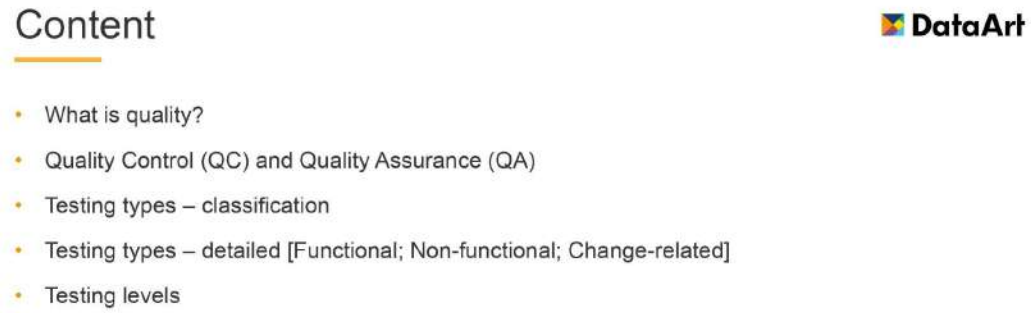
Привет меня зовут Таня, и я работаю в DataArt 7 лет в качестве QA-инженера, начинала свою карьеру с QA-школы. Работала в разных проектах, начиная от тревел и медицины, заканчивая финпрактикой и немножко телекомом. В данный момент я - сеньор QA и выполняю роль тимлида в команде на 15 человек. В ходе сегодняшней лекции мы поговорим про качество и типы тестирования:



- что это такое, каким образом можно обеспечить качество, проконтролировать, и чем отличаются понятия QC и QA;
- что такое уровни тестирования;
- какие бывают виды тестирования и чуть более подробно остановимся на некоторых из них.

Краткое содержание можно видеть на этом слайде. Давайте приступим к изучению.

Что такое качество?

Software quality

DataArt

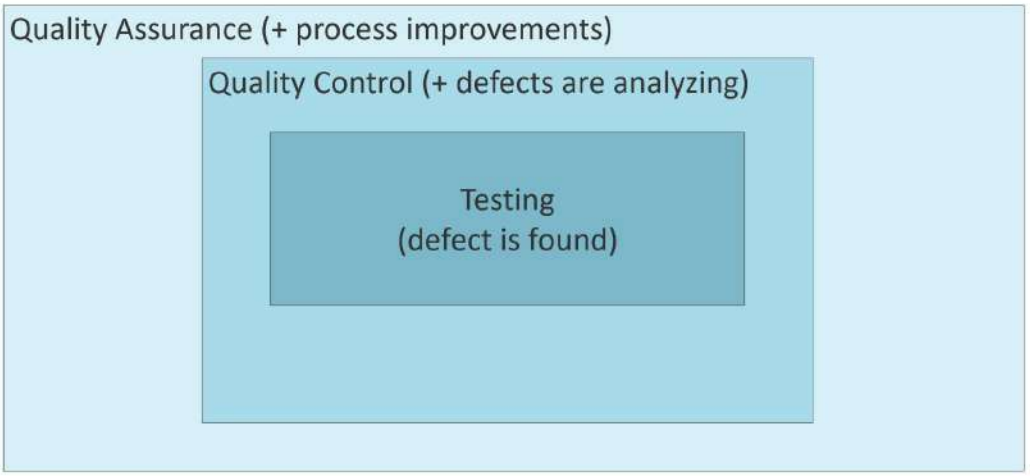
Software functional quality reflects how well it complies with a given design, based on requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product. It is the degree to which the correct software was produced.

Как говорит Википедия: качество - это некоторая мера соответствия конечного продукта заданным требованиям. Но в большинстве случаев отсутствие дефектов совершенно не означает хорошего качества, потому что основной целью любого продукта является удовлетворение конечного пользователя. Например, колбаса, которую вы купили в магазине, вам не понравилась, т.е. не соответствует для вас ожидаемому качеству, даже несмотря на то, что чисто технически она соответствует критериям, которые вы задавали при ее покупке.

Каким образом можно взаимодействовать с качеством?

Testing, QC, QA

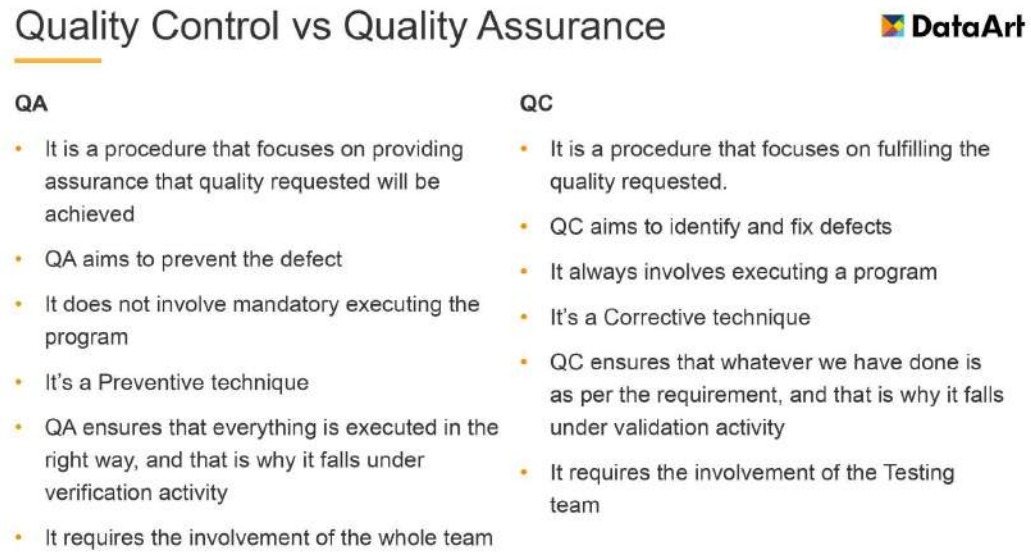
DataArt



Как рассказывал Андрей в рамках предыдущей лекции, одним из базовых понятий является тестирование. Что вообще такое тестирование? Если кратко, это

поиск дефектов, т.е. поиск несоответствий между тем, что мы имеем, и тем, что ожидалось. Помимо поиска дефектов необходимо учитывать, что эти дефекты надо исправлять, и тут в дело вступает Quality Control, то есть контроль качества. В рамках контроля качества мы находим дефекты, анализируем и исправляем, мы обеспечиваем и фиксируем заданный уровень качества в конкретный момент времени. Quality Assurance - более широкое понятие, чем Quality Control, включает в себя вынесение уроков и дальнейшее улучшение процесса, чтобы минимизировать риск возникновения дефекта в будущем.

На следующем слайде мы можем видеть краткую сводку, чем отличаются эти два процесса (QA и QC).



