципов тестирования

Принцип 1 – Тестирование демонстрирует наличие дефектов

Тестирование может показать, что дефекты присутствуют, но не может доказать, что их нет. Тестирование снижает вероятность наличия дефектов, находящихся в программном обеспечении, но, даже если дефекты не были обнаружены, это не доказывает его корректности.

7 принципов тестирования

Принцип 2 – Исчерпывающее тестирование недостижимо

Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев. Вместо исчерпывающего тестирования должны использоваться анализ рисков и расстановка приоритетов, чтобы более точно сфокусировать усилия по тестированию

7 принципов тестирования

Принцип 3 – Раннее тестирование

Чтобы найти дефекты как можно раньше, активности по тестированию должны быть начаты как можно раньше в жизненном цикле разработки программного обеспечения или системы, и должны быть сфокусированы на определенных целях.

ISTQB International Software Testing Qualifications Board

7 принципов тестирования

Принцип 4 – Скопление дефектов

Б***о***льшая часть дефектов, обнаруженных при тестировании или повлекших за собой основное количество сбоев системы, содержится в небольшом количестве модулей.

7 принципов тестирования

Принцип 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 пересматриваться и корректироваться, новые тесты должны быть разносторонними, чтобы охватить все компоненты программного обеспечения, или системы, и найти как можно больше дефектов.

7 принципов тестирования

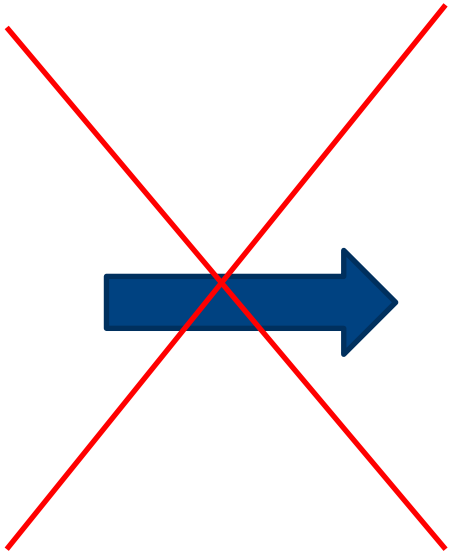
Принцип 6 – Тестирование зависит от контекста

Тестирование выполняется по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем сайт электронной коммерции.

7 принципов тестирования

Принцип 7 – Заблуждение об отсутствии ошибок

Обнаружение и исправление дефектов не помогут, если созданная система не подходит пользователю и не удовлетворяет его ожиданиям и потребностям.



Вот такие ошибки …

- F-16 вверх ногами
  - Испытания американского истребителя F-16 проводились, понятное дело, в северном полушарии. На заключительном этапе самолет решили проверить где-то в Латинской Америке, но уже с другой стороны экватора. При переводе самолета в режим автопилота он автоматически развернулся «вверх ногами».
- Правильно выбирайте типы данных
  - Причиной взрыва 4 июня 1996 г. ракеты Ариан-5, была программная ошибка. В системе управления ракеты использовалось модифицированное программное обеспечение ранее успешно работавшее на Ариан-4, но Ариан-5 ускорялась быстрее предыдущей модификации, в результате когда на 40 секунде полета одна из вспомогательных подпрограмм попыталась преобразовать длинное целое значение в короткое без проверки величины значения, то вышло за границы типа, произошло отключение системы управления ракеты, и она была взорвана по команде на самоликвидацию. Прямой (вместе с ракетой-носителем был потерян коммуникационный спутник) и косвенный ущерб от этого программного сбоя был оценен в полмиллиарда долларов.

Модели жизненного цикла разработки

Программный продукт

Программное обеспечение: Компьютерные программы, алгоритмы и, зачастую, документация и данные, относящиеся к функционированию компьютерной системы.

Проект разработки ПО

Проект: Уникальный набор координируемых и контролируемых задач с датами начала и окончания, предпринимаемый для достижения цели в соответствии с определенными требованиями, включающими в себя ограничения по времени, стоимост
ресурсам

Проект разработки ПО

Разработка программного обеспечения

Программный продукт является результатом проекта по разработке ПО.

Процесс разработки (программного продукта), принятый в проекте, зависит от целей и задач проекта.

Существует огромное количество жизненных циклов разработки, выработанных для достижения различных целей.

Жизненный цикл программного обеспечения

Жизненный цикл (ЖЦ) программного обеспечения : Период времени, начинающийся с момента появления концепции программного обеспечения и заканчивающийся тогда, когда дальнейшее использование программного обеспечения невозможно. Жизненный цикл программного обеспечения обычно включает в себя следующие этапы: концепт, описание требований, дизайн, реализация, тестирование, инсталляция и наладка, эксплуатация и поддержка и, иногда, этап вывода из эксплуатации. Данные фазы могут накладываться друга или проводиться итерационно.

Тестирование – не обособленная процедура. Оно занимает свое место в жизненном цикле, который во многом определяет организацию тестирования.

Модель жизненного цикла разработки

Под моделью обычно понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении жизненного цикла.

- ✔ Этапы:
  - анализ осуществимости; стратегическое планирование; анализ требований;
  - проектирование (предварительное и детальное);
  - кодирование (программирование);
  - отладка и тестирование; интеграция;
  - внедрение; эксплуатация и сопровождение.
- ✔ Результат работ на каждом этапе

ЖЦ ПО: ключевые характеристики 1/2

Эффективность

- Затраты/бюджет
- Сроки
- Приемлемое качество продукта

Прозрачность

- Статус работ известен в любой момент проекта
- Даже если всё заканчивается хорошо, не знать этого до последнего момента - плохо...

ЖЦ ПО: ключевые характеристики 2/2

Предсказуемость

- Реальные трудозатраты и сроки находятся в запланированных (сметных) пределах

Управляемость

- Возможность внесения корректив по ходу проекта (изменяющиеся требования и др.)

Сдерживание рисков

- Устойчивость к влиянию внешних факторов

Модели жизненного цикла разработки

- Каскадная модель
- Итеративная или инкрементальная модель
- Спиральная модель

Процесс разработки ПО зависит от выбранной модели разработки.

