**1. Что подразумевается под тестовой средой?**

Среда тестирования — это настройка программного и аппаратного обеспечения для групп тестирования для выполнения тестовых случаев.

Другими словами, он поддерживает выполнение теста с настроенным оборудованием, программным обеспечением и сетью.

**2. В чем отличие build от release?**

build — это в буквальном переводе — "сборка". Т.е. берём срез исходников, создаём некие deliverables, т.е. исполняемые файлы, конфиги, скрипты SQL и т.п. Т.е. билд — полученные из исходников рабочие продукты. Создаётся как вручную по требованию, так и автоматическими системами сборок по расписанию. Как угодно, в общем — было бы что собирать.

release — это билд, который команда разработчиков предоставляет наружу. В качестве потребителя релиза может быть как команда тестеров, так и пользователи. Соответственно, внутренний релиз — это то, что отдаётся потребителю внутри компании (или команды), внешний релиз — соответственно, отдаётся наружу.

**3. Основные фазы тестирования?**

|  |
| --- |
| – модульное тестирование; |
| – интеграционное тестирование; |
| – системное тестирование (в том числе адаптационное, функциональное и нагрузочное); |
| – регрессионное тестирование; |
| – приемочное тестирование. |

На стадии **модульного тестирования** достаточно просто выявляются просчеты в логике построения модуля и/или реализации алгоритмов, ошибки, связанные с неправильным использованием локальных ресурсов, а также ошибки в локальных переменных.

Следующим этапом является **интеграционное тестирование**. Оно проходит в несколько стадий и заключается в тестировании частей системы, состоящих из двух и большего количества модулей. На стадии интеграционного тестирования ошибки управления потоком, ошибки в передаче и/или интерпретации данных, использовании ресурсов, средств коммуникации.

Третьим этапом является **системное тестирование**, которое существенным образом отличается от модульного и интеграционного. Оно рассматривает систему как черный ящик и работает исключительно с пользовательскими интерфейсами. Основной целью системного тестирования является проверка стабильности и корректности работы системы, а также соответствия функциональностей системы спецификациям, указанным в сопроводительных документах, проверяется работы системы на базе различных операционных систем и аппаратного обеспечения. Исследуется производительность тестируемого программного обеспечения.

Основной целью **регрессионного тестирования** является проверка того, что в результате внесения изменений в программный код зафиксированные ранее ошибки исправлены, а новые не появляются. Для регрессионного тестирования используется специально разработанный набор тестов, из которого каждый раз выбираются те тесты, которые позволяют обнаружить появление новых ошибок с наибольшей вероятностью.

**Приемочное тестирование** — тестирование интегральной функциональности системы на соответствие требованиям заказчика. Планируется с участием представителей заказчика. Проводится на заключительном этапе с целью удостовериться в том, что система работает так, как и ожидалось, все необходимые формы, процедуры присутствуют и функционируют нормально. Данное тестирование является высокоуровневой проверкой отсутствия пробелов в функциональности системы.

Положительные результаты бизнес-тестирования означают, что все компоненты системы работоспособны и включены в данную версию продукта.

**4. Что такое импакт анализ?**

Импакт анализ — еще один инструмент в разработке ПО, который применяется программистами для улучшения тестирования. Основная идея импакт анализа — это указание затронутых мест в проекте при разработке новой или изменении старой функциональности.

Impact Analysis (импакт анализ) - это исследование, которое позволяет указать затронутые места (affected areas) в проекте при разработке новой или изменении старой функциональности, а также определить, насколько значительно они были затронуты.

Затронутые области требуют большего внимания во время проведения регрессионного тестирования.

* есть изменения в требованиях;
* получен запрос на внесение изменений в продукт;
* ожидается внедрение нового модуля или функциональности в существующий продукт;
* каждый раз, когда есть изменения в существующих модулях или функциональностях продукта.

(<https://habr.com/ru/post/539208/>)

**5. Критерии выбора тестов?**

1. **Критерий должен быть достаточным**, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.
2. **Критерий должен быть полным**, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.
3. **Критерий должен быть надежным**, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы
4. **Критерий должен быть легко проверяемым**, например вычисляемым на тестах

Для нетривиальных классов программ в общем случае **не существует полного и надежного критерия**, зависящего от программ или спецификаций.

Поэтому мы стремимся к идеальному общему критерию через реальные частные.

**6. Принципы тестирования?**

1. Тестирование демонстрирует наличие [дефектов](https://beqa.pro/blog/%d0%b1%d0%b0%d0%b3-%d0%b8-%d0%b1%d0%b0%d0%b3-%d1%80%d0%b5%d0%bf%d0%be%d1%80%d1%82/), а не их отсутствие
2. Исчерпывающее тестирование недостижимо
3. Раннее тестирование сохраняет время и деньги
4. Кластеризация дефектов
5. Парадокс пестицида
6. Тестирование зависит от контекста
7. Заблуждение об отсутствии ошибок

**7. Что такое независимое тестирование?**

Независимое тестирование (аутсорсинг тестирования) - это тестирование силами дополнительно привлечённой компании или группы людей, не участвующих непосредственно в разработке программного обеспечения. Данный подход имеет ряд положительных особенностей, позволяющих существенно повысить качество создаваемого программного продукта.

