**Обеспечение качества и тестирование программного обеспечения - основные понятия и определения**

**Качество программного обеспечения** **(Software Quality) -**это степень, в которой программное обеспечение обладает требуемой комбинацией свойств.

**Качество программного обеспечения** **(Software Quality) -**это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

**Обеспечение качества (Quality Assurance - QA) -**это совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения (ПО) информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО для обеспечения требуемого уровня качества выпускаемого продукта.

**Контроль качества (Quality Control - QC) -**это совокупность действий, проводимых над продуктом в процессе разработки для получения информации о его актуальном состоянии в разрезах: "готовность продукта к выпуску", "соответствие зафиксированным требованиям", "соответствие заявленному уровню качества продукта".

**Тестирование программного обеспечения** **(Software Testing) -**это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

**Верификация (verification) -**это процесс оценки системы или её компонентов с целью определения, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. Т.е. выполняются ли наши цели, сроки, задачи по разработке проекта, которые были определены в начале текущей фазы.

**Валидация (validation) -**это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

**План Тестирования** **(Test Plan) -**это документ, который описывает весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

**Тест дизайн** **(Test Design) -**это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест-кейсы), в соответствии с определенными ранее критериями качества и целями тестирования.

**Тестовый случай** **(Test Case) -**это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

**Баг/Дефект Репорт** **(Bug Report) -**это документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования с указанием причин и ожидаемого результата.

**Тестовое Покрытие** **(Test Coverage) -**это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

**Детализация Тест-Кейсов (Test Case Specification) -**это уровень детализации описания тестовых шагов и требуемого результата, при котором обеспечивается разумное соотношение времени прохождения к тестовому покрытию.

**Время Прохождения Тест Кейса(Test Case Pass Time) -**это время от начала прохождения шагов тест-кейса до получения результата теста.

**Качество программного обеспечения**

Каждый день в своей работе мы сталкиваемся с достаточно абстрактным понятием «качество ПО» и если задать вопрос тестировщику или программисту «что такое качество?», то у каждого найдется своё толкование. Рассмотрим определение "качества ПО" в контексте международных стандартов:

**Качество программного обеспечения -**это степень, в которой ПО обладает требуемой комбинацией свойств.

**Качество программного обеспечения -**это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

**Характеристики качества ПО**

**Функциональность (Functionality) -**определяется способностью ПО решать задачи, которые соответствуют зафиксированным и предполагаемым потребностям пользователя, при заданных условиях использования ПО. Т.е. эта характеристика отвечает то, что ПО работает исправно и точно, функционально совместимо соответствует стандартам отрасли и защищено от несанкционированного доступа.

**Надежность (Reliability) –**способность ПО выполнять требуемые задачи в обозначенных условиях на протяжении заданного промежутка времени или указанное количество операций. Атрибуты данной характеристики – это завершенность и целостность всей системы, способность самостоятельно и корректно восстанавливаться после сбоев в работе, отказоустойчивость.

**Удобство использования (Usability) –**возможность легкого понимания, изучения, использования и привлекательности ПО для пользователя.

**Эффективность (Efficiency) –**способность ПО обеспечивать требуемый уровень производительности, в соответствии с выделенными ресурсами, временем и другими обозначенными условиями.

**Удобство сопровождения (Maintainability) –**легкость, с которой ПО может анализироваться, тестироваться, изменяться для исправления дефектов для реализации новых требований, для облегчения дальнейшего обслуживания и адаптирования к имеющемуся окружению.

**Портативность (Portability) –** характеризует ПО с точки зрения легкости его переноса из одного окружения (software/ hardware) в другое.

**Модель качества программного обеспечения**

На данный момент, наиболее распространена и используется многоуровневая модель качества программного обеспечения, представленная в наборе стандартов ISO 9126. На верхнем уровне выделено 6 основных характеристик качества ПО, каждую из которых определяют набором атрибутов, имеющих соответствующие метрики для последующей оценки.Рис.1. Модель качества программного обеспечения (ISO 9126-1)

**Кто такой тестировщик и что он делает**

Если поискать информацию по ключевым фразам из названия этой главы,то можно найти уйму совершенно противоречивых ответов. И дело здесь, в первую очередь, в том, что авторы большинства «должностных обязанностей» приписывают всей профессии некий утрированный набор характеристик отдельных её представителей.

В начале карьеры любой специалист (и тестировщик не является исключением) является исполнителем и учеником. Достаточно хорошо понимать, что такое тест-кейсы, отчёты о дефектах, уметь читать требования, пользоваться парой инструментальных средств и хорошо уживаться в команде.

Постепенно тестировщик начинает погружаться во все стадии разработки проекта, понимая их всё полнее и полнее, начинает не только активно использовать, но и разрабатывать проектную документацию, принимать всё более ответственные решения.