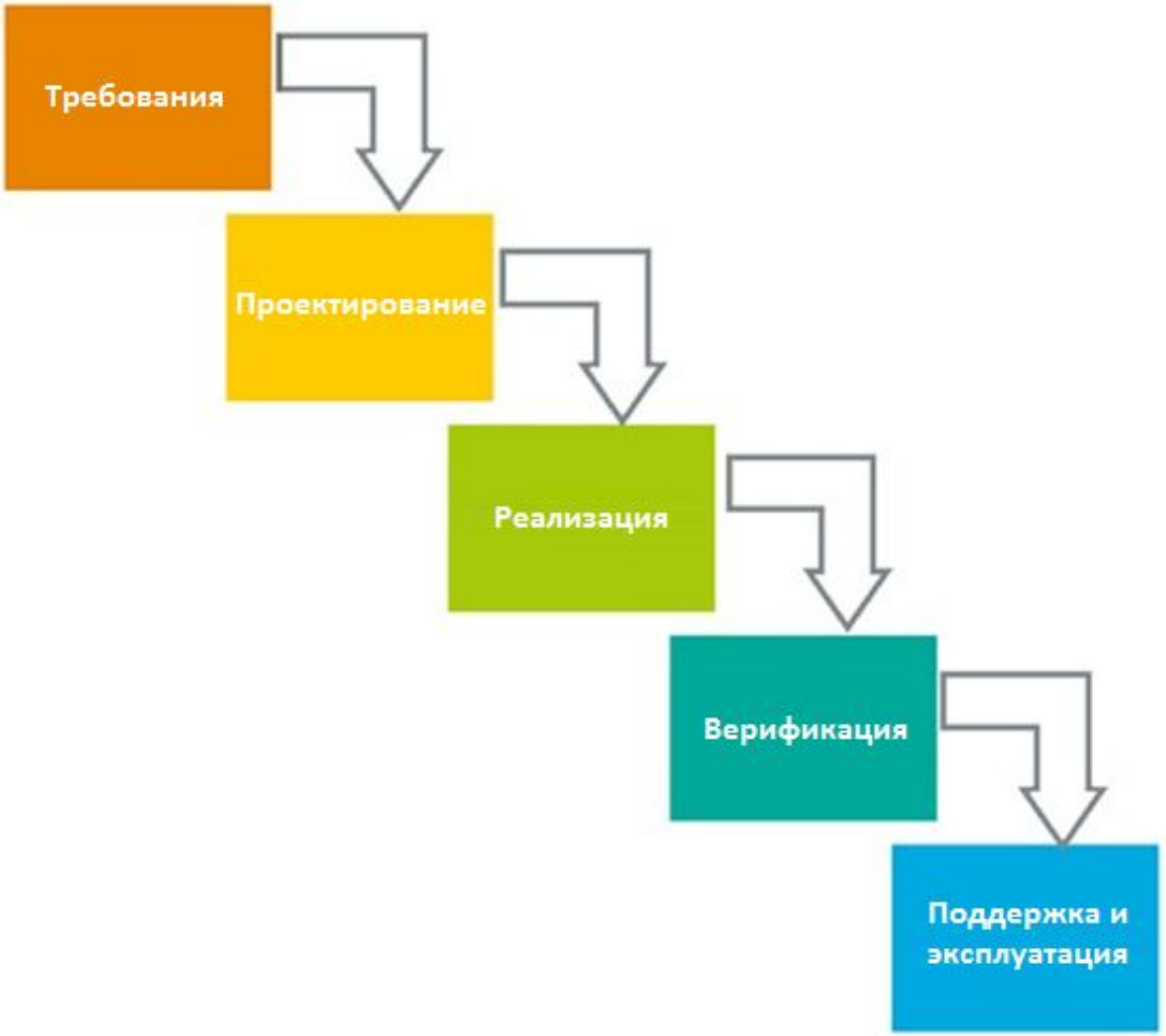
Каскадная модель

Наиболее популярный пример – водопадная модель.

Водопадная модель стала одной из первых разработанных моделей. Она предполагает строгое последовательное (во времени) выполнение всех фаз.

Проекты, в которых за основу взята данная модель, развиваются путем последовательного перехода от фазы к фазе, от первоначального замысла к конечному продукту.

Каскадная (водопадная) модель



Каскадная (водопадная) модель

Ключевые характеристики

- Данная модель требует наличия четко-определенных требований, которые остаются неизменными на протяжении всего проекта.
- Четкое планирование: каждый этап и его составные части планируется и включается в график до начала работ.
- Продукт можно считать завершенным только после окончания последнего этапа.

Каскадная модель

Преимущества

- Четкое документирование: документируется каждая фаза проекта.

Благодаря этому приходящим в проект людям легче включиться в работу

- Простая для понимания и использования
- Приспособлена для разработки ПО высокого архитектурного уровня и сложной структуры

Каскадная модель

Недостатки

- Невозможность вернуться на предыдущую фазу
- Высокий риск конструктивных дефектов
- Непригодна, если заказчик меняет требования.
  - Подходит только для тех проектов, где требования не меняются на протяжении всего цикла разработки
- Нерациональное использование времени: пока проектировщики полностью не закончат работу, разработчики не могут приступить к написанию кода
- Требует много времени и документирования
- Количество тестирования непредсказуемо, велик

Трудности тестирования в каскадной модели 1/3

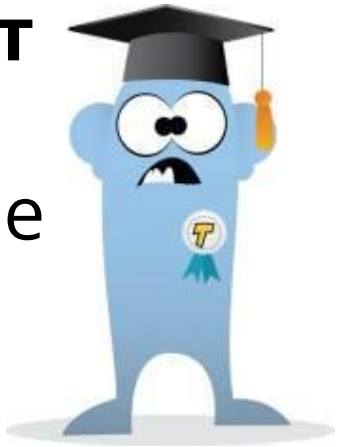
#1 Поиск компромисса между качеством и сроками поставки

Поскольку тестирование начинается только в конце проекта, оно, как правило, выполняется в условиях нехватки времени. В таких ситуациях приходится выбирать между качеством и сроками поставки. Распространенная ситуация: тестировщики под давлением дают добро продукту, многие дефекты которого проявятся во время эксплуатации.

В таких ситуациях тестировщиков могут обвинить в том, что они тормозят процесс

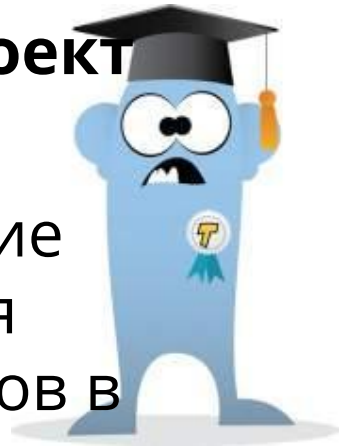
Трудности тестирования в каскадной модели 2/3

#2 Разработчиков также поджимают сроки, в результате чего они передают тестировщикам нестабильные и часто не поддающиеся тестированию системы. Это приводит к тому, что львиная часть графика тестирования уходит на то, что по своей сути, является вынужденным, повторным модульным тестированием.



Трудности тестирования в каскадной модели 3/3

#3 Позднее включение тестировщиков в проект



Когда тестировщики подключаются к работе на стадии формирования требований, тестирование выполняет превентивную функцию, анализируя требования и предотвращая попадание дефектов в код.

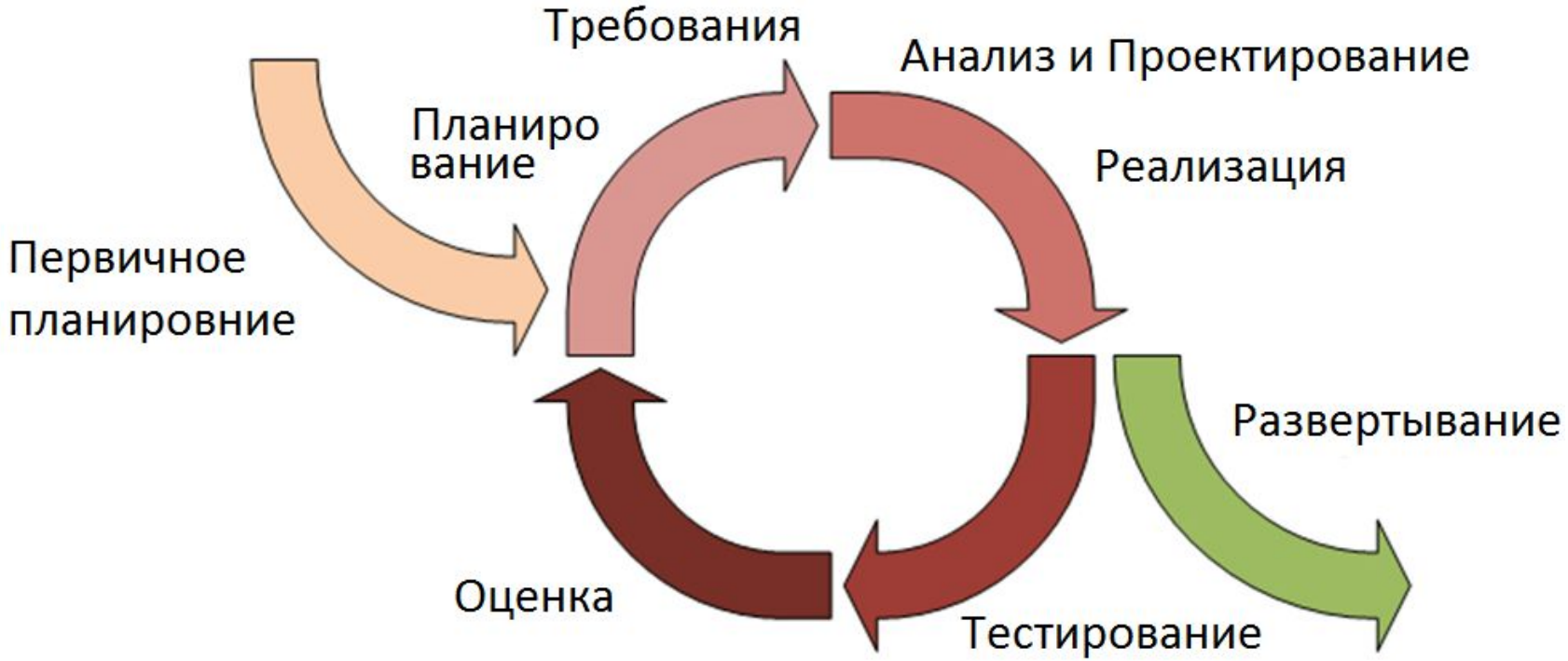
При подключении тестировщиков на более поздних стадиях тестирование выполняется в лучшем случае в рамках реактивной стратегии, в худшем – экспромтом. При этом нет ни предупреждения дефектов, ни четких границ, ни ограниченного объема тестирования.