Quality Control - это корректирующая техника, то есть мы нашли проблему, мы её исправили, мы говорим, что теперь качество у нас в порядке. Quality Assurance - это превентивная мера, когда мы говорим, что дефектов в будущем может и не быть, если принять соответствующие меры. Например, есть многоквартирный многоэтажный дом, есть лифт, который застревает каждую неделю, и в случае Quality Control, каждую неделю этот несчастный лифт чинят, а через неделю он опять ломается. Каким образом можно избежать этих проблем в будущем? Можно проанализировать, что вызывает проблему, и устранить непосредственно причину на корню. В этом случае можно вызвать лифтёра-диагноста, который посмотрит на общее состояние, проведёт анализ и найдет, что нужно исправить, и это уже про Quality Assurance, про обеспечение качества. Важно отметить, в процессе Quality Control задействована в основном тестовая команда - специальная группа людей, которые находят расхождения, заявляют о них и передают их на исправление. Тогда как при Quality Assurance в обеспечении качества должна быть заинтересована вся команда: девелоперы, тестировщики и сам пользователь. Пока пользователь не скажет, что его не устраивает, пока тестировщики не проанализируют проблемы, не передадут ее девелоперам, а они ее не устранят и не улучшат процессы, то ни о каком обеспечении качества речи быть не может. При Quality Control у нас есть определённый уровень и мы стараемся его придерживаться, тогда как Quality Assurance позволяет не только поддерживать уровень качества, но и потенциально улучшать его в будущем.

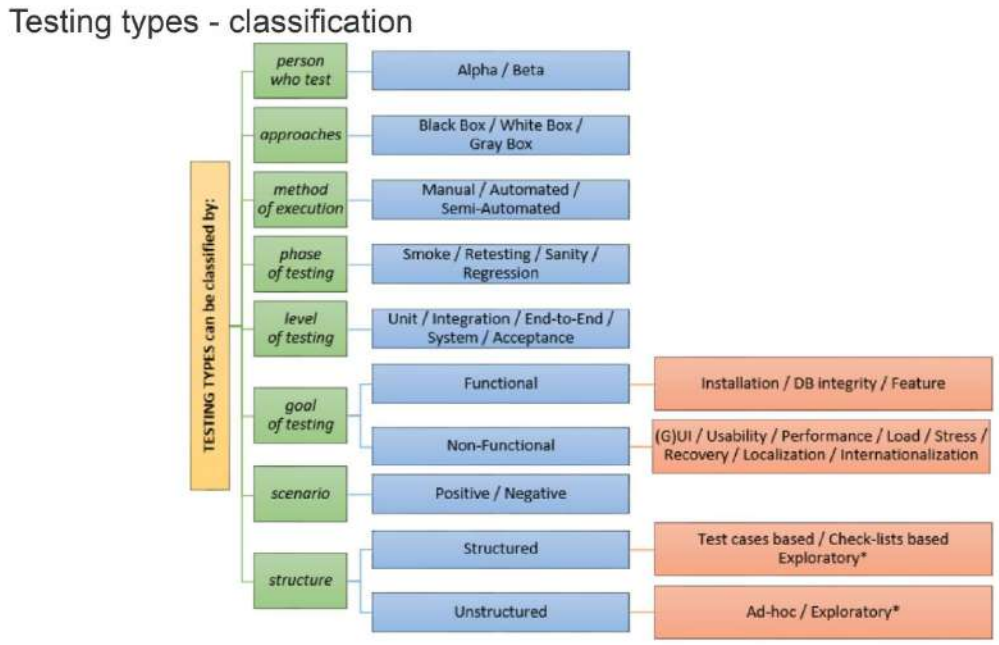
Пример сайта с не самым удачным качеством.

С точки зрения функциональности может быть этот сайт неплох, мы находим нужную нам информацию на каждом разделе сайта. Но когда мы открываем эту страничку, мы немножко теряемся. Мы не понимаем, куда мы попали, хотя в принципе на странице написано “городская клиническая поликлиника”, но мы в растерянности: разные шрифты, орфографические опечатки, дизайн не очень удачный и читабельный. У нас складывается впечатление, что если бы была альтернатива этому сайту, где мы могли бы найти ту же самую информацию, мы лучше воспользовались им, то есть мы здесь говорим о потенциальном низком качестве исполнения.

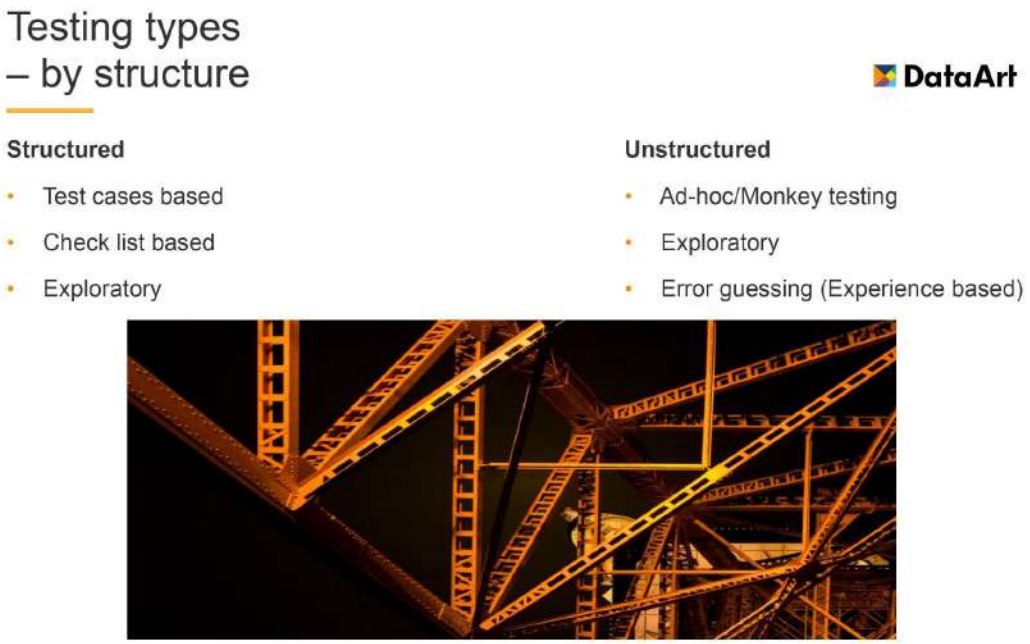
Из чего состоит процесс:

- нам предоставляют список требований, что ожидается от конкретного продукта, и непосредственно сам продукт на тестирование;
- мы изучаем, поведение и состояние продукта, находим проблемы, анализируем, исправляем и предпринимаем меры по дальнейшему их предотвращению, то есть мы получаем плюсом к списку дефектов непосредственно уже обеспеченное качество.