Если выразить образно главную цель тестировщика, то она будет звучать так: «понимать, что в настоящий момент необходимо проекту, получает ли проект это необходимое в должной мере и, если нет, то как изменить ситуацию к лучшему». Звучит похоже на цель руководителя проекта, верно? Верно. Начиная с некоторого уровня развития, IT-специалисты, по большому счёту, различаются лишь наборами технических навыков и основной областью приложения этих навыков.

Так какие же технические навыки нужны, чтобы успешно начать работать тестировщиком? Прежде чем приступить к самому перечислению, оговорим особо: этот список рассчитан, в первую очередь, на тех, кто приходит в тестирование из не технических профессий (хотя часто его же приходится озвучивать и студентам технических вузов).

1. Знание иностранных языков. Да, это не технический навык. Но, тем не менее, он идёт под номером «ноль». Можете считать это аксиомой: «нет знания английского — нет карьеры в IT». Другие иностранные языки тоже приветствуются, но английский первичен.
2. Уверенное владение компьютером на уровне по-настоящему продвинутого пользователя и желание постоянно развиваться в этой области. Можете ли Вы представить себе профессионального повара, который не может пожарить картошку (не «не обязан», а «не умеет в принципе»)? Выглядит странно? Не менее странно выглядит «IT’шник» (именно так, в кавычках), неспособный набрать вменяемо отформатированный текст, скопировать файл по сети, развернуть виртуальную машину или выполнить любое иное повседневное рутинное действие.
3. Программирование. Оно на порядок упрощает жизнь любому IT’шнику, а тестировщику — в первую очередь. Можно ли тестировать без знания программирования? Да, можно. Можно ли это делать по-настоящему хорошо? Нет. И сейчас самый главный (почти религиозно-философский) вопрос: какой язык программирования изучать? C/C++/C#, Java, PHP, JavaScript, Python, Ruby и т.д. — начинайте с того, на чём написан ваш проект. Если проекта пока ещё нет, начинайте с JavaScript (на текущий момент — самое универсальное решение).
4. Базы данных и язык SQL. Здесь от тестировщика тоже не требуется квалификация на уровне узких специалистов, но минимальные навыки работы с наиболее распространёнными СУБД и умение писать простые запросы можно считать обязательными.
5. Понимание принципов работы сетей и операционных систем. Хотя бы на минимальном уровне, позволяющем провести диагностику проблемы и решить её своими силами, если это возможно.
6. Понимание принципов работы веб-приложений и мобильных приложений. В наши дни почти всё пишется именно в виде таких приложений, и понимание соответствующих технологий становится обязательным для эффективного тестирования.

Надеюсь, Вы обратили внимание на то, что самого тестирования в списке нет. Всё верно, ведь ему посвящена вся эта книга целиком, так что позволим себе не копировать её сюда.

Также отметим личностные качества, позволяющие тестировщику быстрее стать отличным специалистом:

1. Повышенная ответственность и исполнительность;
2. хорошие коммуникативные навыки, способность ясно, быстро, чётко выражать свои мысли;
3. терпение, усидчивость, внимательность к деталям, наблюдательность;
4. хорошее абстрактное и аналитическое мышление;
5. способность ставить нестандартные эксперименты, склонность к исследовательской деятельности.

Да, сложно найти человека, который бы в равной мере обладал всеми перечисленными качествами, но всегда полезно иметь некий ориентир для саморазвития.

**Откуда берутся ошибки в ПО?**

Почему бывает так, что программы работают неправильно? Все очень просто – они создаются и используются людьми. Если пользователь допустит ошибку, то это может привести к проблеме в работе программы – она используется неправильно, значит, может повести себя не так, как ожидалось.

**Ошибка (error)** – это действие человека, которое порождает неправильный результат.

Однако программы разрабатываются и создаются людьми, которые также могут допускать (и допускают) ошибки. Это значит, что недостатки есть и в самом программном обеспечении. Они называются дефектами или багами (оба обозначения равносильны). Здесь важно помнить, что программное обеспечение – нечто большее, чем просто код.

**Дефект, Баг (Defect, Bug)** – недостаток компонента или системы, который может привести к отказу определенной функциональности. Дефект, обнаруженный во время исполнения программы, может вызвать отказ отдельного компонента или всей системы.

При исполнении кода программы дефекты, заложенные еще во время его написания, могут проявиться: программа может не делать того, что должна или наоборот делать то, чего не должна – происходит сбой.

**Сбой (failure)** – несоответствие фактического результата (actual result) работы компонента или системы ожидаемому результату (expected result).

Сбой в работе программы может являться индикатором наличия в ней дефекта.

Таким образом, баг существует при одновременном выполнении трех условий:

* известен ожидаемый результат;
* известен фактический результат;
* фактический результат отличается от ожидаемого результата.