**8. Что такое тестирование со сдвигом влево?**

Тестирование со сдвигом влево - это подход к тестированию программного обеспечения и системного тестирования, при котором тестирование выполняется на более ранней стадии жизненного цикла (т. Е. Перемещается влево на временной шкале проекта). Это первая половина изречения «Проверяйте рано и часто». Он был придуман Ларри Смитом в 2001 году.

**9. Почему тестирование делится на отдельные этапы?**

**10. Какие есть этапы тестирования?**

Всего принято выделять 7 этапов тестирования:

1. Работа с требованиями. Знакомство с требованиями заказчика, что должен из себя представлять итоговый продукт, обсуждение.
2. Разработка стратегии тестирования. Оценка сроков тестирования, выявление среды тестирования, объединение всей информации, полученной при работе с требованиями.
3. Создание тестовой документации. Написание сценариев, которые позволят проверить функционал.
4. Тестирование прототипа. Тестирование основного функционала продукта, корректировка целей, добавление фичей.
5. Основное тестирование. Выполнение общей проверки продукта.
6. Стабилизация. На данном этапе происходит работа над устранением багов.
7. Эксплуатация. Проводится регресс-тестирование, устранение ошибок, которые нашел конечный пользователь.

**11. Как вы тестируете продукт, если требования еще не зафиксированы?**

1. Для начала делается smoke test. Проверить работают ли главные функциональности.

2. Быть на связи с project manager

3. Cделать примерный check list.

**12. Как вы узнаете, было ли создано достаточно тестов для тестирования продукта?**

## Критерии завершения или выхода:

Критерий выхода позволяет установить, какой объем тестирования следует считать достаточным. Определяется он по завершении цикла тестирования и включается в план. Это набор условий или активностей, которые должны быть выполнены, чтобы тестирование можно было назвать законченным.

* Все 100% требований учтены.
* Дефекты установлены/ожидаемое число дефектов обнаружено.
* Все дефекты, относящиеся к классу Show Stopper или Blocker, устранены, ни у одного из критических дефектов нет статуса «открытый».
* Все дефекты с высоким приоритетом идентифицированы и исправлены.
* Defect Rate (скорость дефектообразования) ниже установленного допустимого уровня.
* Очень небольшое число дефектов среднего уровня критичности «открыты», их разбор проведен.
* Число «открытых» дефектов среднего уровня, которые не влияют на пользование системой, очень небольшое.
* Все дефекты с высоким уровнем приоритета закрыты и соответствующие регрессивные сценарии успешно проведены.

**13. Что подразумевается под тестовыми данными?**

Тестовые данные — это данные, которые используются для выполнения тестов на тестовом программном обеспечении. Данные испытаний должны быть точными и исчерпывающими для выявления дефектов.

Тестовые данные на самом деле являются входными данными для программы. Он представляет данные, которые влияют или зависят от выполнения определенного модуля. Некоторые данные могут использоваться для положительного тестирования, как правило, для проверки того, что данный набор входных данных для данной функции дает ожидаемый результат. Другие данные могут использоваться для отрицательного тестирования, чтобы проверить способность программы обрабатывать необычный, экстремальный, исключительный или неожиданный ввод. Плохо спроектированные данные тестирования могут не проверять все возможные сценарии тестирования, которые будут ухудшать качество программного обеспечения.

**14. Что такое бета-тестирование?**

**Бе́та-тести́рование** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *beta testing*) — интенсивное использование почти готовой версии продукта (как правило, [программного](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) или [аппаратного обеспечения](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) с целью выявления максимального числа [ошибок](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3) в его работе для их последующего устранения перед окончательным выходом продукта на рынок, к массовому потребителю.

В отличие от [альфа-тестирования](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C%D1%84%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), проводимого силами штатных разработчиков или тестировщиков, бета-тестирование предполагает привлечение добровольцев из числа обычных будущих пользователей продукта, которым доступна предварительная версия продукта (так называемая **бета-версия**).

Такими добровольцами (их называют **бета-тестерами**) часто движет любопытство к новому продукту — любопытство, ради удовлетворения которого они вполне согласны мириться с возможностью испытать последствия ещё не найденных (а потому и не исправленных) ошибок. Кроме любопытства, мотивация может быть обусловлена желанием повлиять на процесс разработки и в итоге получать более удовлетворяющий их нужды продукт и многим другим.

Кроме того, открытие бета-тестирования может использоваться как часть стратегии продвижения продукта на рынок (например, бесплатная раздача бета-версий позволяет привлечь широкое внимание потребителей к окончательной дорогостоящей версии продукта), а также для получения предварительных отзывов о нём от широкого круга будущих пользователей[[1]](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-1).

