**Книги, литература**

Савин - Тестирование DOT COM

Foundations of Software Testing: ISTQB Certification – гуру книг о тестинге

А вообще любая литература на этот счёт будет полезна.

Также есть сайты на которых публикуются посты статьи интересные мнения - [http://software-testing.ru](http://software-testing.ru/)

<http://www.protesting.ru/> и т.д

Сначала попробуем понять, чем тестирование НЕ является.

**Тестирование не разработка**,

даже если тестировщики умеют программировать, в том числе и тесты (автоматизация тестирование = программирование), могут разрабатывать какие-то вспомогательные программы (для себя).

Тем не менее, тестирование — это не деятельность по разработке программного обеспечения.

**Тестирование не анализ**,

и не деятельность по сбору и анализу требований.

Хотя, в процессе тестирования иногда приходится уточнять требования, а иногда приходится их анализировать. Но эта деятельность не основная, скорее, это приходится делать просто по необходимости.

**Тестирование не управление**,

несмотря на то, что во многих организациях есть такая роль, как «тест-менеджер». Конечно же, тестировщиками надо управлять. Но само по себе тестирование управлением не является.

**Тестирование не техписательство**,

однако тестировщикам приходится документировать свои тесты и свою работу.

Тестирование нельзя считать ни одной из этих деятельностей просто потому, что в процессе разработки (или анализа требований, или написания документации для своих тестов) всю эту работу тестировщики делают для себя, а не для кого-то другого.

Деятельность значима только тогда, когда она востребована, то есть тестировщики должны что-то производить «на экспорт». Что они делают «на экспорт»?

Можно считать, что отчуждаемыми результатами работы тестировщиков являются дефекты, описания дефектов, или отчеты о тестировании.

**Главная деятельность тестировщиков**

В разработке программных систем положительной обратной связью, конечно же, является какая-то информация, которую мы получаем от конечных пользователей. Это запросы на какую-то новую функциональность, это увеличение объема продаж (если мы выпускаем качественный продукт).

Отрицательная обратная связь тоже может поступать от конечных пользователей в виде каких-то негативных отзывов. Либо она может поступать от тестировщиков.

Чем раньше предоставляется отрицательная обратная связь, тем более слабый сигнал ей еще нужно модифицировать, и поэтому тем меньше энергии необходимо для модификации этого сигнала. Именно поэтому тестировать нужно начинать как можно раньше, на самых ранних стадиях проекта, и предоставлять эту обратную связь и на этапе проектирования, и еще, может быть, раньше, еще на этапе сбора и анализа требований.

**Тестирование — это**

Тестирование – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на ограниченном наборе тестов, выбранном определенным образом.

проверка соответствия программы требованиям,

осуществляемая путем наблюдения за ее работой

в специальных, искусственно созданных ситуациях, выбранных определенным образом.

**Общая схема тестирования примерно следующая:**

1. Тестировщик на входе получает программу и/или требования.
2. Он с ними что-то делает, наблюдает за работой программы в определенных, искуственно созданных им ситуациях.
3. На выходе он получает информацию о соответствиях и несоответствиях.
4. Далее эта информация используется для того, чтобы улучшить уже существующую программу. Либо для того, чтобы изменить требования к еще только разрабатываемой программе.

**Что такое тест**

Это специальная, искусственно созданная ситуация, выбранная определенным образом,

и описание того, какие наблюдения за работой программы нужно сделать

для проверки ее соответствия некоторому требованию.

**Ну и таким образом мы можем заключить, что тестировщик делает в процессе тестирования две вещи.**

1.Во-первых, он управляет выполнением программы и создает эти самые искусственные ситуации, в которых мы собираемся проверять поведение программы.

2.И, во-вторых, он наблюдает за поведением программы и сравнивает то, что он видит с тем, что ожидается.

**Основные задачи тестирования**

Еще несколько терминов, которые связаны с упомянутыми двумя задачами, которыми занимается тестировщик, это стимулы, реакции и оракул.

Стимулы – это данные, которые подаются на вход программе.

Реакции — это то, что получается на выходе.