Итеративная или инкрементальная модель

Итеративные или инкрементальные модели – это модели, в которых система реализуется и тестируется итерационно, блоками

Формирование требований, проектирование, сборка, и тестирование системы делиться на большое количество итераций. Примеры: Rapid Application Development (RAD), Rational Unified Process (RUP) и гибкие методологии разработки (Agile).

Итеративная или инкрементальная модель



Цикл «планирование-действие-проверка-корректировка»

Итеративная или инкрементальная модель

Ключевые характеристики

- Система разрабатывается повторяющимися циклами (итеративная модель). Одновременно разрабатываются лишь небольшие части системы (инкрементальная модель)
- В результате каждой итерации появляется рабочий продукт, являющийся частью конечного разрабатываемого продукта

Итеративная или инкрементальная модель

Преимущества

- Гибкость в принятии новых требований или изменений
- Возможность адаптации процесса на основе уроков, извлеченных из предыдущих итераций
- Более короткие сроки вывода продукта на рынок: возможность получить отзывы от заказчиков/пользователей путем демонстрации рабочих частей системы

Недостатки

- Стоимость продукта неизвестна
- Могут возникнуть проблемы с архитектурой

Трудности тестирования в итеративной или инкрементальной модели

#1 Большие объемы регрессионного тестирования

Каждое расширение системы требует регрессионного тестирования всех функций и возможностей, представленных в предыдущих итерациях.

Поскольку наиболее важные функции и возможности, как правило, выполняются на более ранних итерациях, очень важно проследить, чтобы они не были повреждены.

Трудности тестирования в итеративной или инкрементальной модели

#2 Отсутствие четкого плана для обнаружения и устранения дефектов

Проявляется это в том, что программисты уделяют все рабочее время последующим расширениям в то время, как тестировщики тестируют текущую итерацию.

Как только тестировщики сообщают об обнаруженных дефектах, бизнес-аналитики, проектировщики и программисты, которые должны решать эти проблемы, начинают работать в режиме полной загрузки.

Трудности тестирования в итеративной или инкрементальной модели

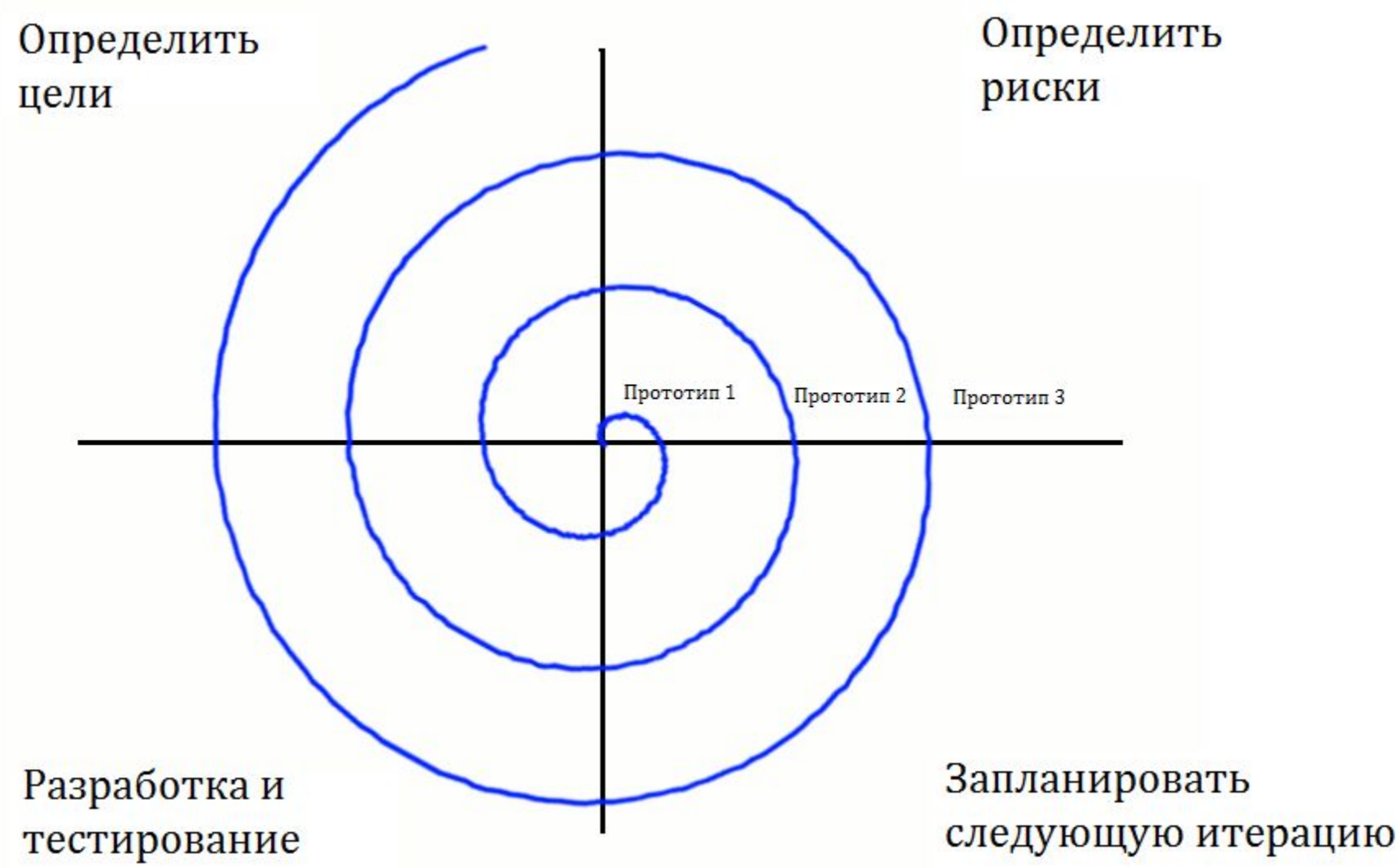
#3 Отсутствие должного внимания и уважения к тестированию

Эта проблема не имеет всеобщего охвата, однако, в сообществе гибкой разработки есть, в некотором смысле пренебрежительное отношение к тестированию и связанными с ним активностями.

Спиральная модель

Система разрабатывается на основе ранних прототипов. Разработка движется от прототипа к прототипу, каждый из которых тестируется, затем перепроектируется и повторно прототипируется, после чего снова тестируется. И так до тех пор, пока все рискованные конструктивные решения не пройдут тестирование (или не пройдут и будут отвергнуты) .

Спиральная модель



Спиральная модель

Ключевые характеристики

Спиральная модель сочетает в себе концепцию итеративной разработки с систематикой и контролем водопадной модели:

- Данная модель включает в себя большую часть этапов водопадной модели, и в том же порядке. Однако этапы отделены друг от друга планированием, оценкой рисков, прототипированием и имитацией.
- На каждой итерации по всему циклу продукт является расширением более раннего продукта (как в итеративной модели)
- Расширение модели осуществляется только после анализа рисков: во время каждого цикла проводиться поиск крупных рисков и делаются попытки по их устранению

Спиральная модель предназначена для крупных, дорогостоящих и сложных проектов (с высокими проектными

Спиральная модель

Преимущества

- Лучший способ разработки систем с большим количеством неизвестных величин
- Одна из наиболее гибких моделей: изменения могут быть внесены позже в жизненном цикле
- Управление рисками – одна из встроенных функций данной модели, что делает ее более привлекательной по сравнению с другими моделями

Недостатки

- Стоимость продукта неизвестна
- Чересчур трудный подход для проектов с четкими техническими требованиями к продукту

Трудности тестирования в спиральной модели

В силу особенностей самой модели, конструкция системы будет неизменно меняться.

Гибкость имеет первостепенное значение для решений, касающихся тестовых сценариев, тестовых данных, инструментов тестирования и тестового окружения на ранних этапах проекта. К примеру, если руководитель тестирования зациклен на каком-то одном способе генерации тестовых данных, а структура системного хранилища данных радикально меняется, скажем, от реляционной базы данных к XML файлам, потребуется значительная переработка тестовых артефактов, инструментов и подходов.