Бета-версия не является финальной версией продукта, поэтому разработчик не гарантирует полного отсутствия ошибок, которые могут нарушить работу компьютера и/или привести к потере данных. Хотя и в финальных версиях таких гарантий разработчики, как правило, не дают.

Кроме программного и аппаратного обеспечения понятие **бета-версия** в последнее время часто применяется к сайтам, либо к их определённым разделам или функциям. При этом в большинстве случаев бета-тестирование является открытым, а доработки производятся в процессе функционирования сайта.

Бета-тестирование может быть открытым[[2]](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-2)[[3]](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-3) и закрытым[[4]](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-4), когда программу тестируют только разработчики или пользователи по приглашениям.

**15. Что означает пилотное тестирование? (Pilot)**

Пилотный тест или прогон (1–2 сессии перед реальными тестами) помогает подготовиться к исследованиям и приводит к более достоверным результатам. Такой тест позволяет утвердить формулировки задач, оценить время проведения сессий и, если все пройдет удачно, показывает дополнительные опорные точки для дальнейших исследований.

**16. Что такое бизнес – логика?**

**Бизнес-логика** — в разработке [информационных систем](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) — совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика — это реализация правил и ограничений автоматизируемых операций. Является синонимом термина «логика предметной области» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *domain logic*). Бизнес-логика задает правила, которым подчиняются данные предметной области.

Проще говоря, бизнес-логика — это реализация предметной области в [информационной системе](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). К ней относятся, например, формулы расчёта ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в [системах управления проектами](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8)), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д.

**17. Какие существуют основные виды тестирования ПО?**

* [1 Функциональное тестирование (functional testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D0.BE.D0.BD.D0.B0.D0.BB.D1.8C.D0.BD.D0.BE.D0.B5_.D1.82.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.28functional_testing.29)
* [2 Системное тестирование (system testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.A1.D0.B8.D1.81.D1.82.D0.B5.D0.BC.D0.BD.D0.BE.D0.B5_.D1.82.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.28system_testing.29)
* [3 Тестирование производительности (performance testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.A2.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.D0.BF.D1.80.D0.BE.D0.B8.D0.B2.D0.BE.D0.B4.D0.B8.D1.82.D0.B5.D0.BB.D1.8C.D0.BD.D0.BE.D1.81.D1.82.D0.B8_.28performance_testing.29)
* [4 Регрессионное тестирование (regression testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.A0.D0.B5.D0.B3.D1.80.D0.B5.D1.81.D1.81.D0.B8.D0.BE.D0.BD.D0.BD.D0.BE.D0.B5_.D1.82.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.28regression_testing.29)
* [5 Модульное тестирование (unit testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.9C.D0.BE.D0.B4.D1.83.D0.BB.D1.8C.D0.BD.D0.BE.D0.B5_.D1.82.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.28unit_testing.29)
* [6 Тестирование безопасности (security testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.A2.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.D0.B1.D0.B5.D0.B7.D0.BE.D0.BF.D0.B0.D1.81.D0.BD.D0.BE.D1.81.D1.82.D0.B8_.28security_testing.29)
* [7 Тестирование локализации (localization testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.A2.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.D0.BB.D0.BE.D0.BA.D0.B0.D0.BB.D0.B8.D0.B7.D0.B0.D1.86.D0.B8.D0.B8_.28localization_testing.29)
* [8 Юзабилити тестирование (usability testing)](http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E#.D0.AE.D0.B7.D0.B0.D0.B1.D0.B8.D0.BB.D0.B8.D1.82.D0.B8_.D1.82.D0.B5.D1.81.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D0.B5_.28usability_testing.29)

<http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E>

**18. Методы тестирования?**

* метод "Чёрного ящика" (Black box testing)
* метод "Белого ящика" (White box or «glass-box» testing)
* метод "Серого ящика" (Grey box)
* Тестирование не функциональных аспектов программы.

**19. Что такое пирамида / уровни тестирования?**

«Пирамида тестов» — метафора, которая означает группировку тестов программного обеспечения по разным уровням детализации. Она также даёт представление, сколько тестов должно быть в каждой из этих групп. Несмотря на то, что концепция тестовой пирамиды существует довольно давно, многие команды разработчиков по-прежнему пытаются неправильно реализовать её на практике должным образом. В этой статье рассматривается первоначальная концепция тестовой пирамиды и показано, как её воплотить в жизнь. Она показывает, какие виды тестов следует искать на разных уровнях пирамиды, и даёт практические примеры, как их можно реализовать.

Пирамида тестирования — один из способов обеспечения качества ПО, визуализация, которая помогает группировать тесты по типу их назначения. Так же, позволяет согласовать правила написания тестов, разделения их на типы, обозначить основной фокус тестирования в каждой из групп.