Классификации видов тестирования

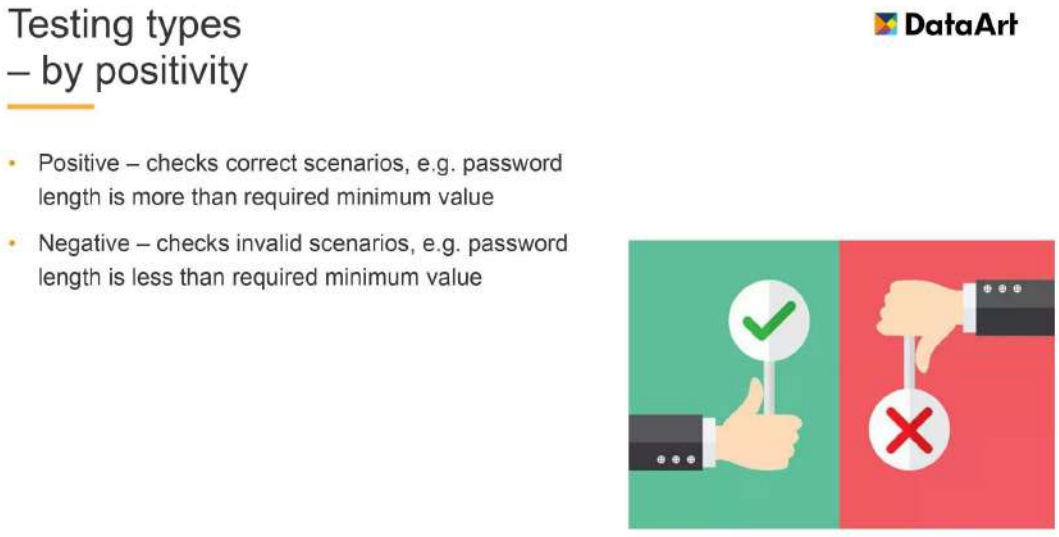


Существует несколько классификаций типов тестирования - основные виды это разделение по: подходу, методу выполнения, моменту исполнения, структурности тестирования, позитивности и уровню тестирования.
Рассмотрим отдельные классификации.



Бывает структурное и неструктурное тестирование. Они различаются тем, что в случае со структурным тестированием мы говорим о каком-то формальном и системном описании того, что нам нужно сделать. То есть либо есть хорошо прописанные тест-кейсы, либо чек-лист со списком базовых проверок, которые нам необходимо осуществить, либо другая техническая документация. В неструктурном тестирование мы можем не знать по системе вообще ничего. И тут популярным видом тестирования являются исследовательское тестирование, интуитивное тестирование и предугадывание ошибок.
В интуитивном тестировании: *Ad-hoc/Monkey testing* мы пытаемся хоть что-то сделать с продуктом, мы пытаемся его сломать, при том что мы можем не знать, какая у него была конечная цель. В некоторых случаях это эмуляция поведения конечного пользователя, который не интересовался конкретной функциональностью продукта и который просто нажимает на всё подряд, стремясь найти что-нибудь, интересующее его. Следующий вариант не структурного тестирования, это *предугадывание ошибок*, или тестирование основанное на опыте, на best practices в конкретной области. Опытный тестировщик может сказать, что скорее всего в этом месте у продукта будет знакомая известная болячка - такая же, как и его аналогов. И последний вид неструктурного тестирования, который также упоминается в структурном, это *исследовательское тестирование*. Исследовательское тестирование нельзя однозначно отнести к структурному или неструктурному, потому что в некотором числе случаев у тестировщика может быть документ с высокоуровневым описанием того, что делает система и что от неё требуется, но никакой специфичной тестовой документации может не быть, и мы изучаем продукт параллельно с тестированием, но используя доступный нам документ как некую основу.

Типы тестирования.



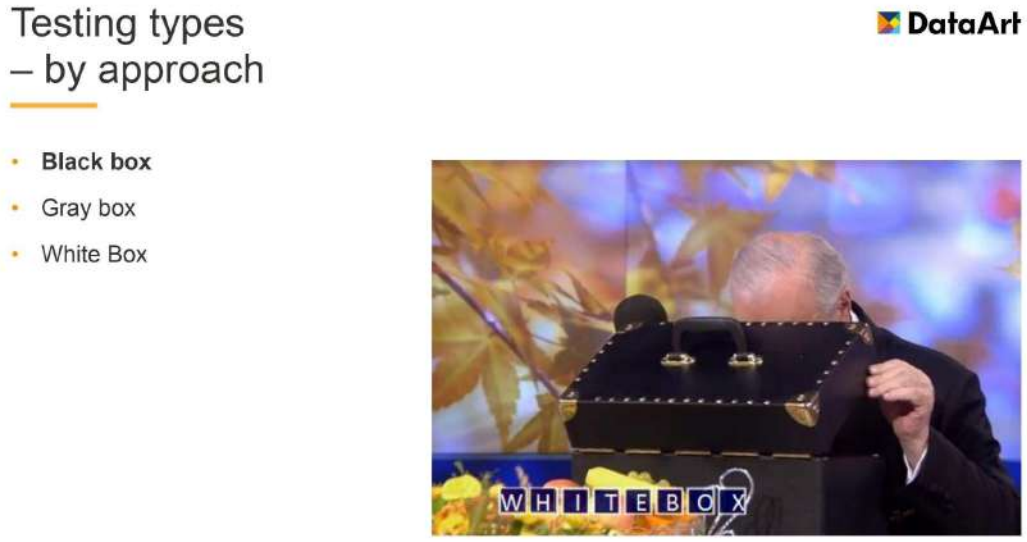
Тестирование можно разделить по позитивности исполнения, мы проверяем некоторый корректный сценарий и поведение, которое предусмотрено системой. Например, мы говорим что длина пароля не должна быть меньше 8 символов, идет проверка создания пользователя с введением пароля в 10 символов, и мы ожидаем, что система

позволит создать пользователя с заданным паролем из 10 символов. В случае негативной проверки, мы проверяем некоторые невалидные сценарии, некорректное поведение, например, как отреагирует система на создание пользователя с паролем в шесть символов.

Тестирование разделяется по методу выполнения:

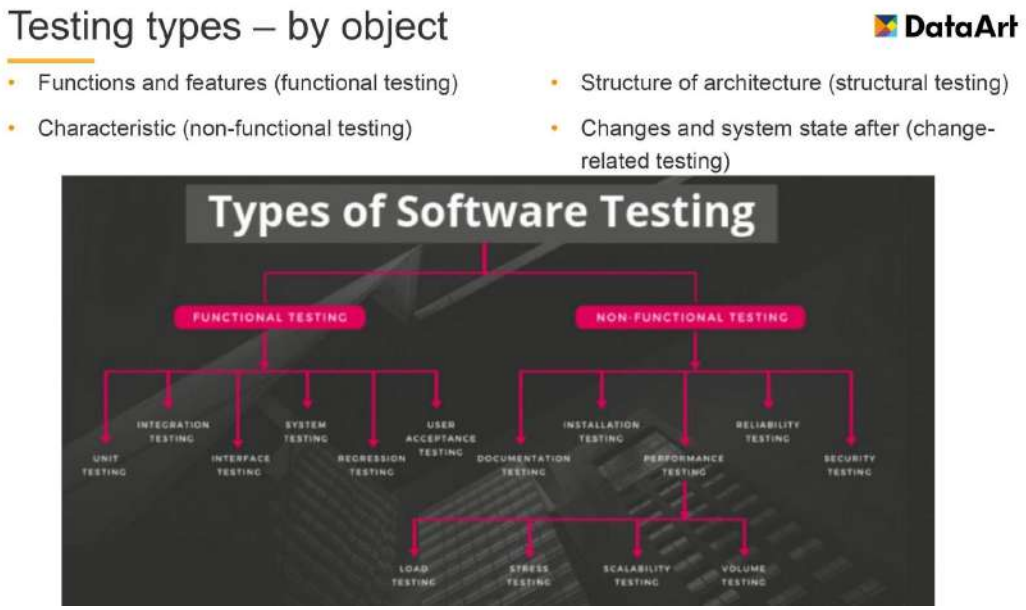
- ручное тестирование (все\большинство процессов делаются вручную);
- автоматическое (составление тестов, прогон, анализ и сбор результатов, reporting проводится в автоматическом режиме);
- полуавтоматическое (часть процессов может проходить вручную, тогда как другая автоматизирована).

