В последние несколько лет **тестирование ПО** в отечественных компаниях наконец-то превратилось не просто в поиск ошибок или же средство получения информации о состоянии проекта, а в мощный инструмент разработки, позволяющий достичь нужного уровня качества.

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

Testing is a process of executing a program with intend of finding errors.

Test is the act of executing software with test cases with an objective of:

* Finding failure;
* Demonstrate correct execution.

Тестировщик:

1. Управляет выполнением программы, создает искусственные ситуации
2. Наблюдает за поведением программы и сравнивает наблюдаемое поведение с ожидаемым.

**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.

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.