Цель использования данного процесса:

* Экономия времени и ресурсов в процессах обеспечения качества
* Сократить количество сложных кейсов тестирования
* Группировка и типизация тестов
* Снижение рисков возникновения критических дефектов ПО
* Формализация тех. долга в части QA
* Формализация правил написания тестов
* Формализация требований к кодовой базе: code coverage, тестовые планы
* Автоматизация рутинных и частых операций
* Чётко-регламентированная изоляция ресурсов в зависимости от вида тестирования

## **Уровни тестирования ПО:**

1. Модульное тестирование, иногда еще называется компонентное тестирование
2. Интеграционное тестирование
3. Системное тестирование

### **Дополнительные уровни тестирования ПО:**

4. Приемочное тестирование

5. Регрессионное тестирование

**20. Что такое деструктивное/разрушающее/негативное тестирование? (DT — Destructive testing)**

ОТРИЦАТЕЛЬНОЕ ТЕСТИРОВАНИЕ — тип тестирования ПО для поиска точек отказа в программном обеспечении, который проверяет систему на обработку исключительных ситуаций (срабатывание валидаторов на некорректные данные), а также проверяет, что вызываемая приложением функция не выполняется при срабатывании валидатора. Неожиданные условия могут быть чем угодно, от неправильного типа данных до хакерской атаки. Целью отрицательного тестирования является предотвращение сбоя приложений из-за некорректных входных данных. Просто проводя положительное тестирование, мы можем только убедиться, что наша система работает в нормальных условиях. Мы должны убедиться, что наша система может справиться с непредвиденными условиями, чтобы обеспечить 100% безошибочную систему.

Типичные примеры: ввести неправильно составленный e-mail и номер телефона, загрузить файл не предусмотренного расширения или размера.

Для деструктивного тестирования существует множество способов его тестирования:

Метод анализа точек отказа: это пошаговое прохождение системы, проводящее оценку того, что может пойти не так в разных точках. Для этой стратегии может быть использована помощь BA (Business Analyst).

Экспертная проверка тестировщика: проанализируйте или дайте на ревью ваши Test вашему коллеге-тестировщику, который менее знаком с системой/функцией

Бизнес-анализ тестовых случаев. Конечные пользователи или эксперты могут подумать о многих допустимых сценариях, которые иногда тестировщики могут не учитывать или упустить, так как все их внимание будет сосредоточено на тестировании требований.

Проведите предварительное тестирование с использованием контрольных таблиц (run sheets). Исследовательское тестирование с использованием контрольных таблиц поможет определить, что было проверено, повторить тесты и позволит вам контролировать охват тестами.

Используйте другой источник: вы можете попросить кого-нибудь сломать программный продукт и проанализировать различные сценарии.

**21. Что такое недеструктивное/неразрушающее/позитивное тестирование? (NDT – Non Destructive testing)**

НЕДЕСТРУКТИВНОЕ ТЕСТИРОВАНИЕ — это тип тестирования программного обеспечения, который включает в себя правильное взаимодействие с программным обеспечением. Другими словами, неразрушающее тестирование (NDT) также можно назвать позитивным тестированием или тестированием «счастливого пути». Это дает ожидаемые результаты и доказывает, что программное обеспечение ведет себя так, как ожидалось. Пример: - Ввод правильных данных в модуль входа в систему и проверка, принимает ли он учетные данные и переходит на следующую страницу

**22. Что подразумевается под компонентным/модульным/юнит тестированием? (Component/Module/Unit testing)**

Модульное тестирование (юнит-тестирование). Модульные тесты используются для тестирования какого-либо одного логически выделенного и изолированного элемента системы (отдельные методы класса или простая функция, subprograms, subroutines, классы или процедуры) в коде. Очевидно, что это тестирование методом белого ящика и чаще всего оно проводится самими разработчиками. Целью тестирования модуля является не демонстрация правильного функционирования модуля, а демонстрация наличия ошибки в модуле. Изоляция тестируемого блока достигается с помощью заглушек (stubs), манекенов (dummies) и макетов (mockups).

Компонентное тестирование — тип тестирования ПО, при котором тестирование выполняется для каждого отдельного компонента отдельно, без интеграции с другими компонентами. Его также называют модульным тестированием (Module testing), если рассматривать его с точки зрения архитектуры. Как правило, любое программное обеспечение в целом состоит из нескольких компонентов. Тестирование на уровне компонентов (Component Level testing) имеет дело с тестированием этих компонентов индивидуально. Это один из самых частых типов тестирования черного ящика, который проводится командой QA. Для каждого из этих компонентов будет определен сценарий тестирования, который затем будет приведен к Test case высокого уровня -> детальным Test case низкого уровня с предварительными условиями.