Тестирование разделяется по подходу и знанию устройства системы:



- тестирование чёрного ящика (Black Box), когда нам ничего неизвестно о внутренней структуре системы, мы получаем какой-то формальный набор требований, тест-кейсов и сам продукт, и ожидаем, что при запуске конкретного сценария, мы получим определенный результат. Соответственно, мы прогоняем тест и сравниваем наши ожидания и полученный результат. Отсутствуют знания структуры, никакого доступа к архитектуре нет. Противоположностью является тестирование белого ящика (или прозрачного ящика).
- White Box testing - проверяющий знает всю систему досконально, есть доступ к исходникам, к коду, к структуре, есть возможность анализа. Данный вид тестирования, как правило, осуществляется девелоперами, потому что при тестировании белого ящика в основном проверяются логика, алгоритмы, точность написания кода, чистота кода. Одним из частных случаев является Unit testing, про который мы поговорим позже.
- Тестирование серого ящика (Grey Box) включает в себя особенности тестирования чёрного и белого ящика. Тестировщик обладает общими знаниями о том, как система устроена, но нет необходимости углубляться. В этом случае можно более эффективно составлять сценарии и тесты, проще достигать места программы и логики, которые были бы труднодоступны без знаний архитектуры.

Тестирование разделяется по объекту.



Объектом тестирования могут быть:

- конкретная функциональность - тогда это функциональное тестирование
- характеристика - это нефункциональное тестирование
- архитектура продукта - структурное тестирование.
  Пожалуйста, учитывайте, что структурное тестирование, которое упоминается здесь, и структурное тестирование, которые мы рассматривали ранее - это разные вещи. Там структурное тестирование про саму организацию тестирования, а здесь мы говорим про проверку архитектуры приложения.
- изменения - тогда это тестирование

Функциональное тестирование.

Functional testing

DataArt

Testing of the functional aspects of a software application. When you're performing functional tests, you have to test each and every functionality

What system does

- Based on **functions and features** (test basis);
- Functional testing considers the **external behavior** of software (black-box);

Applicable for all test levels

- Suitability (пригодность к использованию)
- Accuracy (аккуратность, точность)
- Interoperability (способность к взаимодействию)
- Security (безопасность)
- Functionality Compliance (соответствие стандартам и правилам)

Представляет собой проверку, ***что*** система делает, насколько корректно отрабатывают фичи. В большинстве случаев используется подход чёрного ящика (blackbox testing), при котором проверяются такие аспекты как пригодность к использованию, точность исполнения, способность к взаимодействию, безопасность соответствует формальным стандартам и нормам закона.

Нефункциональное тестирование.

Non-functional testing

DataArt

How the system works

- Reliability/Stability testing (Стабильность)
- Usability testing (Удобство использования)
- Load/Efficiency testing (Нагрузка)
- Maintainability testing (Поддержка)
- Portability testing (Портативность)
- Security testing (Безопасность)

Applicable to all test levels

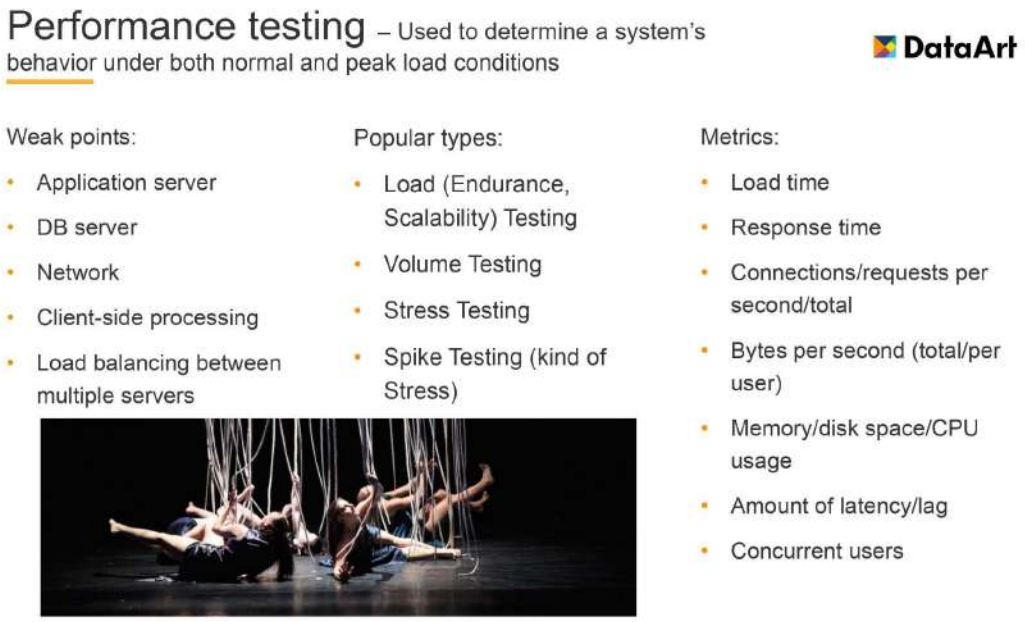
При нефункциональном тестировании проверяется ***как*** система работает, проверяются иные нефункциональные характеристики. Система стабильна и удобна, выдерживает

нагрузку, её удобно поддерживать, имеется возможность установки на заявленные платформы, проверяется безопасность системы в целом..

Maintainability testing - проверка того, насколько удобно работать с системой с точки зрения поддержки, насколько просто анализировать проблемы, вносить изменения в продукт, насколько он стабилен и тестируем. В идеале приложение должно иметь список логов, чтобы в будущем можно было легко и быстро понять, как система себя вела в тот или иной момент времени. Это удобно, чтобы понять в каком месте произошёл сбой или когда нужно оценить текущее состояние.

Reliability Testing - оценка частоты падения системы, насколько такие падения критичны, как быстро система восстанавливается, происходит ли потеря каких-то пользовательских данных - например, приложение вылетело один раз, и мы потеряли

все данные и само приложение непригодно для использования в будущем - это самый плохой сценарий. Поэтому тестированию стабильности и восстановления нужно выделять особое внимание, и оно частично перекликается с нагрузочным тестированием или **performance testing**.



Любое приложение рассчитано на определённую нагрузку. Например, ваш сайт рассчитан на 100 пользователей. Мы можем проверить, что будет с системой если залогинены 50 из них, смотрим на то, каким образом у нас идёт потребления тех или иных ресурсов - тогда мы говорим об стандартном нагрузочном тестированим (*load testing*). Когда мы смотрим, как система себя поведёт при постепенном изменении объёма нагрузки, например, постепенно наращиваем количество пользователей в системе, то это *volume testing*. Если пытаемся проверить систему на нагрузку, на которую она не рассчитана (150 пользователей) - *stress testing*. В идеале, каждая система должна быть способна выдержать краткосрочные сверхвысокие нагрузки для того, чтобы люди, которые занимаются поддержкой приложения, успели бы исправить ситуацию. *Spike testing* - тестирование, когда нагрузка распределена неравномерно, происходят некоторые пиковые нагрузки в случайные моменты времени, например, у нас сейчас залогинен 1 пользователь, а через секунду уже 90, а ещё через секунду уже 20, а потом 150, и мы смотрим как система реагирует на внезапные изменения нагрузки.