Трудности тестирования в спиральной модели

Специфичная, экспериментальная манера тестирования на ранних этапах проекта.

Тестирование ранних прототипов сводится к поиску неизвестного и тщательному тестированию сомнительных конструкций с целью обнаружить что-то, что не работает.

Повышение уверенности в качестве продукта не является целью, как правило. Такое раннее тестирование, которое переходит к выполнению более типичных функций на финальных этапах, требует от руководителя тестирования менять планы и стратегии по мере развития проекта. Опять же, гибкость здесь – ключевой фактор.

Трудности тестирования в спиральной модели

В силу того, что в спиральной модели мы пытаемся решить проблемы с неизвестными величинами путем повторяющего прототипирования, составление графиков не поддается прогнозированию. Оценка и планирование тестирования весьма затруднительны, особенно при наличии других активных проектов.

Независимо от используемой модели..

Есть несколько характеристик хорошего тестирования:

- каждому этапу разработки соответствует этап тестирования;
- каждый уровень тестирования имеет тестовые цели, характерные для данного уровня;
- анализ и проектирование тестов для данного уровня тестирования должны начинаться во время соответствующего этапа разработки;
- тестировщики должны начинать рецензирование документов, как только их

В жизни …

- В реальной жизни …
- В реальных проектах …
- В реальных ситуациях …

- *Модель в чистом виде используется редко*
- Как правило используется адаптация модели под контурные нужны, что и было задумано, например для RUP, Agile и некоторых других моделей разработки

Команда тестирования

Команда тестирования

Independence of testing

Независимость тестирования

Тип мышления, требуемый для тестирования и рецензирования, отличается от типа мышления, требуемого для разработки. С правильной установкой разработчики сами могут тестировать собственный код, однако ответственность за это передается тестировщику, как правило, для того чтобы сфокусироваться именно на тестировании и получить ряд дополнительных преимуществ, таких как независимый взгляд обученных и профессиональных тестировщиков. Независимое тестирование может быть выполнено на любом уровне тестирования.

Независимость тестирования

Независимость тестирования: Разделение ответственностей, которое позволяет выполнять объективное тестирование.

Уровни независимости

Ниже описываются несколько уровней независимости, в порядке от низкого к высокому:

- Разработчики тестируют собственный код (низкий уровень независимости)
- Независимые тестировщики (например, из команды разработчиков)
- Независимая команда или группа тестирования из другой организационной группы или независимые тестировщики (например, специалисты по тестированию удобства использования и производительности)
- Независимые тестировщики, привлеченные на

Важность независимости тестирования 1/2

Причина 1 – Редактировать и править собственный код – не самая лучшая идея. Свежий взгляд необходим, т.к. проверяя свою работу, вы руководствуетесь теми же предположениями, что и при написании, а, значит, серьезные дефекты останутся незамеченными.

Например, программа может содержать ошибки, обусловленные непониманием программиста поставленной задачи или технического задания. В этом случае, непонимание программиста, скорее всего, отразится и в самой программе.

Важность независимости тестирования 2/2

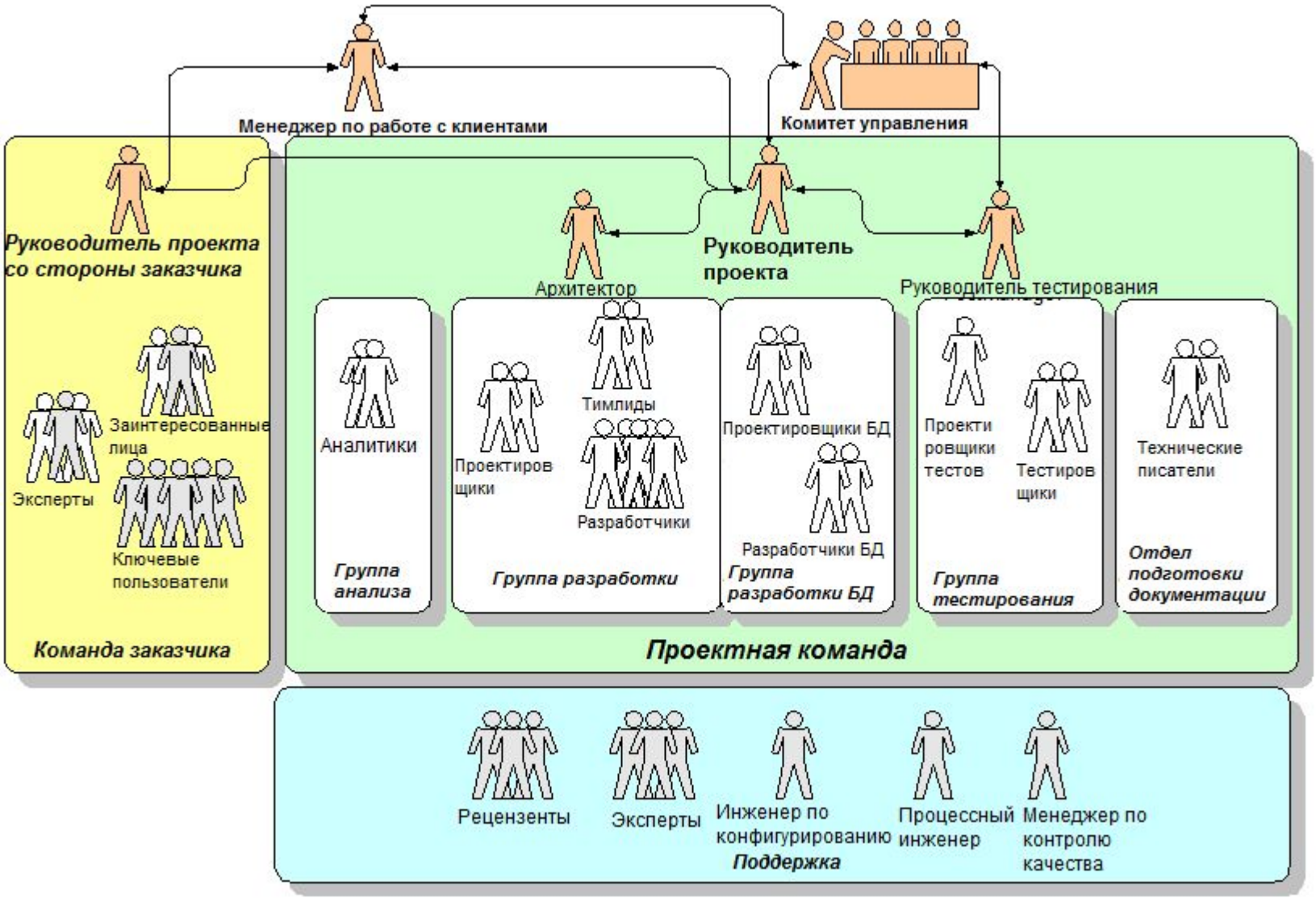
Причина 2 – Никому не нравится находить ошибки в своей работе. Это распространяется и на разработчиков программных продуктов.

Причина 3 – Смена фокусировки в проектной активности так же представляет собой проблему. После *конструктивной работы по проектированию и написанию кода* программисту чрезвычайно сложно переключиться, и вести в отношении собственной же программы *деструктивную деятельность*.

Команда тестирования

Команда

Взаимодействие в проектной команде



Роль тестировщика 1/6

Тестирование выполняет сервисную функцию. Как тестировщик, вы оказываете услуги по тестированию различным «заказчикам»:

Руководитель проекта (PM):

Руководитель проекта обязан быть в курсе деятельности тестировщика и влиять на нее. Тестировщик должен, в свою очередь, по запросу извещать PM’а о статусе тестирования, об обнаруженных серьезных проблемах, и не быть «бутылочным горлышком» для проекта.

Роль тестировщика 2/6

Программист:

Тестировщик облегчает работу программиста, сообщая ему о дефектах в его работе, причем, делая это быстро.

О тестировщика требуется понимание своего ремесла и знание продукта, чтобы не тратить время программиста ошибочными или поверхностными отчетами.

Роль тестировщика 3/6

Технический писатель:

Специалисты, пишущие руководства, получают неполную информацию о продукте. Тестировщик может лучше объяснить им, как работает программа и предостеречь от тех или иных ошибок в документации.

Писатели так же могут помочь группам тестирования. Изучая сам продукт и то, как он эксплуатируется, они могут предупредить тестировщиков о новых областях использования продукта, недочетах в тестовом плане и о дефектах, с которыми сталкиваются пользователи.