Исходя из глубины уровней тестирования, компоненное тестирование можно классифицировать как:

Тестирование компонентов в малом (CTIS — Component testing In Small). Тестирование компонентов может проводиться с или без изоляции остальных компонентов в тестируемом программном обеспечении или приложении. Если это выполняется с изоляцией другого компонента, то это называется CTIS. Пример: веб-сайт, на котором есть 5 разных веб-страниц, тестирование каждой веб-страницы отдельно и с изоляцией других компонентов.

Тестирование компонентов в целом (CTIL — Component testing In Large). Тестирование компонентов, выполненное без изоляции других компонентов в тестируемом программном обеспечении или приложении, называется CTIL. Давайте рассмотрим пример, чтобы понять это лучше. Предположим, что есть приложение, состоящее из трех компонентов, таких как Компонент A, Компонент B и Компонент C. Разработчик разработал компонент B и хочет его протестировать. Но для того, чтобы полностью протестировать компонент B, некоторые его функции зависят от компонента A, а некоторые — от компонента C. Функциональный поток: A -> B -> C, что означает, что существует зависимость от B как от A, так и от C, заглушка - вызываемая функция, а драйвер - вызывающая функция. Но компонент A и компонент C еще не разработаны. В этом случае, чтобы полностью протестировать компонент B, мы можем заменить компонент A и компонент C заглушкой и драйверами по мере необходимости. Таким образом, в основном, компоненты A & C заменяются заглушками и драйверами, которые действуют как фиктивные объекты до тех пор, пока они фактически не будут разработаны.

Unit testing:Тестирование отдельных программ, модулей, функций для демонстрации того, что программа выполняется согласно спецификации

Проверка в(на) соответствии с design documents

Пишутся и выполняются(обычно) разработчиками

Выполняется первым

Component testing:

Тестирование каждого объекта или частей программного обеспечения отдельно с или без изоляции других объектов

Проверка в(на) соответствии с test requirements, use case

Выполняется тестировщиками

Выполняется после Unit

Другой источник:

Разница между компонентным и модульным тестированием: По-существу эти уровни тестирования представляют одно и тоже, разница лишь в том, что в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы, а в модульном тестировании — конкретные значения

**23. Что подразумевается под интеграционным тестированием?**

Интеграционное тестирование предназначено для проверки насколько хорошо два или более модулей ПО взаимодействуют друг с другом, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Уровни интеграционного тестирования:

Компонентный интеграционный уровень (Component Integration testing)

Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

Системный интеграционный уровень (System Integration testing)

Проверяется взаимодействие между разными системами после проведения системного тестирования.

Подходы к интеграционному тестированию:

Подход Большого взрыва:

Инкрементальный подход:Нисходящий подходПодход снизу-вверхСэндвич-подход

Некоторые утверждают, что всех участников (например, вызываемые классы) тестируемого субъекта следует заменить на имитации (mocks) или заглушки (stubs), чтобы создать идеальную изоляцию, избежать побочных эффектов и сложной настройки теста. Другие утверждают, что на имитации и заглушки следует заменять только участников, которые замедляют тест или проявляют сильные побочные эффекты (например, классы с доступом к БД или сетевыми вызовами). Иногда эти два вида юнит-тестов называют одинокими (solitary) в случае тотального применения имитаций и заглушек или общительными (sociable) в случае реальных коммуникаций с другими участниками.

Информация должна приходить в течение нескольких секунд или нескольких минут с быстрых тестов на ранних этапах конвейера. И наоборот, более длительные тесты — обычно с более широкой областью — размещаются на более поздних этапах, чтобы не тормозить фидбек от быстрых тестов.

**24. Разница между Unit testing и Integration testing?**

Интеграционные тесты показывают, что не работает. Но они бесполезны в догадках, где может быть проблема.

Модульные тесты являются единственными тестами, которые говорят вам, где именно ошибка. Чтобы получить эту информацию, они должны запустить метод в смоделированной среде, где все другие зависимости должны работать правильно.

Вот почему я думаю, что ваше предложение «Или это просто юнит-тест, охватывающий 2 класса» как-то смещено. Модульный тест никогда не должен охватывать 2 класса.

Именно здесь больше всего споров о названиях. «Область» интеграционных тестов также весьма противоречива, особенно по характеру доступа к приложению (тестирование в чёрном или белом ящике; разрешены mock-объекты или нет). На практике получается так: если тест...

использует базу данных,

использует сеть для вызова другого компонента/приложения,

использует внешнюю систему (например, очередь или почтовый сервер),

читает/записывает файлы или выполняет другие операции ввода-вывода,

полагается не на исходный код, а на бинарник приложения,

... то это интеграционный, а не модульный тест

Подведём итог: хотя теоретически можно использовать только интеграционные тесты, на практике

Юнит-тесты легче поддерживать.

Юнит-тесты легко воспроизводят пограничные случаи и редкие ситуации.

Юнит-тесты выполняются гораздо быстрее интеграционных тестов.

Сбойные юнит-тесты легче исправить, чем интеграционные.