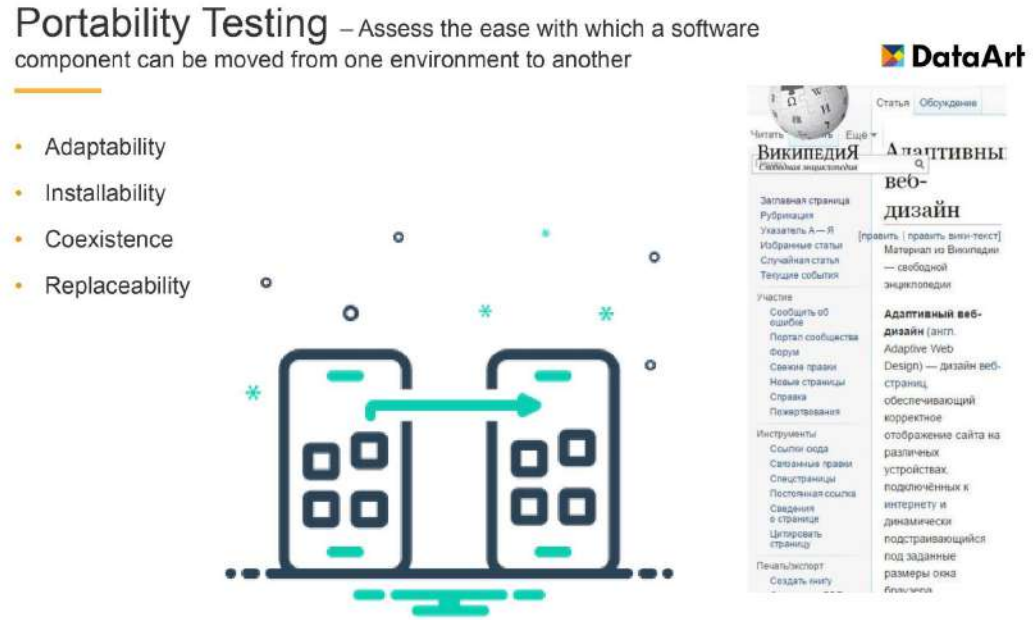
Каким образом мы вообще можем оценивать насколько система производительна? Для этого используются характеристики, которые связаны со временем, пропускной способностью, потреблением ресурсов. Такими метриками являются время загрузки, время ответа, время задержки, количество информации, которое можно обработать в единицу времени, количество соединений, пользователей в единицу времени или в целом, сколько ресурсов памяти используется. Т.е. проверяется, насколько затратна для системы разная нагрузка.

Слабые места в веб-приложениях это, как правило:

- сервера (приложение или бд),
- сеть,
- обработка данных на стороне клиента,

- балансировка между отдельными серверами или приложениями.

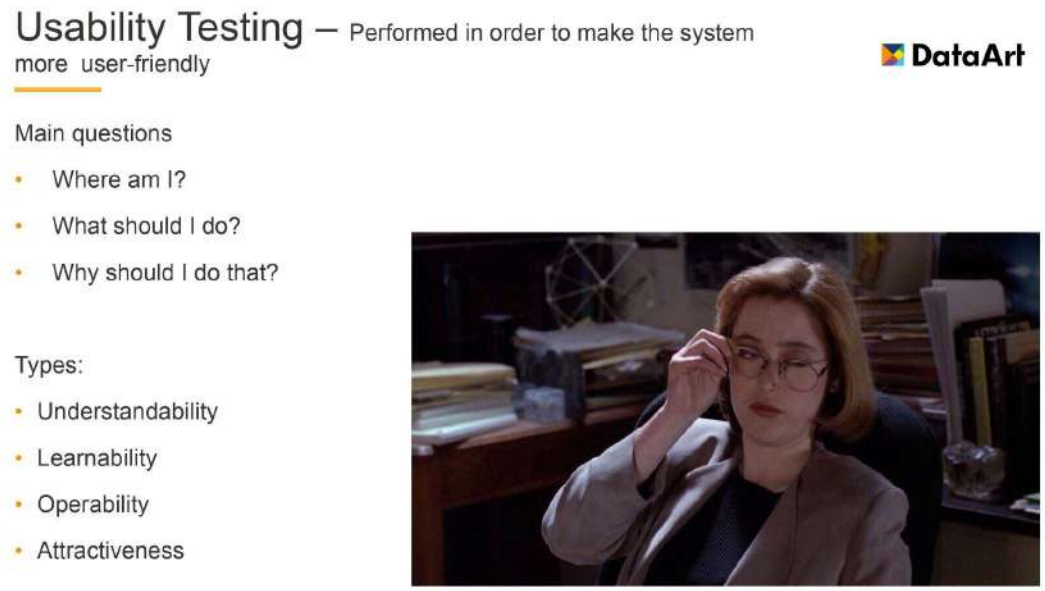
Пример: вы отправляете сообщение в мессенджере, и вместо обычных 1-2 секунд, отправка занимает 10, и вы говорите что сегодня приложение подтормаживает. В этом случае проблема может быть связана с вашей интернет-сетью, или вы обновили приложение и что-то поменялось на стороне обработки, либо на стороне компании, которые занимаются поддержкой мессенджера, произошли сбои на серверах. В любом случае мы получаем один и тот же результат, у нас происходит просадка производительности.



Portability testing - тестирование портируемости, которое проверяет насколько удобно и эффективно можно использовать ваше приложение на различных платформах, операционных системах, с использованием различного железа и окружения. Здесь выделяются такие аспекты как *адаптивность, удобство удаления и установки, возможность существовать с другими приложениями*, и *заменяемость* (насколько удобно ваше приложение заменять другим, или наоборот, вашим приложением заменить аналог). Под адаптивностью, мы предполагаем, что на любой из предлагаемых платформ приложение должно вести себя одинаково эффективно. На слайде есть примеры того, как плохо выглядит сайт Википедии с мобильной версии (страница про адаптивный веб-дизайн). При использовании сайта, который можно открыть с телефона и с десктопа, полагается, что пользователь не будет чувствовать ощутимой разницы при использовании приложения с разных видов платформ.

И про разные виды платформ - существует ряд приложений, которые изначально разработаны под конкретную операционную систему (например, Windows), а потом производитель решил размножить его на unix-подобные системы. И в случае с установкой этого приложения на Windows всё просто, но в случае с Ubuntu вам нужно открыть командную строку, выдать права суперпользователя, прогнать несколько команд и скриптов, и только после этого ваше приложение будет установлено. Соответственно, если у вас есть выбор, пользоваться этим приложением на Windows или Ubuntu, то вы с некоторой долей вероятности предпочтёте Windows, потому что так проще.