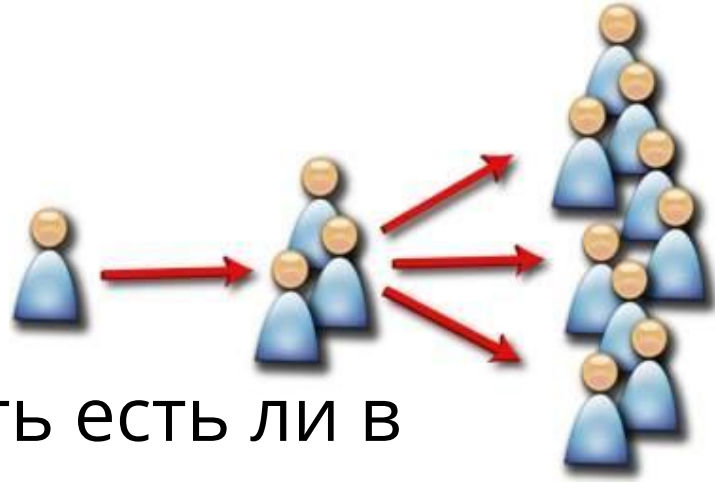
Роль тестировщика 4/6

Техническая поддержка

Тестировщики ставят группы поддержки в известность о тех аспектах продукта, которые могут доставить неудобства пользователям.

Специалисты из службы поддержи так же помогают тестировщикам, поскольку могут обосновать необходимость исправления дефекта.

Роль тестировщика 5/6



Отдел маркетинга:

Отдел маркетинга должен знать есть ли в продукте что-либо несоответствующее его ключевым характеристикам, которые должны быть поставлены заказчику. Дефект, который кажется незначительным разработчикам, может оказаться критически важным для маркетинга.

Также тестировщик может помочь отделу маркетинга в составлении точного отчета о возможностях продукта.

Роль тестировщика 6/6

Пользователь:

В сущности, тестировщик работает на пользователей продукта. Их удовлетворение является приоритетной задачей проекта и, конечно же, тестировщика

Типы и уровни тестирования

Уровень тестирования

Уровень тестирования: группа задач по тестирования которые управляются совместно. Уровень тестирования связан с областями ответственности в проекте.

Примерами уровней тестирования могут служить

- компонентное тестирование,
- интеграционное тестирование,
- системное тестирование
- приёмочное тестирование.

Уровень тестирования

Для каждого уровня тестирования может быть определено:

1. Цель
2. Объекты тестирования
3. Прослеживание связи с базисом тестирования (при наличии)
4. Критерии входа и выхода
5. Артефакты процесса тестирования, которые будет поставлять отдел тестирования - тестовые сценарии, протоколы тестирования, отчетность о результатах и другие
6. Тестовые методики

Уровни тестирования

Процесс тестирования включает в себя следующие уровни:

Уровни тестирования – расширенная структура

Как правило, такое деление тестовых активностей по уровням делается для комплексных систем (системы систем) – системное тестирование на нижем уровне называется подсистемым

Компонентное тестирование

Компонентное тестирование: Тестирование отдельных компонентов программного обеспечения

Так же известно, как модульное тестирование

Компонентное тестирование: общий обзор

- Выполняется самим разработчиком (иногда модульное тестирование доверяется другому разработчику, не автору кода, для повышения уровня независимости)
- Тестирование функциональных и нефункциональных характеристик программы

Компонентное тестирование

- Пример кода

Calculator.java | CalculatorSumTest.java

```
1   package foo.bar;
2
3   public class Calculator {
4
5       public static int Sum(int a, int b){
6
7           return a + b;
8       }
9
10  }
11
```

Компонентное тестирование

- Модульный тест

Calculator.java | CalculatorSumTest.java

```
1  import static org.junit.Assert.*;
2
3  import org.junit.Test;
4
5  import foo.bar.Calculator;
6
7
8  public class CalculatorSumTest {
9
10     @Test
11     public void testSum() {
12         assertEquals(4, Calculator.goodSum(2, 2));
13     }
14 }
15
```

- Далее эту программу запускаем, таким образом автоматически тестирую код.

Компонентное тестирование

- Цель
  - Изолировать отдельные части программы и показать, что по отдельности все части работают.
- Объекты тестирования
  - Компоненты
  - Программы
  - Модули БД
- Базис тестирования
  - Требования к компонентам
  - Детальный дизайн

Компонентное тестирование

- Критерии входа
  - Бизнес- и функциональные требования выработаны и утверждены.
  - Разработка компонентов закончена.
  - Среда разработки стабильна.
  - Документация по тестовым сценариям модульных тестов составлена.
- Критерии выхода
  - Все тестовые сценарии модульных тестов исполнены. Тестовые результаты доступны.
  - Обнаруженные дефекты исправлены и закрыты.
  - Проверка кода завершена.

Компонентное тестирование

- Отчетность
  - Как правило, дефекты устраняются по мере обнаружения, без составления отчетов.
- Тестовые методики
  - Тестирование операторов
  - Тестирование ветвей
  - Тестирование условий
  - Тестирование путей
- Метрики и измерения
  - Покрытие операторов

Компонентное тестирование

- Инструментарий
  - Инструмент отладки
  - Интегрированная среда модульного тестирования
    - Junit
    - Nunit
    - Другие

Компонентное тестирование

Преимущества

- Возможность протестировать часть программы, не ожидая готовности остальных частей
- Раннее обнаружение дефектов
- Программисты обнаруживают и мгновенно исправляют проблемы. Упрощенная отладка
- Лучшее структурное покрытие кода
- Модульное тестирование экономичнее других этапов тестирования
- Упрощенная интеграция

Компонентное тестирование

Недостатки

- Время от времени требуется реализовывать *заглушки* и *драйвера*
- Модульное тестирование основано, в первую очередь, на написанном коде. Поэтому, если что-то было пропущено, модульное тестирование этого не покажет

Тестирование интеграции компонентов

Тестирование интеграции компонентов: Тестирование, выполняемое для выявления дефектов в интерфейсах и взаимодействии между интегрированными компонентами.

Тестирование интеграции компонентов: общий обзор

- Как правило, следует за компонентным тестированием
- Выполняется разработчиками или тестировщиками, специализирующихся на интеграционном тестировании (редкая квалификация)
- Тестирование функциональных и нефункциональных характеристик программы

Тестирование интеграции компонентов

- Цель
  - Удостовериться, что взаимодействие двух или более компонентов дает результат, отвечающий функциональным требованиям
  - Обнаружить проблемы интерфейса
- Объекты тестирования
  - Подсистемы
  - Инфраструктура
  - Интерфейсы

Тестирование интеграции компонентов

- Базис тестирования
  - Проект программы и системы
  - Архитектура
  - Технологический процесс (workflow)
  - Сценарии использования
- Критерии входа
  - Модули для интеграционного тестирования закончены
  - Компонентное тестирование закончено
  - Проблемы, обнаруженные в компонентном тестировании, исправлены и закрыты

Тестирование интеграции компонентов

- Критерии выхода
  - Дефекты, обнаруженные во время интеграционного тестирования, исправлены и закрыты
  - Все тестовые сценарии исполнены; для каждого сценария есть результаты тестирования
- Методы интеграционного тестирования
  - «Большой взрыв» (“Big Bang)”
  - «Сверху вниз» (“Top down”)
  - «Снизу вверх» (“Bottom up”)
  - *Методы подробно разбираются в тренинге «SQA-028 Основы*

Тестирование интеграции компонентов

Преимущества

- Большая стабильность по сравнению с тестированием графического пользовательского интерфейса
- Положительно влияет на внутренний дизайн программы
- Ранняя и более легкая локализация дефектов интерфейса на стадии системного тестирования

Недостатки

Тестировщик должен читать код, а временами и писать его

Системное тестирование

Системное тестирование: Процесс тестирования системы в целом с целью проверки того, что она соответствует установленным требованиям

Системное тестирование: общий обзор

- Выполняется тестировщиками
- Тестирование функциональных и нефункциональных характеристик программы
- Системное тестирование является разновидностью тестирования методом черного ящика, а, следовательно, не требует знания внутренней структуры кода или логики
- Включает тестирование взаимодействия с операционной системой и системными

Системное тестирование

- Цель
  - Протестировать законченную, интегрированную систему, чтобы оценить ее соответствие указанным требованиям (функциональным/нефункциональным)
- Объекты тестирования
  - Система в целом
- Базис тестирования
  - Функциональная спецификация (FRS)
  - Спецификация системных требований к ПО (SRS)
  - Сценарии использования

Системное тестирование

- Критерии входа
  - Модульное и интеграционное тестирование всех модулей закончено.
  - Окружение для системного тестирования готово.
  - Спецификации продукта закончены и утверждены.
  - Сценарии системного тестирования отражены в документах.
  - Пользовательский интерфейс и тестируемый функционал заморожены.
- Критерии выхода
  - Программа отвечает всем требованиям и обладает требуемым функционалом.
  - Дефекты, обнаруженные во время системного тестирования исправлены и закрыты.