Если у вас есть только интеграционные тесты, то вы впустую тратите и время разработки, и деньги компании. Нужны как модульные, так и интеграционные тесты одновременно. Они не взаимоисключающие.

**25. Что такое системное интеграционное тестирование? (SIT — System Integration testing)**

Это тип тестирования программного обеспечения, проводимого в интегрированной аппаратной и программной среде для проверки поведения всей системы. Это тестирование, проведенное на полной интегрированной системе для оценки соответствия системы ее установленным требованиям. SIT выполняется для проверки взаимодействия между модулями программной системы. Оно занимается проверкой требований к программному обеспечению высокого и низкого уровня, указанных в Software Requirements Specification/Data and the Software Design Document. Он также проверяет сосуществование программной системы с другими и тестирует интерфейс между модулями программного приложения. В этом типе тестирования модули сначала тестируются индивидуально, а затем объединяются в систему. Например, программные и / или аппаратные компоненты объединяются и тестируются постепенно, пока не будет интегрирована вся система.

**26. В чем разница между тест-драйвером и тест-заглушкой? (Test Driver and Test Stub)**

Тестовый драйвер — это фрагмент кода, который вызывает тестируемый программный компонент. Это полезно при тестировании по принципу «снизу-вверх». Тестовая заглушка - это фиктивная программа, которая интегрируется с приложением для полной функциональности. Они актуальны для тестирования, в котором используется нисходящий подход. Давайте возьмем пример.

Допустим, есть сценарий для проверки интерфейса между модулями A и B. Мы разработали только модуль-A. Затем мы можем проверить модуль-A, если у нас есть реальный модуль-B или фиктивный модуль для него. В этом случае мы называем модуль-B тестовой заглушкой.

Теперь модуль B не может отправлять или получать данные напрямую из модуля A. В таком сценарии мы перемещаем данные из одного модуля в другой, используя некоторые внешние функции, называемые Test Driver.

Заглушки и драйверы не реализуют всю логику программного модуля, а только моделируют обмен данными с вызывающим модулем. Заглушка: вызывается тестируемым модулем. Драйвер: вызывает модуль для тестирования.

**27. Что подразумевается под системным тестированием?**

Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т. д. Для минимизации рисков, связанных с особенностями поведения в системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи. Системное тестирование относят к черному ящику.

Можно выделить два подхода к системному тестированию:

на базе требований (requirements based): Для каждого требования пишутся Test case, проверяющие выполнение данного требования.

на базе случаев использования (use case based): На основе представления о способах использования продукта создаются случаи использования системы (Use Cases). По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся Test case, которые должны быть протестированы.

**28. Что такое функциональное тестирование?**

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (компонентном, интеграционном, системном, приемочном). Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases).

Тестирование в перспективе «требования» использует спецификацию функциональных требований к системе как основу для дизайна Test case. В этом случае необходимо сделать список того, что будет тестироваться, а что нет, приоритезировать требования на основе рисков (если это не сделано в документе с требованиями), а на основе этого приоритезировать тестовые сценарии. Это позволит сфокусироваться и не упустить при тестировании наиболее важный функционал.

Тестирование в перспективе «бизнес-процессы» использует знание этих самых бизнес-процессов, которые описывают сценарии ежедневного использования системы. В этой перспективе тестовые сценарии (test scripts), как правило, основываются на случаях использования системы (use cases).

Преимущества функционального тестирования:

имитирует фактическое использование системы;

Недостатки функционального тестирования:

возможность упущения логических ошибок в программном обеспечении;

вероятность избыточного тестирования.

**29. Что такое тестирование совместимости/взаимодействия? (Compatibility/Interoperability testing)**

Тестирование взаимодействия — функциональное тестирование, проверяющее способность приложения/устройства взаимодействовать с одним и более компонентами/системами/устройствами и включающее в себя тестирование совместимости (compatibility testing) и интеграционное тестирование (integration testing).

ПО с хорошими характеристиками взаимодействия может быть легко интегрировано с другими системами, не требуя каких-либо серьезных модификаций. В этом случае, количество изменений и время, требуемое на их выполнение, могут быть использованы для измерения возможности взаимодействия. Например, тестирование совместимости проводится между смартфонами и планшетами для проверки передачи данных через Bluetooth.

Существуют разные уровни тестирования совместимости:

Аппаратное обеспечение: проверяет совместимость программного обеспечения с различными аппаратными конфигурациями.

Операционные системы: Он проверяет ваше программное обеспечение на совместимость с различными операционными системами, такими как Windows, Unix\*, Mac OS и т. д.

Программное обеспечение: проверяет ваше разработанное программное обеспечение на совместимость с другим программным обеспечением. Например, приложение MS Word должно быть совместимо с другими программами, такими как MS Outlook, MS Excel, VBA и т. д.

Сеть: оценка производительности системы в сети с различными параметрами, такими как пропускная способность, скорость работы, емкость.