Оракул — это способ проверки наблюдаемого результата, совпадает он с некоторыми ожиданиями или не совпадает.

Программа представляет собой механизм по переработке информации. На вход поступает информация в каком-то одном виде, на выходе информация в некотором другом виде. При этом входов и выходов у программы может быть много, они могут быть разными, то есть у программы может быть несколько разных интерфейсов, и эти интерф ейсы могут иметь разные  виды

* Пользовательский интерфейс (UI)
* Программный интерфейс (API)
* Сетевой протокол
* Файловая система
* Состояние окружения
* События

Наиболее распространенные интерфейсы это

* графический,
* текстовый,
* консольный,
* и речевой.

Через **пользовательский интерфейс** компьютер взаимодействует с человеком, с пользователем.

Через **программный интерфейс** программы взаимодействуют друг с другом (человек тут не нужен).

Ну, и можно выделить такие еще разновидности как **сетевой протокол**, чаще всего тоже для взаимодействия программ друг с другом, но через сеть, а не непосредственно, как это происходит через программный интерфейс.

Это **файловая система**, программы могут писать данные на диск и читать данные с диска.

Это **состояние окружения**, которое могут программы модифицировать и, соответственно, тоже читать.

Это **события**, в частности, таймер. То есть некоторые механизмы отслеживания времени.

Используя все эти интерфейсы, тестировщик каким-то образом создает искусственные ситуации, и проверяет в этих ситуациях как программа себя ведет. Вот это и есть тестирование.

Можно также поговорить о ролях в проектах, требования, ай-тишные жаргоны.

**Testing Axioms**

#### It's Impossible to Test a Program Completely

Пример: даже такое просто приложение как калькулятор слишком тяжело протестировать полностью. Представьте себе, что вам необходимо протестировать калькулятор. Вы решили начать со сложения. Вы пытаетесь сложить 1+0=. Вы получаете ответ 1. Это правильно. Затем пытаетесь сложить 1+1=. Вы получаете 2. И что дальше?? Калькулятор способен складывать 32-разрядные числа, таким образом, что вы должна попробовать все возможные варианты. А если вы начнете тестировать десятичные числа. И т. д. А когда вы проверите, что обычные числа складываются правильно, вы должны попробовать неправильные вводы, чтобы убедиться, что они работают правильно. Помните, что вы не ограничены нажатием клавиш и с клавиатуры. Вы также можете попробовать протестировать буквенные значения: 1+а, z+1, 1f1+2b2, …

Калькулятор позволяет нажатие клавиш Backspace и Delete, поэтому вы должны протестировать и их. 1<backspace>2+2 должно равняться 4. Все, что бы вы не ввели должно быть протестировать совместно с нажатием Backspace для одного или нескольких вводов. И затем вы можете протестировать сложение 3-х, 4-х чисел и т. д. И это только для сложения. А есть еще вычитание, умножение. Деление, проценты.

Таким образом, мы увидели, что невозможно полностью протестировать программу, даже такую простую как калькулятор.

#### Software Testing is a Risk-Based Exercise

Если вы решили не тестировать программу полностью, то вы в некоторой степени рискуете. Вот в примере с калькулятором. Если вы не хотите тестировать сценарий 1024+1024=2048, то ведь есть вероятность, что программист допустил ошибку в сложении именно этих 2-х чисел. Вы это не протестировали, а клиент это заметил, и он найдет эту ошибку. И это будет дорогой баг, так как его найдет именно клиент.

Так как вы все протестировать не сможете, то, вероятно, вы пропустите некоторые баги. Поэтому ключевой концепцией является то, что тестировщикам необходимо изучить, как избежать выполнения огромного количества тестов и как определять: что важно протестировать, а что нет.

На графике показана зависимость между количеством тестов и количеством найденных багов. Если вы попытаетесь протестировать все, то стоимость тестирования возрастет колоссально и количество пропущенных багов уменьшается до точки, когда стоимость такого тестирования становится неоправданной. Если вы в некоторой точке прекратите тестирование, то стоимость тестирования будет маленькой, но вы пропустите большое количество багов. Поэтому целью будет найти эту оптимальную точку, когда количество тестов будет ни слишком большим, ни слишком маленьким.