Приемочное тестирование

Приемочное тестирование - формальное

испытание системы, проводимое с целью определения соответствия реализованных требований, бизнес процессов, потребностей пользователя приемочным критериям. На основании результатов приемочного тестирования пользователь, заказчик или другое уполномоченное лицо принимает решение о приемке системы в эксплуатацию

Приемочное тестирование: общий обзор

- Выполняется заказчиком или пользователем системы
- Поиск дефектов не является главной целью
- Пользовательское приемочное тестирование проверяет готовность системы для использования; Эксплуатационное тестирование проверяет насколько система пригода для эксплуатации в конкретном операционном окружении
- Альфа тестирования выполняется в стенах компании, которая разрабатывает программный продукт. Бета

Классификация тестирования

- С исполнением и без исполнения кода:

  статическое / динамическое

- Различные знания о структуре кода:

  черный ящик / серый ящик / белый ящик

- По свойствам тестируемого объекта:
  функциональность, производительность, совместимость, надежность, удобство...

- По изменениям:

  регрессионное тестирование, подтверждающее тестирование

- По типу прогона тестов:

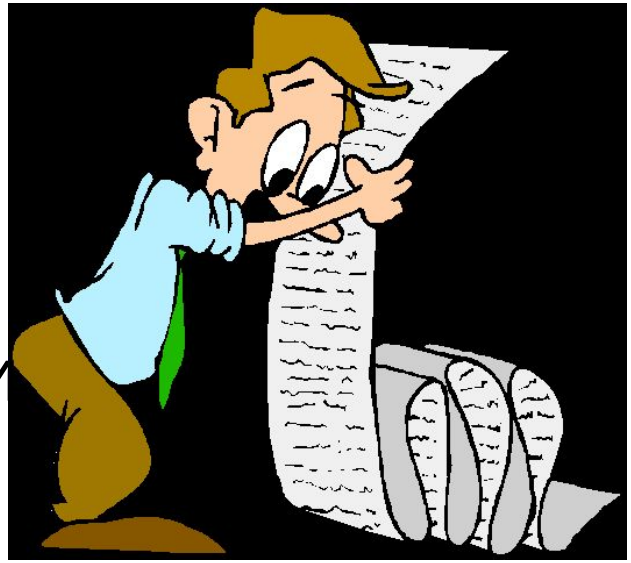
С исполнением и без исполнения кода…

Статическое тестирование: Тестирование компонента или системы на уровне спецификации или реализации без исполнения кода программного продукта, например, рецензирование или статический анализ.

Динамическое тестирование: Тестирование, проводимое во время выполнения программного обеспечения, компонента или системы.

Статическое тестирование

При статическом тестировании код исследуется вручную (рецензирование) или с помощью автоматизированных средств анализа (статистический анализ) без исполнения кода.



Объекты тестирования:

- Код
- Документация с требованиями
- Сценарии использования
- Руководства

Статическое тестирование

Преимущества рецензирования:

- раннее обнаружение и исправление дефектов
- улучшение продуктивности разработки
- уменьшение времени разработки
- уменьшение времени и стоимости тестирования
- сокращение стоимости жизненного цикла
- уменьшение числа дефектов и улучшение коммуникаций

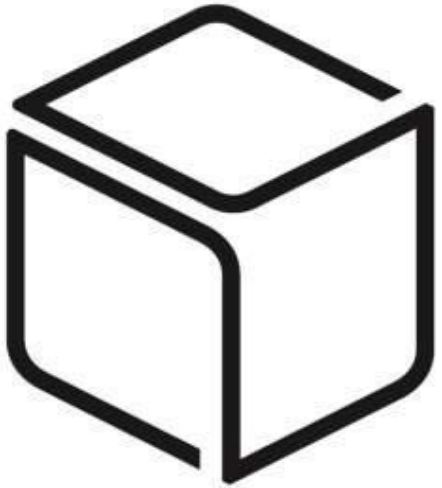
Динамическое тестирование

Цель статического тестирования – поиск *дефектов* в продукте, в то время как цель динамического тестирования, обнаружение *отказов системы*. Только динамическое тестирование дает представление о поведении программы и позволяет выявить различия между ожидаемым и фактическим поведением.

Объекты тестирования:

- Модуль
- Интерфейс
- Система

Различные знания о структуре кода...



Тестирование методом черного ящика

Тестирование методом черного ящика: Тестирование, функциональное или нефункциональное, без знания внутренней структуры компонента или системы.

Тестирование методом серого ящика

Тестирование методом серого ящика: Сочетает в себе тестирование методом черного и белого ящика.

- Например, продукт тестируется методом черного ящика, но тестовые сценарии разрабатываются с разрабатываются с учетом знаний о внутренней структуре продукта.

Тестирование методом белого ящика

Тестирование методом белого ящика:
Тестирование, основанное на анализе внутренней структуры компонента или системы.

Синонимы:

- *тестирование на основе структуры*
- *структурное тестирование*
- *тестирование прозрачного ящика*
- *тестирование методом прозрачного ящ*

По свойствам тестируемого объекта…

- Функциональное тестирование
  - Тестирование установки (инсталляции)
  - Тестирование графического пользовательского интерфейса
- Нефункциональное тестирование
  - Тестирование производительности, нагрузочное тестирование, стрессовое тестирование
  - Тестирование обеспеченности технической поддержкой
  - Тестирование локализуемости
  - Тестирование практичности
  - Тестирование защищенности
  - …

Функциональное тестирование

Функциональное тестирование:
Тестирование, основанное на анализе спецификации функциональности компонента или системы.

Функциональное тестирование включает в себя:

- Тестирование настройки и лицензирования
- Тестирование графического пользовательского интерфейса
- ...

Тестирование установки и лицензирования

Тестирование установки: Вид тестирования, ориентированный на то, что требуется пользователям для успешной установки, настройки и регистрации программного продукта. Процесс тестирования может включать полный и частичный процесс установки/удаления, а так же обновления программы

Тестирование целостности данных

Тестирование целостности данных: Вид тестирования, главной целью которого является проверка целостности данных после различных транзакций (ввод/выбор/обновление). Как правило, целостность данных проверяется в тестированиях методом белого и черного ящика

Тестирование защищенности

Тестирование защищенности: Тестирование с целью оценить защищенность программного продукта

Объекты тестирования:

- пароли
- шифрование
- аппаратные устройства доступа
- уровни доступа к информации
- авторизация
- скрытые каналы

Тестирование графического пользовательского интерфейса

Тестирование графического пользовательского интерфейса: Тестирование графического пользовательского интерфейса продукта с целью удостовериться в том, что он отвечает спецификациям

Нефункциональное тестирование

Нефункциональное тестирование:
Тестирование атрибутов компонента или системы, не относящихся к функциональности, то есть надежность, эффективность, практичность, сопровождаемость и переносимость.