Браузер: проверяет совместимость вашего сайта с различными браузерами, такими как Firefox, Google Chrome, Internet Explorer и т. д.

Устройства: проверяет совместимость вашего программного обеспечения с различными устройствами, такими как устройства USB-порта, принтеры и сканеры, другие мультимедийные устройства и Bluetooth.

Mobile: проверка совместимости вашего программного обеспечения с мобильными платформами, такими как Android, iOS и т. д.

Версии программного обеспечения. Он проверяет совместимость вашего программного приложения с различными версиями программного обеспечения. Например, проверка вашего Microsoft Word на совместимость с Windows 7, Windows 7 SP1, Windows 7 SP2, Windows 7 SP3.

Существует два типа проверки версий:

Тестирование обратной совместимости предназначено для проверки поведения разработанного аппаратного / программного обеспечения с использованием более старых версий аппаратного / программного обеспечения.

Тестирование прямой совместимости заключается в проверке поведения разработанного аппаратного / программного обеспечения с использованием более новых версий аппаратного / программного обеспечения.

Пример тестирования взаимодействия:

Подключите (connect) два или более устройств от разных производителей

Проверьте связь между устройствами

Проверьте, может ли устройство отправлять / получать пакеты или фреймы друг от друга

Проверьте, правильно ли обрабатываются данные на уровне сети и объектов

Проверьте, правильно ли работают реализованные алгоритмы

Результат в порядке: проверьте следующий результат. Результат не в порядке: используйте инструменты мониторинга, чтобы обнаружить источник ошибки

Отчет о результатах в тестовом отчете.

**30. Что такое тестирование на соответствие? (Conformance/Compilance testing)**

CONFORMANCE testing — это тип тестирования программного обеспечения, который удостоверяет, что система программного обеспечения соответствует стандартам и правилам, определенным IEEE, W3C или ETSI. Цель проверки соответствия состоит в том, чтобы определить, в какой степени отдельная реализация конкретного стандарта соответствует индивидуальным требованиям этого стандарта. Включает в себя:

Производительность

Функции

Прочность (Robustness)

Совместимость (Interoperability)

Поведение системы

Тестирование соответствия может быть логическим или физическим, и оно включает в себя следующие типы тестирования:

Тестирование на соответствие (Compliance testing)

Нагрузочное тестирование (Load testing)

Стресс тестирование (Stress testing)

Объемное тестирование (Volume testing)

Conformance testing:

-Conformance является формальным и точным способом тестирования стандартов

-Сертификация Conformance применима только к операционной системе, имеющей официальный Certification Authority

-Conformance testing используется для тестирования системы, которая обеспечивает полную поддержку данных стандартов

Compliance testing:

-Compliance является неформальным и менее точным способом тестирования стандартов

-Операционная система, которая обеспечивает единый API (Portable Operating System Interface), считается совместимой

-Compliance testing используется для тестирования системы, обеспечивающей поддержку некоторых из указанных стандартов

Тестирование соответствия также называется Type testing, который является формальным способом тестирования.

**31. Что такое нефункциональное тестирование?**

НЕФУНКЦИОНАЛЬНОЕ тестирование определяется как тип тестирования ПО для проверки нефункциональных аспектов ПО. Оно предназначено для проверки готовности системы по нефункциональным параметрам, которые никогда не учитываются при функциональном тестировании.

Нефункциональное тестирование должно повысить удобство использования, эффективность, ремонтопригодность и portability продукта.

Помогает снизить производственный риск и затраты, связанные с нефункциональными аспектами продукта.

Позволяет:

оптимизировать способ установки, настройки, выполнения, управления и мониторинга продукта.

Собирать и производить измерения и метрики для внутренних исследований и разработок.

Улучшать и расширять знания о поведении продукта и используемых технологиях.

Основные нефункциональные типы тестирования:

Производительности (Performance)Стрессовое (Stress testing)Тестирование ёмкости/способностей (Capacity testing)Нагрузочное (Load testing)Объемное тестирование (Volume testing)Выносливости/стабильности/надежности (Soak/Endurance/Stability/Reliability testing)Шиповое (Spike)Масштабируемости (Scalability Test)

Тестирование времени отклика (Response Time testing)

Тестирование на отказоустойчивость (Failover testing)

Тестирование совместимости (Compatibility testing)

Тестирование на удобство пользования (Usability testing)

Тестирование на поддерживаемость/ремонтопригодность (Maintainability testing)

Тестирование безопасности (Security testing)

Тестирование аварийного восстановления (Disaster Recovery testing)

Тестирование на соответствие (Compliance testing)

Тестирование переносимости (Portability testing)

Тестирование эффективности (Efficiency testing)

Базовое тестирование (Baseline testing)

Тестирование документации (Documentation testing)

Тестирование восстановления (Recovery testing)

Интернационализация (Globalization/Internationalization testing)

Тестирование локализации (Localization testing)

**32. Что вы знаете о Тестировании удобства пользования?**

Тестирование удобства пользования - это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.