Другой интересный момент, который часто встречается при тестировании мобильных приложений, заключается в следующем - когда вы ставите новое приложение себе на телефон, например, калькулятор, у вас слетает почему-то блокнот. Это может означать, что есть некоторые проблемы с совместимостью приложений, в идеале такого происходить не должно. Иногда проблема может заключаться не только в вашем приложении, то есть калькуляторе, но и в блокноте, который перестал работать. Поэтому при возникновении подобных проблем нужен более глубокий анализ.

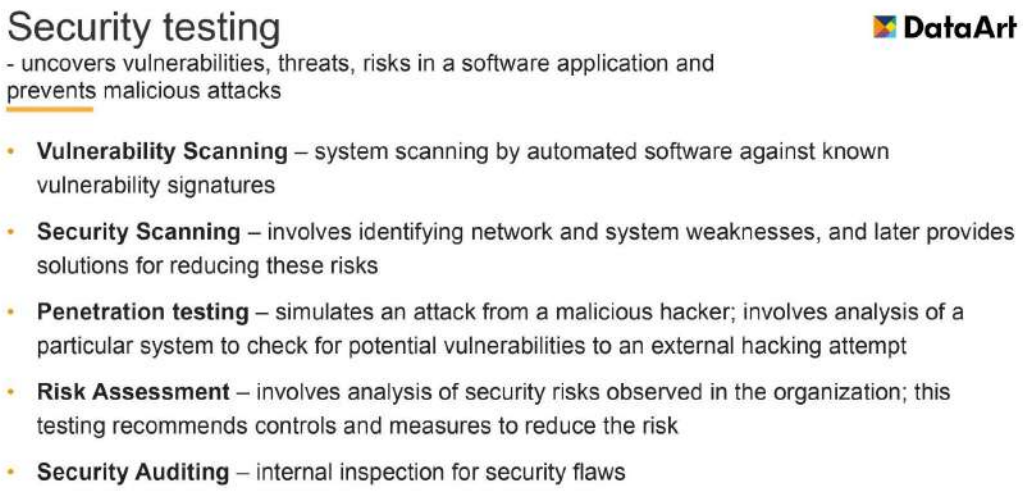


Один из важнейших видов тестирования - тестирования удобства использования (**Usability Testing**) и главные вопросы, который должен задавать себе пользователь и находить внятные ответы на них, это:

- где я нахожусь,
- что я делаю,
- и зачем я это делаю?

Если вы открываете сайт и об этом не задумываетесь, то у сайта всё не так плохо с Usability. Для того, чтобы говорить о Usability, нужно помнить, что приложение должно быть понятным, легко изучаемым, с ним должно быть легко и приятно взаимодействовать, оценивается привлекательность сайта в целом.

Пример: есть много приложений для определения номера входящего вызова, но большинство из нас пользуется одним конкретным приложением. Почему так происходит? Несмотря на то, что функциональность у всех плюс-минус схожа, вам проще пользоваться именно вот этим приложением, или здесь больше нравится дизайн кнопок. Или мне удобнее и проще пользоваться этим определителем, потому что там список злоумышленников подтягивается автоматически, мне не нужно вручную открывать словарь и переносить его в телефон. Всё это так или иначе является аспектами Usability.

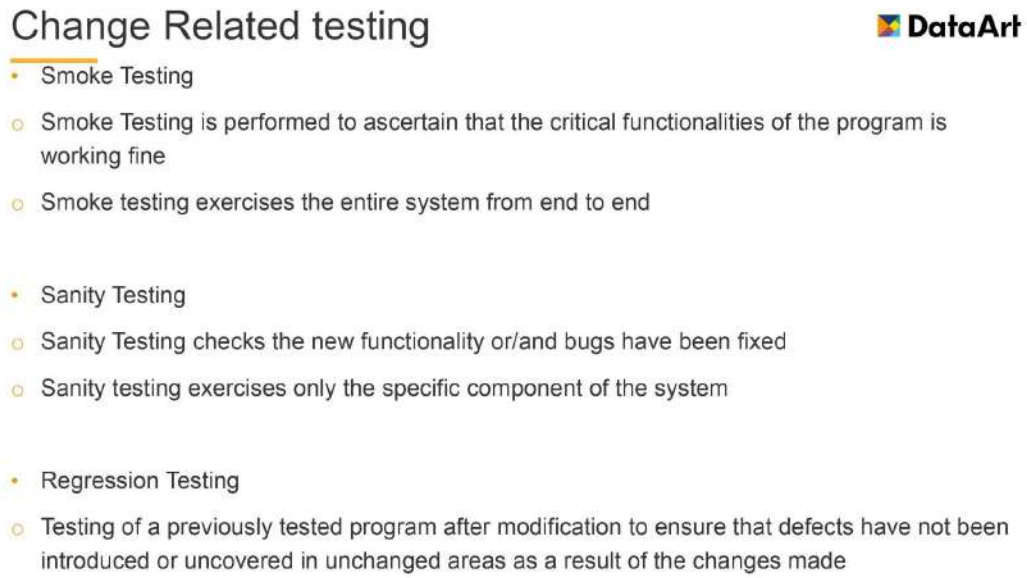


Один из самых сложных видов тестирования - тестирование безопасности (**Security testing**). Основная цель - проверить систему на наличие уязвимостей, на то, как она себя поведёт при действиях злоумышленника, какие у нас есть риски, и как мы можем это предотвратить. Наиболее популярные подвиды - это *поиск наличия уязвимостей*, как правило это один из видов автоматизированного тестирования. Мы запускаем некоторый сканер, который проверяет нашу систему на наличие известных уязвимостей, используя некие известные сигнатуры и словари. Одновременно с этим может происходить *сканирование безопасности или проверка конфигурации*, когда мы проверяем в автоматизированном режиме наличие некоторых слабых мест в системе. Например, вы купили роутер, почитали отзывы, поняли что известных уязвимостей, по крайней мере, пока нет. И надеетесь, что злоумышленник не получит доступ к вашей локальной сети. Но в то же время вы забываете о том, что надо поменять конфиги, сменить стандартный пароль. Вы подключаетесь к своей локальной сети и получаете зловред на телефон. Такое может быть возможно потому, что уязвимостей и не было, но была использована стандартная настройка, и злоумышленники этим воспользовались.

Также есть такой вид как *тестирование на взлом*, когда мы эмулируем действия хакера, основной целью которого является попасть в нашу систему и получить из неё информацию. В этом случае могут быть использованы потенциально слабые места системы, уязвимости, какие-то типичные сценарии для того, чтобы заполучить желаемое. Чтобы этого избежать, происходит *оценка рисков* - уже более формальная процедура, когда есть специально выделенная команда, которая смотрит на то, какие потенциальные риски мы несём при разработке и обслуживании нашего приложения. Как правило это выглядит так - у нас есть хороший офис и наша служба безопасности говорит, что давайте предположим ситуацию: заходит человек с улицы, вход свободный и получает доступ к компьютеру администратора, который отошел за кофе. В этом случае мы можем допустить несанкционированный доступ в сеть организации. Каким образом мы можем предотвратить этот риск? Давайте введем систему пропусков, когда у каждого сотрудника будет пропуск, на котором записана информация об уровнях доступа в те или иные помещения. Рядовой сотрудник сможет

зайти в офис, дойти до своего рабочего места, а администратор сможет зайти в админскую, где расположен главный сервер.
И самый формальный вид тестирования безопасности - *Security audit*. Приходит инспектор, который проверяет что система соответствует основным требованиям безопасности, регуляция и законам. То есть может оказаться, что у вас есть база данных, она не имеет никаких известных уязвимостей, хорошо сконфигурирована, вы пытались её взломать, технически нет никаких проблем с безопасностью, но вы храните пароли в открытом виде. В лучшем случае регулятор попросит вас заплатить штраф, потому что так делать нельзя в соответствии с законодательством. В этом случае мы говорим, что был проведен Security audit, и было вынесено предписание об устранении. А как можно исправить ситуацию в данном случае? Можно шифровать пароли.