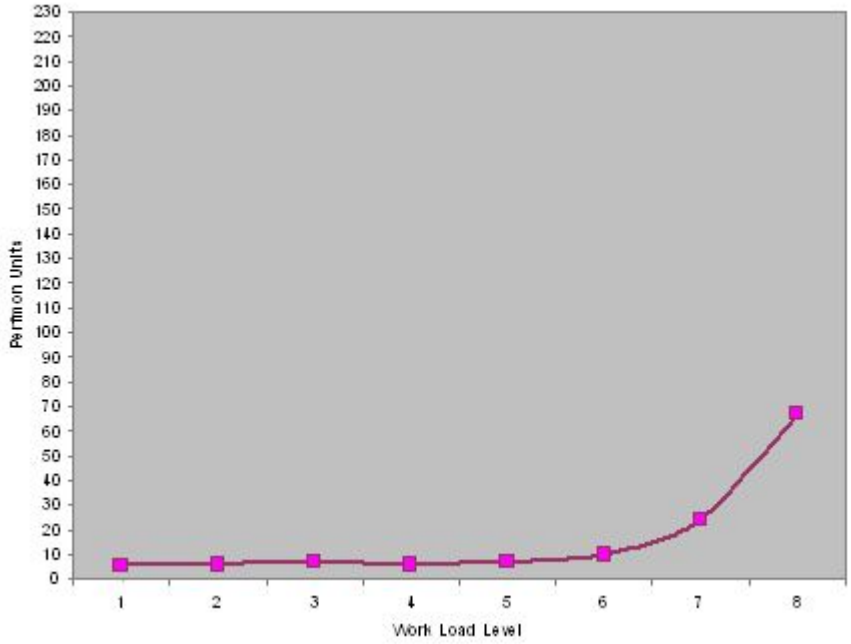
Тестирование производительности

Тестирование производительности: Процесс тестирования с целью определить производительность программного продукта

Тестирование производительности: в инженер программного обеспечения тестирование, которо проводится с целью определения, как быстро работает система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов. http://ru.wikipedia.org/

Нагрузочное тестирование

Нагрузочное тестирование: Тип тестирования производительности, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или операций, а также определения какую нагрузку может выдержать компонент или система.



Стрессовое тестирование

Стрессовое тестирование: Вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу

Тестирование удобства использования

Тестирование удобства использования: Тестирование с целью определения степени понятности, легкости в изучении и использовании, привлекательности программного продукта для пользователя при условии использования в заданных условиях эксплуатации.

Тестирование по изменениям…

Подтверждающее тестирование: Тестирование, во время которого исполняются тестовые сценарии, выявившие ошибки во время последнего запуска, для подтверждения успешности исправления этих ошибок.

Регрессионное тестирование: Тестирование уже протестированной программы, проводящееся после модификации для уверенности в том, что процесс модификации не внес или не активизировал ошибки в областях, не подвергавшихся изменениям. Проводится после изменений в коде программного продукта или его окружении

По типу прогона тестов..

Ручное тестирование: Процесс ручного тестирования продукта. Тестировщик играет роль конечного пользователя, используя максимальное количество функций программы, чтобы удостовериться в их корректной работе.

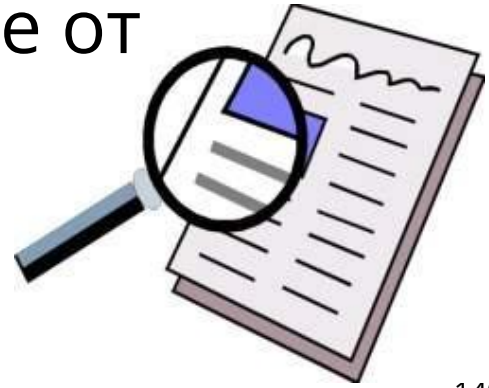
Автоматизированное тестирование: Использование программного обеспечения (помимо тестируемого ПО) для контроля выполнения тестов, сравнения полученных результатов с эталонными, установки предусловий тестов и других функций контроля тестирования и организации отчетов.

Дефекты

Дефекты – основная продукция тестировщиков

Отчет о дефекте – основной продукт работы большинства тестировщиков.

Хорошими отчетами тестировщик зарабатывает себе хорошую репутацию. Плохие отчеты, написанные в критикующей манере и недостаточно обоснованные, создают дополнительную работу для программистов, тратят их время и формируют негативное впечатление от работы тестировщика.



Отчет о дефекте

Отчет о дефекте: Документ, содержащий отчет о любом недостатке в компоненте или системе, который может привести компонент или систему к невозможности выполнить требуемую функцию.

Инструмент управления дефектами

Система управления дефектами: Инструмент, обеспечивающий фиксирование дефектов и изменений, а также поддержку их состояний. Часто имеет процессно-ориентированные возможности для поддержки и контроля распределения, исправления и повторной проверки дефектов, а также возможности отчетности

Тренинг "SQA-024 Основы управление дефектами"

Жизненный цикл отчета об ошибке

Термин «ЖЦ отчета об ошибке» относится к различным стадиям, которые дефект проходит в инструменте управления дефектами за время своего жизненного цикла.

В большинстве проектных команд установлены правила о том, кто может менять статус дефекта или назначать его кому-либо. Например, только руководитель проекта может принять решение отсрочить исправление дефекта или же только тестировщик имеет разрешение на закрытие

Пример ЖЦ дефекта 1/3

Пример ЖЦ дефекта 2/3

После сообщения о дефекте, отчет изучается коллегой тестировщика, сообщившего о нем, или руководителем тестирования.

Если при рецензировании дефект подтвердиться, *открывается* отчет о дефекте, и проектная команда должна принять решение, исправлять дефект или нет. В случае, если дефект подлежит исправлению, *назначается* программист для решения задачи.

Как только программист исправит дефект, отчет о нем возвращается тестировщику на подтверждающее тестирование. Если исправления не будут подтверждены результатами тестирования дефект будет *переоткрыт* и *переназначен*.

Пример ЖЦ дефекта 3/3

После подтверждения тестировщика об устранении дефекта, отчет о нем *закрывается*. Работы по нему заканчиваются.

Любой статус помимо «*отклонен*», «*отложен*» или «*закрыт*» требует работ по устранению дефекта до окончания проекта. У отчета о дефекте в таком статусе есть владелец, ответственный за переход инцидента в следующий статус.

Стрелками на схеме показаны допустимые направления таких переходов.

Отчет о дефекте 1/6

Идентификатор. Уникальный ID, присваиваемый отчету о дефекте, который может быть использован при его поиске и упоминании о нем.

Краткое описание. Краткое описание дефекта.

Подробное описание. Детальное описание дефекта.

Влияние. Критичность и серьйозность дефекта.

IEEE 829 Standard for Software Test Documentation

Отчет о дефекте 2/6

Краткое описание – очень важное поле, на которое будут обращать внимание руководитель проекта и прочие руководители, и исполнители, изучая список дефектов, которые не были или не будут устранены. Должно включать в себя:

- краткое, но конкретное описание, которое даст читателю представление о характере проблемы
- краткое описание границ и зависимостей дефекта (насколько узки или широки обстоятельства, обуславливающие данный дефект?)
- краткое описание влияния или последствий данного дефекта

Отчет о дефекте 3/6

Описание инцидента может содержать следующую информацию:

- Время и дата
- Имя тестировщика
- Аппаратные и программные конфигурации
- Входные данные
- Шаги процедуры
- Ожидаемые результаты
- Фактические результаты
- Попытки воспроизвести дефект, описание испытанных средств

Отчет о дефекте 4/6

Влияние касается приоритета и критичности дефекта

- **Критичность.** Важность воздействия конкретного дефекта на систему.
- **Приоритет.** В какой мере дефект в данном месте системы влияет на ценность продукта в глазах заказчиков и пользователей,

Отчет о дефекте 5/6

Уровни критичности:

1. Полный отказ системы, потеря данных, повреждение данных, бреши в защите
2. Операционная ошибка, неверный результат, потеря функциональности
3. Небольшие проблемы, орфографические ошибки, разметка пользовательского интерфейса, редкие случаи
4. Предложения по улучшению

Обычно критичность не меняется до тех пор, пока не вскрываются скрытые последствия

Отчет о дефекте 6/6

Приоритеты:

1. Требует немедленного устранения, делает невозможным дальнейшее тестирование, явный дефект
2. Должен быть устранен до релиза
3. Устранить, когда будет время
4. Желательно устранить, но не препятствует релизу продукта

По мере развития проекта приоритеты могут меняться

Классификация дефектов 1/6

- **Комментарии:** Несоответствующие/некорректные/дезориентирующие или пропущенные комментарии в исходном коде
- **Ошибка в вычислениях:** Неправильный расчет по формуле/ неправильная бизнес валидация в коде
- **Ошибка данных:** Некорректная совокупность данных/обновление БД
- **Ошибка базы данных:** Ошибка в схеме/структуре БД
- **Упущения при проектировании:** Проектные данные/методы проектирования упущены/не отражены в документации и не отвечают

Классификация дефектов 2/6

- **Некорректное или условно-оптимальное проектирование:** Проектные данные/методы проектирования требуют корректировки для того, чтобы считаться полными. Описанные конструктивные особенности не являются оптимальными для требуемого решения
- **Неправильное проектирование:** Неправильное или неточное проектирование
- **Нечеткое проектирование:** Проектные данные/методы проектирования не ясны для рецензента. Слова допускают двоякое толкование, проектные данные нечетки