#### Testing Can't Show That Bugs Don't Exist

Представьте, что есть дезинсектор, который осматривает дом на наличие насекомых. Он видит живых багов, мертвых или личинок – тогда он с уверенностью может сказать, что баги в доме есть.

Он посещает другой дом. В этот раз он не обнаруживает никаких следов багов. Он заглядывает во всевозможные места и не находит очагов заражения. Возможно, он найдет пару мертвых жуков или старых, но живых и здоровых он не обнаруживает. Можно ли с уверенностью сказать, что в доме больше нет насекомых? Нет. Можно только сделать вывод, что он просто не нашел живых багов. Если конечно не просмотреть весть дом с фундамента и до крыши, но сказать с уверенностью, что вы не пропустили ни одного, нельзя.

В тестинге все точно так же, как с дезинсектором. Можно показать, что баги есть, но утверждать, что их нет - нельзя. Вы можете выполнить тестовые кейсы, найти и зарегистрировать баги, но нет гарантии, что еще какие-нибудь баги не обнаружатся. Вы можете продолжить тестинг и, возможно, найти еще что-то.

**Важной причиной тестирования является предотвращение дефектов. Вы можете проходить тесты, находить и заводить дефекты, но гарантировать, что багов нет, вы не можете.**

#### The More Bugs You Find, the More Bugs There Are

Баги, найденные в приложении, можно сравнивать с настоящими багами. А все почему? Если вы находите ошибку, то очень большая вероятность того, что другие ошибки вы найдете где-то рядом. Почему так происходит?

- У программистов тоже бывают плохие дни. Как у всех людей. Код, написанный в один их удачных дней, может быть превосходным, но в другой день может быть плохим. Ошибка может свидетельствовать про то, что другие проблемы могут быть где-то близко.

- Программисты часто допускают одни и те же ошибки. У каждого человека есть свои привычки. Программист, который допустил однажды ошибку, с большой вероятностью допустит её еще раз.

- Некоторые баги действительно могут быть «the tip of iceberg». Часто дизайн и архитектура приложения имеют фундаментальные ошибки. Тестер, который найдет несколько багов и которые на первый взгляд могут показаться несвязанными, в действительности могут иметь общую причину.

**You can’t guarantee quality.**

Как тестировщик, вы не можете протестировать все и не отвечаете за качество продукта. Важно, что у тестировщика нет возможностей гарантировать качество, т.к. задача тестировщика – искать и находить проблемы.

**Баги**

В программировании баг обычно обозначающее ошибку в [программе](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) или системе, которая выдает неожиданный или неправильный результат. Большинство багов возникают из-за ошибок, допущенных разработчиками программы в её исходном коде, либо в её дизайне. Также некоторые баги возникают из-за некорректной работы [компилятора](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), вырабатывающего некорректный код. Программу, которая содержит большое число багов и/или баги, серьёзно ограничивающие её работоспособность, называют нестабильной или, на жаргонном языке, «глючной»

## Полный цикл тестирования и его задачи

Рассмотрим более подробно существующие активности/задачи связанные с тестированием:

1) планирование тестов:

* определение требований к тестам;
* оценка рисков;
* выбор [стратегии тестирования](http://www.4stud.info/software-construction-and-testing/lecture10.html);
* определение ресурсов;
* создание расписания/последовательностей;
* разработка Плана тестирования;

2) дизайн тестов:

* анализ объёма работ;
* определение и описание тестовых случаев;
* определение и структурирование тестовых процедур;
* обзор и оценка тестового покрытия;

3) разработка тестов:

* запись или программирование тестовых скриптов;
* определение тесто-критичной функциональности в Дизайне и Модели реализации;
* создание/подготовка внешних наборов данных;