Важно понимать, что не все баги становятся причиной сбоев – некоторые из них могут никак себя не проявлять и оставаться незамеченными (или проявляться только при очень специфических обстоятельствах).

Причиной сбоев могут быть не только дефекты, но также и условия окружающей среды: например, радиация, электромагнитные поля или загрязнение также могут влиять на работу как программного, так и аппаратного обеспечения.

Всего существует несколько источников дефектов и, соответственно, сбоев:

* ошибки в спецификации, дизайне или реализации программной системы;
* ошибки использования системы;
* неблагоприятные условия окружающей среды;
* умышленное причинение вреда;
* потенциальные последствия предыдущих ошибок, условий или умышленных действий.

Дефекты могут возникать на разных уровнях, и от того, будут ли они исправлены и когда, будет напрямую зависеть качество системы.

**Качество (Quality)** – степень, в которой совокупность присущих характеристик соответствует требованиям.

**Качество программного обеспечения (Software Quality)**– это совокупность характеристик программного обеспечения, отражающих его способность удовлетворять установленные и предполагаемые потребности.

**Требование (Requirement)** – потребность или ожидание, которое установлено. Обычно предполагается или является обязательным.

В первом случае все было сделано правильно и мы получили продукт, полностью соответствующий ожиданиям заказчика и удовлетворяющий критериям качества.

Во втором случае ошибки были допущены уже при кодировании, что привело к появлению дефектов в готовом продукте. Но на этом уровне баги достаточно легко обнаружить и исправить, поскольку мы видим несоответствие требованиям.

Третий вариант хуже – здесь ошибки были допущены на этапе проектирования системы. Заметить это можно лишь проведя тщательную сверку со спецификацией. Исправить такие дефекты тоже непросто – нужно заново перерабатывать дизайн продукта.

В четвертом случае дефекты были заложены еще на этапе формирования требований; вся дальнейшая разработка и даже тестирование пошли по изначально неправильному пути. Во время тестирования мы не найдем багов – программа пройдет все тесты, но может быть забракована заказчиком.

Условно можно выделить **пять причин появления дефектов в программном коде**.

1. **Недостаток или отсутствие общения в команде**. Зачастую бизнес-требования просто не доходят до команды разработки. У заказчика есть понимание того, каким он хочет видеть готовый продукт, но, если должным образом не объяснить его идею разработчикам и тестировщикам, результат может оказаться не таким, как предполагалось. Требования должны быть доступны и понятны всем участникам процесса разработки ПО.
2. **Сложность программного обеспечения**. Современное ПО состоит из множества компонентов, которые объединяются в сложные программные системы. Многопоточные приложения, клиент-серверная и распределенная архитектура, многоуровневые базы данных – программы становятся все сложнее в написании и поддержке, и тем труднее становится работа программистов. А чем труднее работа, тем больше ошибок может допустить исполняющий ее человек.
3. **Изменения требований.** Даже незначительные изменения требований на поздних этапах разработки требуют большого объема работ по внесению изменений в систему. Меняется дизайн и архитектура приложения, что, в свою очередь, требует внесения изменений в исходный код и принципы взаимодействия программных модулей. Такие текущие изменения зачастую становятся источником трудноуловимых дефектов. Тем не менее, часто меняющиеся требования в современном бизнесе – скорее правило, чем исключение, поэтому непрерывное тестирование и контроль рисков в таких условиях – прямая обязанность специалистов отдела обеспечения качества.
4. **Плохо документированный код.** Сложно поддерживать и изменять плохо написанный и слабо документированный программный код. Во многих компаниях существуют специальные правила по написанию и документированию кода программистами. Хотя на практике часто бывает так, что разработчики вынуждены писать программы в первую очередь быстро, а это сказывается на качестве продукта.
5. **Средства разработки ПО.** Средства визуализации, библиотеки, компиляторы, генераторы скриптов и другие вспомогательные инструменты разработки – это тоже зачастую плохо работающие и слабо документированные программы, которые могут стать источником дефектов в готовом продукте.

**Принципы тестирования**

**Тестирование программного обеспечения**– креативная и интеллектуальная работа. Разработка правильных и эффективных тестов – достаточно непростое занятие. Принципы тестирования, представленные ниже, были разработаны в последние 40 лет и являются общим руководством для тестирования в целом.

**1. Тестирование показывает наличие дефектов**

Тестирование может показать наличие дефектов в программе, но не доказать их отсутствие. Тем не менее, важно составлять тест-кейсы, которые будут находить как можно больше багов. Таким образом, при должном тестовом покрытии, тестирование позволяет снизить вероятность наличия дефектов в программном обеспечении. В то же время, даже если дефекты не были найдены в процессе тестирования, нельзя утверждать, что их нет.