Классификация дефектов 3/6

- **Ошибка интерфейса:** Внутренние или внешние по отношению к приложению ошибки интерфейса, некорректная обработка переданных параметров, неправильное расположение полей и объектов, неудобное положение окна/ экрана
- **Логическая ошибка:** Отсутствующая, недостаточная, неактуальная или неоднозначная функциональность в исходном коде
- **Ошибки в сообщениях:** Несоответствующие/некорректные/ошибочные или отсутствующие сообщения об ошибках в исходном коде
- **Ошибка навигации между объектами:** Навигация

Классификация дефектов 4/6

- **Ошибка производительности:** Ошибка, связанная с производительностью/ оптимальностью кода
- **Пропущенные требования:** Неявные/явные требования были пропущены/не отражены в документах на стадии сбора требований
- **Неполноценные требования:** Требования нуждаются в расширении для того, чтобы быть полными
- **Некорректные требования:** Ошибочные или неточные требования

Классификация требований 5/6

- **Нечеткие требования:** Требования не ясны рецензенту. Используются слова, допускающие двоякое толкование (например, вроде, возможно, может быть и т.д.)
- **Ошибка настроек времени/очередности:** Ошибка, вызванная неверными/отсутствующими расчетами времени ожидания и очередности
- **Стандарты:** Не соблюдаются стандарты, например, стандарты по проектированию/сбору требований/кодированию, имеющие

Классификация требований 6/6

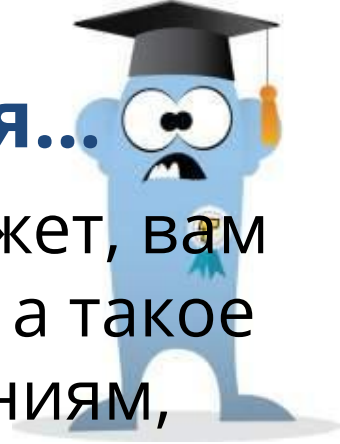
- **Системная ошибка:** Аппаратные ошибки и ошибки ОС, потеря доступа к памяти, системный сбой
- **Ошибка тестового плана/сценария:** Неполноценные/неверные/нечеткие/дублирующие или пропущенные тестовые планы/сценарии, неполная/неверная конфигурация тестов
- **Типографическая ошибка:** Орфографические/грамматические ошибки в документации/исходном коде
- **Ошибка объявления переменных:** Неверная

Далеко не все дефекты устраняются...

В реальности далеко не все найденные дефекты устраняются. Некоторые не устраняются вовсе, исправление же некоторых откладывается до следующего релиза.

Причина 1 – Нехватка времени. В любом проекте приходится иметь дело с огромным количеством программных свойств. Людей, которые эти свойства могли бы записать в код и протестировать, как правило не хватает, не говоря уже о времени для завершения работ.

Далеко не все дефекты устраняются...



Причина 2 – Дефект вовсе не дефект. Может, вам доводилось слышать фразу: «Это не дефект, а такое свойство!». Это часто приводит к непониманиям, ошибкам в тестах или изменениям в спецификации, поскольку потенциальные дефекты рассматриваются как свойства системы.

Причина 3 – Устранять неисправность слишком рискованно. К сожалению, это случается очень часто. ПО – хрупкая и взаимосвязанная система. Устранение одной неисправности может привести к появлению других. Менять что-либо в программе незадолго до выпуска релиза может быть очень рискованной затеей. Лучше оставить в программе известный дефект, чтобы избежать риска появления новых, неизвестных

Далеко не все дефекты устраняются...

Причина 4 – Это того не стоит. Дефекты, которые будут проявляться очень редко или в малоиспользуемых функциях, могут быть оставлены без изменений. Дефекты, которые можно обойти или предотвратить, так же часто не устраняются. Все сводится к оценке рисков.

Причина 5 – Неэффективный отчет о дефектах. Тестировщик недостаточно обосновал необходимость устранения определенного дефекта. В результате, дефект не был воспринят как таковой, был воспринят как не препятствующий выпуску продукта, как слишком рискованный для устранения или просто как не стоящий усилий по устранению.

Упражнение

Представьте, что вы тестируете калькулятор Windows, который выдает: 1+1=2, 2+2=5, 3+3=6, 4+4=9, 5+5=10 и 6+6=13. Напишите отчет о дефекте, который бы эффективно описывал проблему.

- 5 мин на размышления
- 10 мин на разбор

Портрет тестировщика ПО

Личные навыки

Навыки тестирования ПО приобретаются через опыт или обучение в различных направлениях деятельности. Все нижеперечисленное может внести свой вклад в базу знаний тестировщика:

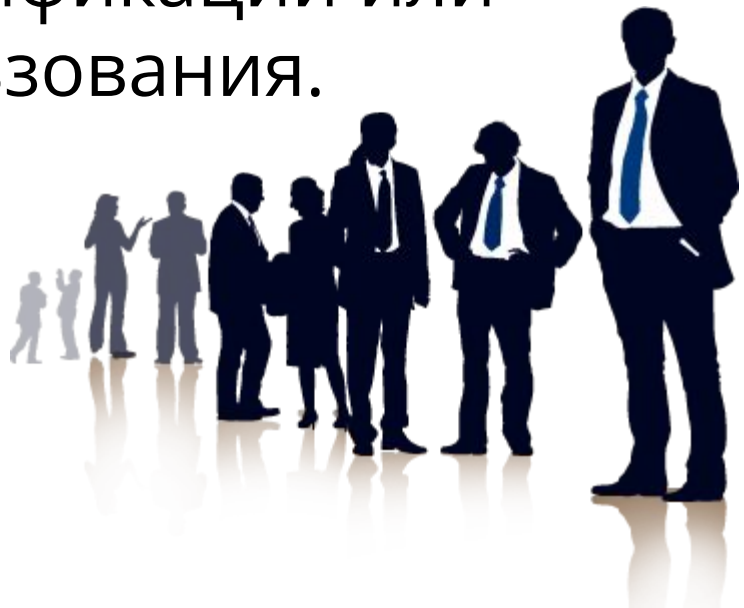
- Использование программных систем
- Знание предметной области или бизнеса
- Участие в различных этапах разработки ПО, включая анализ, разработку и техническую поддержку
- Участие в тестировании ПО

Использование программных систем

Пользователи программных систем хорошо знакомы с системой и точно знают, *как она работает, какие именно неисправности имеют существенное влияние, и какова должна быть ожидаемая реакция системы .*

Знание проблемной области или бизнеса

Пользователи с экспертизой в проблемной области знают области *наибольшей важности* для бизнеса и *как они влияют на способность бизнеса решать проблемы.* Эти знания могут быть использованы для приоритизации тестовых активностей, создании реалистичных тестовых данных и сценариев, и верификации или поставки сценариев использования.



Участие в различных этапах разработки ПО

Знание процесса разработки ПО (анализ требований, проектирование и программирование) дает понимание того, *как появляются ошибки, где они могут быть обнаружены и как предотвратить их появление.* Опыт технической поддержки дает знания о *пользовательском опыте, ожиданиях и требованиях к практичности.* Опыт в разработке ПО необходим для работы с профессиональными инструментами автоматизации тестирования, которые требуют опыта в программировании и проектировании.

Участие в тестировании ПО

К специфичным навыкам тестирования ПО относится умение анализировать спецификации, участвовать в анализе рисков, разрабатывать тестовые сценарии, а так же внимательно прогонять тесты и записывать результаты.

Навыки межличностного общения

Навыки межличностного общения такие, как *умение критиковать и воспринимать критику, оказывать влияние на других и умение вести переговоры* так же важны для тестировщика. Технически грамотный тестировщик, скорее всего, не справиться со своей задачей, если не овладеет и не будет использовать необходимые навыки межличностного общения.

Успешного специалиста в области тестирования так же отличает *организованность, внимание к деталям и сильные навыки письменной и устной коммуникации.*

Литература

Дополнительная литература

- “Lessons Learned in Software Testing” by Cem Kaner, James Bach and Bret Pettichord
- “Foundations of Software Testing” by Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black
- “Software testing” by Ron Patton
- “Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах” Роман Савин
- “How to Break Software: A Practical Guide to Testing” James A. Whittaker