Тестирование связанное с внесением изменений.

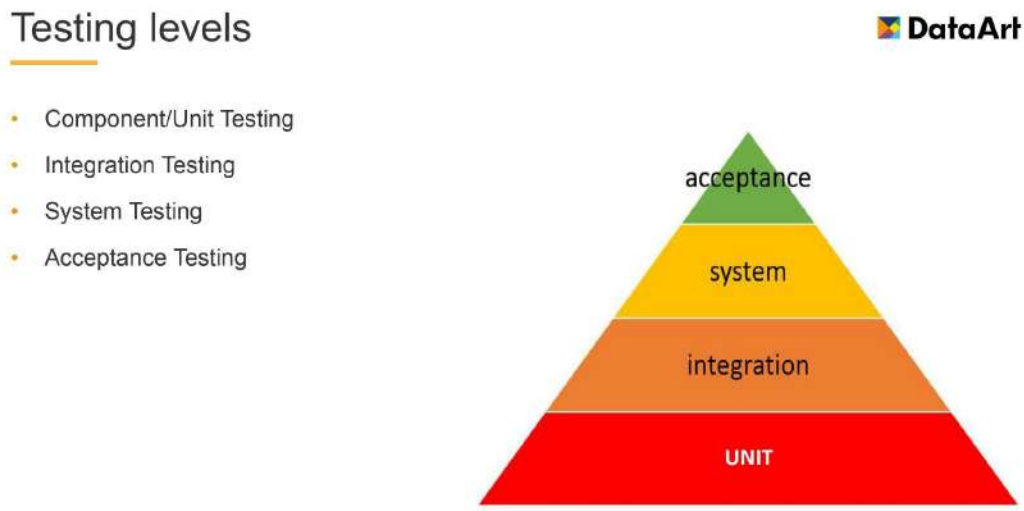


Тестирование связанное с внесением изменений разделяется на три основные группы:

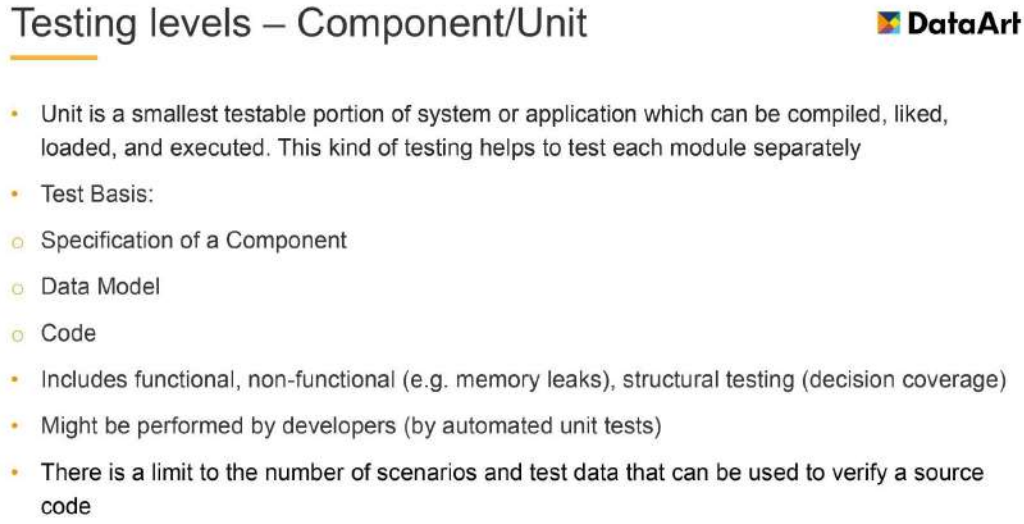
- smoke testing
- sanity testing
- regression testing

Чем они различаются между собой? Smoke testing - тестирование, при котором проверяются основные критические места системы, проверяем, что все ключевые функциональности работают как положено. То есть это некоторое тестирование ***вширь*** всей системы. Sanity testing - тестирование ***вглубь***, может понадобится тогда, когда мы внесли изменения в существующую фичу, либо мы разработали новую. И в этом случае мы проверяем отдельный компонент, отдельную функциональность, не затрагивая соседние области. И последний вид - это регрессионное тестирование, которое проверяет систему на наличие новых дефектов после внесения изменений, то есть это перетестирование всей системы. Основное отличие от smoke, в том, что при регрессии мы проверяем вообще всё. Например, бывает так, что починили один

критический баг, но этим вы привнесли 10 новых маленьких. И когда вы это находите, скорее всего, вы это делаете в процессе регрессии.

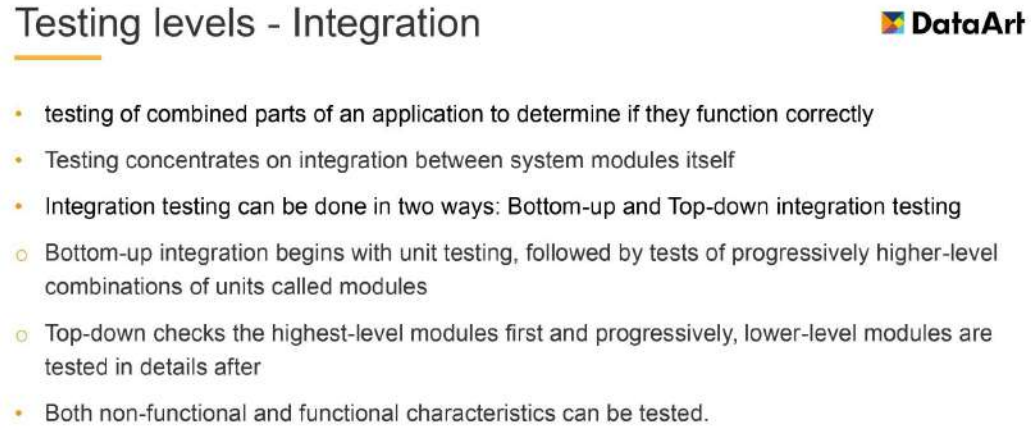


Если отвлечься от теории, то систему можно рассматривать как тортик, где его составляющей является коржик, т.е. в соответствии с построением системы - это отдельный компонент - unit. Когда мы говорим про интеграцию - два коржика должны быть надежно сцеплены между собой. Когда коржики сцеплены между собой, мы рассматриваем это уже как систему - единое целое. И когда мы несем этот тортик показывать маме, можем говорить о приемочном тестировании.
Немного подробнее про каждый уровень.



Unit тестирование.