**2. Исчерпывающее тестирование невозможно**

Невозможно провести исчерпывающее тестирование, которое бы покрывало все комбинации пользовательского ввода и состояний системы, за исключениям совсем уж примитивных случаев. Вместо этого необходимо использовать анализ рисков и расстановку приоритетов, что позволит более эффективно распределять усилия по обеспечению качества ПО.

**3. Раннее тестирование**

Тестирование должно начинаться как можно раньше в жизненном цикле разработки программного обеспечения и его усилия должны быть сконцентрированы на определенных целях.

**4. Скопление дефектов**

Разные модули системы могут содержать разное количество дефектов, то есть плотность скопления дефектов в разных элементах программы может отличаться. Усилия по тестированию должны распределяться пропорционально фактической плотности дефектов. В основном, большую часть критических дефектов находят в ограниченном количестве модулей. Это проявление принципа Парето: 80% проблем содержатся в 20% модулей.

**5. Парадокс пестицида**

Прогоняя одни и те же тесты вновь и вновь, Вы столкнетесь с тем, что они находят все меньше новых ошибок. Поскольку система эволюционирует, многие из ранее найденных дефектов исправляют и старые тест-кейсы больше не срабатывают.

Чтобы преодолеть этот парадокс, необходимо периодически вносить изменения в используемые наборы тестов, рецензировать и корректировать их с тем, чтобы они отвечали новому состоянию системы и позволяли находить как можно большее количество дефектов.

**6. Тестирование зависит от контекста**

Выбор методологии, техники и типа тестирования будет напрямую зависеть от природы самой программы. Например, программное обеспечение для медицинских нужд требует гораздо более строгой и тщательной проверки, чем, скажем, компьютерная игра. Из тех же соображений сайт с большой посещаемостью должен пройти через серьезное тестирование производительности, чтобы показать возможность работы в условиях высокой нагрузки.

**7. Заблуждение об отсутствии ошибок.**

Тот факт, что тестирование не обнаружило дефектов, еще не значит, что программа готова к релизу. Нахождение и исправление дефектов будет не важным, если система окажется неудобной в использовании и не будет удовлетворять ожиданиям и потребностям пользователя.

И еще несколько важных принципов:

* тестирование должно производиться независимыми специалистами;
* привлекайте лучших профессионалов;
* тестируйте как позитивные, так и негативные сценарии;
* не допускайте изменений в программе в процессе тестирования;
* указывайте ожидаемый результат выполнения тестов.

**Верификация и валидация**

Эти два понятия тесно связаны с процессами тестирования и обеспечения качества. К сожалению, их часто путают, хотя отличия между ними достаточно существенны.

**Верификация (verification)**– это процесс оценки системы или её компонентов с целью определения того, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. То есть выполняются ли задачи, цели и сроки по разработке продукта.

**Валидация (validation)**– это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

Следующая таблица поможет выделить ключевые отличия между этими понятиями:

С помощью **валидации**Вы можете быть уверенным в том, что создали «правильный» продукт. Продукт, который полностью удовлетворяет заказчика.

С помощью **верификации**Вы можете увериться в том, что продукт сделан «правильно»: придерживаясь необходимых методик, инструментов и стандартов.

На практике отличия верификации и валидации имеют большое значение:

* заказчика интересует, в большей степени, валидация (удовлетворение собственных требований);
* исполнителя, в свою очередь, волнует не только соблюдение всех норм качества (верификация) при реализации продукта, а и соответствие всех особенностей продукта желаниям заказчика.

**QA, QC и тестирование**

Так в чем же разница между QA и тестированием и что такое Quality Control?

Многие люди до сих пор путают эти понятия, что, в общем, и не удивительно, принимая во внимание, что в нашей стране они зачастую могут использоваться для описания одних и тех же процессов. Но с формальной точки зрения, а именно она нас, как специалистов, и интересует, эти три понятия имеют существенно отличающиеся значения.

Можно оформить их соотношение в виде таблицы:

Таким образом, мы можем построить модель иерархии процессов обеспечения качества: Тестирование – часть QC. QC – часть QA.

Иными словами, Quality Assurance обеспечивает правильность и предсказуемость процесса, в то время как Quality Control предполагает контроль соблюдения требований. Тестирование же, в свою очередь, обеспечивает сбор статистических данных и внесение их в документы, созданные в рамках QC-процесса.

Если провести аналогию с процессом конструирования, скажем, велосипеда, то получим такую картину:

* С помощью тестирования мы можем определить, работают ли все детали и сам велосипед в целом так, как мы ожидаем. Из правильных ли материалов он сделан, с применением нужных методик и инструментов или нет. То есть подразумевается, что тестируемый объект уже существует.
* Задачей же QA является обеспечение соответствия всех этапов конструирования нашего велосипеда определенным стандартам качества, начиная с планирования и создания чертежей и заканчивая сборкой уже готового велосипеда. То есть качеству объекта внимание уделяется еще до создания самого объекта.