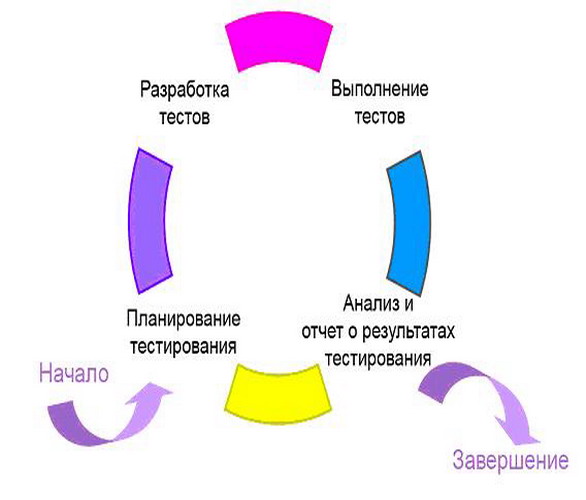
4) выполнение тестов:

* выполнение тестовых процедур;
* оценка выполнения тестов;
* восстановление после сбойных тестов;
* проверка результатов;
* исследование неожиданных результатов;
* запись ошибок;

5) оценка тестов:

* оценка покрытия тестовыми случаями;
* оценка покрытия кода;
* анализ дефектов;
* определение критериев завершения и успешности тестирования.

На основе перечисленных задач и активностей можно определить полный цикл активностей тестирования, приведенный на рис. 6.



*Рис. 6. Полный цикл тестирования, определяющий основные активности специалистов*

Таким образом, помимо уже определенной итеративности [V-модели жизненного цикла](http://www.4stud.info/software-construction-and-testing/lecture8.html) ТП, она приобретает двойную цикличность за счет того, что общие и/или частные циклы тестирования могут происходить конечное число раз в пределах итерации.

What makes a good tester?

* **Be a customer.** Будьте клиентом, попробуйте использовать систему в качестве пользователей. Чтобы понять это, поручите человеку, который не имеет представления о приложении, попользовать его некоторое время, и вы будете поражены, увидев количество проблем, с которыми этот человек столкнется. Проводя тестирование таким образом, можно столкнуться со множеством неожиданных испытаний - стрессы, нагрузки и т.д.
* **Know the technology**. Знание технологии, с помощью которой разрабатывается приложение. Это является существенным преимуществом любого тестера. Это помогает лучше проектировать дизайн, зная слабые или сильные стороны этой технологии. Хорошие тестеры знают, что приложение будет поддерживать, а что нет. И таким образом,   
  сосредоточившись на этих основных пунктах, в дальнейшем это поможет им быстро найти дефекты в приложении.
* **An explorer**. Немного творчества и отношение к риску помогает тестерам в поиске неизвестных ранее ситуаций и находить такие ошибки, которые в противном случае не заметили бы.
* **Troubleshooting**. Устранение неполадок и выяснение, почему что-то не работает, помогает тестерам быть уверенными в действительности найденной проблемы и ясно излагать дефект разработчикам.
* **Possess people skills and tenacity.** Тестеры могут столкнуться сильным сопротивлением программистов. Быть умным и социально-дипломатичным не значит, что нужно быть нерешительным. Лучшие тестеры одновременно и неконфликтны-коммуникабельны, и упорные, где это имеет значение.
* **Organized.**  Лучший тестер очень хорошо понимает, что он тоже может ошибаться, но попытка для тестирования всего одна. Он очень хорошо организован и имеют checklists для проверки, использует файлы, факты и цифры для подтверждения своих выводов, которые могут быть использованы в качестве доказательств и подтверждения своих выводов.
* **Objective and accurate.** Объективные и точные. Неточная информация теряет очень много доверия. Хорошие тестеры должны убедиться, что их выводы точные и воспроизводимые.
* **Дефекты ценны.** Хорошие тестеры учатся на дефектах. Каждый дефект - это возможность узнать новое и совершенствоваться дальше. Дефект, обнаруженный на ранней стадии, существенно дешевле по сравнению с найденным в более позднем этапе. Дефекты могут вызвать серьезные проблемы, если их не удалось найти вовремя. Изучение дефектов помогает предотвратить проблемы в будущем, отслеживать улучшения продукта, улучшать прогнозирование и оценки возникновения проблем в будущем.