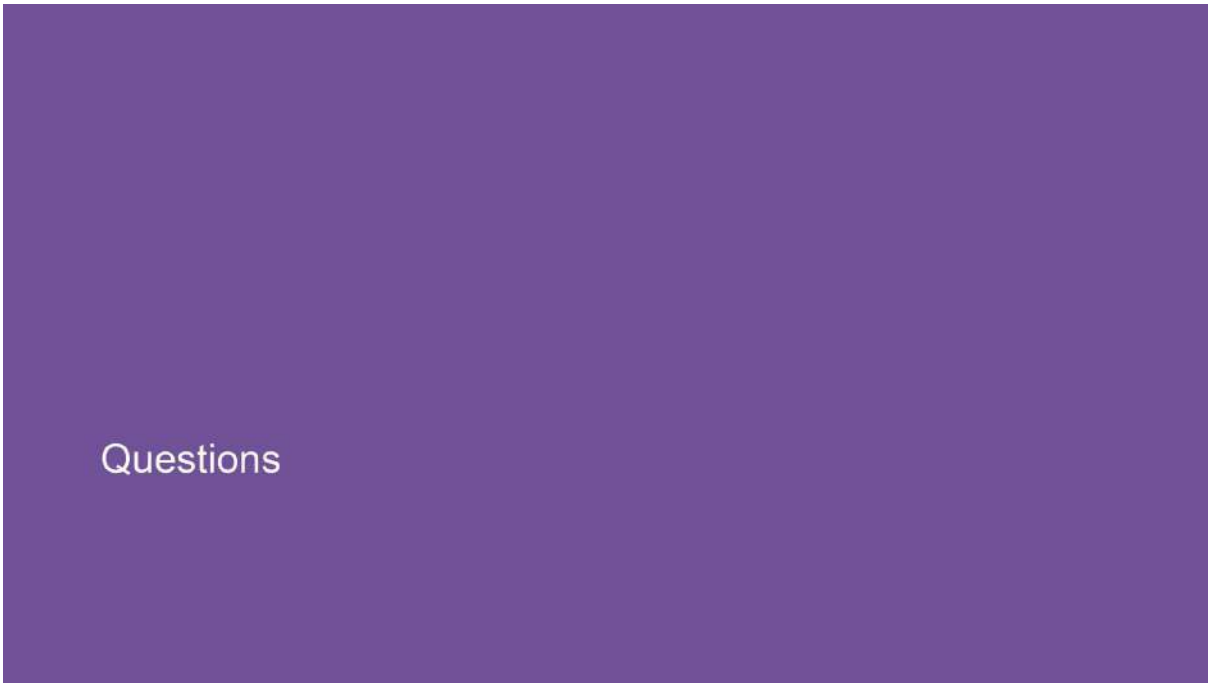
У нас есть коржик, что мы о нем знаем? Его состав, энергетическую ценность, знаем каким образом коржик был реализован. То есть мы говорим о том, что у нас была техническая документация, ключевая информация и возможно был доступ к исходному коду. В данном случае мы можем смотреть на такие аспекты компонента, как то, насколько четко и точно у нас работает конкретная фича. Либо на нефункциональные аспекты о том, что у нас это работает достаточно быстро, нету никаких утечек памяти, нет просадок по производительности. В большинстве случаев Unit тестированием занимаются непосредственно девелоперы, в этой же лекции чуть ранее я говорила о тестировании белого ящика, которое чаще всего применяется как раз в связке с Unit testing. Плюсом, а может быть минусом, данного вида тестирования является ограниченность проверок, в основном проверяется точность реализации, логика и алгоритм, который должен выполнять каждый конкретный unit.



Интеграционное тестирование.

При интеграционном тестировании мы проверяем каким образом у нас взаимодействуют несколько компонентов системы между собой, то есть у нас есть два коржика, давайте проверим, как они будут между собой сцепляться. Можем говорить о двух подходах: о подходе снизу вверх и сверху вниз. Подход сверху вниз: мы смотрим сначала на связку целиком, на несколько компонентов, которые взаимодействуют, на конкретную область, состоящую из нескольких модулей - эта область отвечает требованиям, которые мы задали. И после того, как мы убедились в том, что область выглядят адекватно, мы переходим к проверке уже каждого конкретного отдельного кусочка. По отдельности - тестирование снизу-вверх, тут всё наоборот. У нас есть опять те же самые коржики сцепленные между собой, и мы сначала рассматриваем каждый отдельный коржик, посмотрим, что каждый отдельный кусочек системы работает так, как мы ожидаем, и после этого мы переходим к более высокому уровню - смотрим связки. Когда мы говорим про интеграционное тестирование мы можем обратить внимание на функциональные и нефункциональные аспекты, то есть тортик, например, тортик должен быть вкусным, сытным, красивым. И мы смотрим на то, что коржики сытные, питательные и не разваливаются, всё это плавно перетекает в системный уровень. Таким образом, мы плавно перешли к понятию системного

тестирования, когда мы рассматриваем все наши компоненты, все наши связки, как некоторые единое целое. То есть это уже не просто набор коржиков сцепленных между собой, это уже полноценный торт. Как правила тестирование на системном уровне проводится уже исключительно тестовой командой, мы смотрим, что наша система удовлетворяет одновременно и функциональным и техническим требованиям, проверяем, что наша система отвечает нуждам бизнеса, и что мы всё сделали правильно. Когда мы убедились, что системное тестирование пройдено успешно, мы переходим к приёмочному тестированию, когда мы говорим конечному пользователю или заказчику о том, что продукт готов. В этом случае можно рассмотреть пример. Вы пришли в магазин выпечки и говорите о том, что вам нужен Медовик. И вам говорят, Медовик на вот этой полке, вы смотрите на витрину, видите торт, но он выглядит или немножко кривым, или не аппетитным, и вы не решаетесь его взять. В этом случае приемочное тестирование для конкретного тортика было завалено. То есть вы можете хорошо, классно, правильно технически реализовать продукт, но он окажется не привлекателен для конечного пользователя и не отвечать нуждам бизнеса. В идеале приемочное тестирование надо проводить на окружении, максимально похожем на реальные боевые условиям. Проводят такое тестирование как правило заказчик, либо непосредственно его представители или специально нанятая команда, обращается внимание на то, насколько удобно и правильно пользоваться вашей системой, в том числе и в плане обслуживания. Проверяется соответствие контракту и договоренностям, регулятивным нормам, проверяется ваша система не только на соответствие техническим требованиям бизнеса. Приемочное тестирование могут проводить люди как знакомые с продуктом, так и не знакомые. Это альфа- и бета-тестирование. В альфа-тестировании есть некоторая группа людей, которые хотя бы базово понимают о чём продукт, какие есть требования. Бета-тестирование - тестирование, которое проводится обычно реальными пользователями, может быть какой-то группой, контрольной выборкой, т.е. это могут быть люди, которые не вдаются глубоко в технические детали, которым важно, чтобы продукт просто работал так, как они себе это представляют.



В Интернете можно найти много таблиц и разделений видов тестирования, но я бы рекомендовала обращаться к проверенным источникам. Например, ISTQB (Международная квалификационная комиссия по тестированию программного обеспечения), рассказывает про основные моменты тестирования: виды тестирования, тестовую документацию, глоссарий с терминами на русском и английском.