Тестирование удобства пользования дает оценку уровня удобства использования приложения по следующим пунктам:

производительность, эффективность (efficiency) — сколько времени и шагов понадобится пользователю для завершения основных задач приложения, например, размещение новости, регистрации, покупка и т. д. ? (меньше - лучше)

правильность (accuracy) — сколько ошибок сделал пользователь во время работы с приложением? (меньше - лучше)

активизация в памяти (recall) - как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени? (повторное выполнение операций после перерыва должно проходить быстрее чем у нового пользователя)

эмоциональная реакция (emotional response) - как пользователь себя чувствует после завершения задачи — растерян, испытал стресс? Порекомендует ли пользователь систему своим друзьям? (положительная реакция - лучше)

Проверка удобства использования может проводиться как по отношению к готовому продукту, посредством тестирования черного ящика (black box testing), так и к интерфейсам приложения (API), используемым при разработке — тестирование белого ящика (white box testing). В этом случае проверяется удобство использования внутренних объектов, классов, методов и переменных, а также рассматривается удобство изменения, расширения системы и интеграции ее с другими модулями или системами. Использование удобных интерфейсов (API) может улучшить качество, увеличить скорость написания и поддержки разрабатываемого кода, и как следствие улучшить качество продукта в целом.

Отсюда становится очевидно, что тестирование удобства пользования может производиться на разных уровнях разработки ПО: модульном, интеграционном, системном и приемочном.

**33. Отличия тестирование на удобство пользования и тестирования доступности? (Usability Vs. Accessibility testing)**

USABILITY testing показывает, насколько проста в использовании и удобна система программного обеспечения. Здесь небольшой набор целевых конечных пользователей «использует» программную систему для выявления дефектов юзабилити. Основное внимание в этом тестировании уделяется простоте использования приложения пользователем, гибкости в управлении средствами управления и способности системы выполнять свои задачи. Это также называется тестированием пользовательского опыта (UX - user expirience). Это тестирование рекомендуется на начальном этапе разработки SDLC, что позволяет лучше понять ожидания пользователей. Исследования (Virzi, 1992 и Neilsen Landauer, 1993) показывают, что 5 пользователей достаточно для выявления 80% проблем с юзабилити, хотя некоторые исследователи предлагают другие цифры.

Тестирование доступности (accessibility testing) — это подмножество юзабилити-тестирования. Его цель — убедиться в том, что наш продукт удобен в использовании для людей с различными видами инвалидности или особенностей восприятия. Это могут быть проблемы со зрением, слухом или ограничения в подвижности рук.

Ваш продукт должен правильно работать с соответствующим ПО. Примеры такого программного обеспечения:

Speech Recognition Software — ПО преобразует произнесенное слово в текст, который служит вводом для компьютера.

Программа для чтения с экрана — используется для озвучивания текста, отображаемого на экране

Программное обеспечение для увеличения экрана — используется для увеличения масштаба элементов и облегчения чтения для пользователей с нарушениями зрения.

Специальная клавиатура, облегчающая ввод для пользователей, у которых проблемы с двигательными функциями.

Еще один из примеров — люди с цветовой слепотой (дальтонизмом). Эта особенность довольно широко распространена. Различными видами цветовой слепоты страдают около 8 % мужчин и 0,4 % женщин - не так уж мало!

Цвет не должен быть единственным способом передачи информации. Если вы используете цвет для того, чтобы, допустим, отобразить статус, эту информацию стоит продублировать еще каким-то образом — геометрическими фигурами, иконками или текстовым комментарием. Хорошая контрастность. Хорошая контрастность обеспечивает нормальную видимость элементов управления и текста даже для людей, не различающих те или иные оттенки. Есть отличный инструмент для тестирования веб-сайтов на предмет доступности для людей с различными формами цветовой слепоты: Color Blind Web Page Filter.

Если вы хотите сократить количество тестов, можно ограничиться только тремя фильтрами: дейтеранопия, протанопия и тританопия. Это наиболее выраженные формы цветовой слепоты (не считая крайне редкого черно-белого зрения). Остальные люди с особенностями цветовосприятия видят больше оттенков, и если ваш UI достаточно хорошо виден с этими тремя фильтрами, то и для остальных будет отображаться корректно.

Пример чек-листа:

Предоставляет ли приложение клавиатурные эквиваленты для всех действий мышью и окон?

Предоставляются ли инструкции как часть пользовательской документации или руководства? Легко ли понять и использовать приложение, используя документацию?

Упорядочены ли вкладки логически для обеспечения плавной навигации?

Предусмотрены ли сочетания клавиш для меню?

Поддерживает ли приложение все операционные системы?

Четко ли указано время отклика каждого экрана или страницы, чтобы конечные пользователи знали, как долго ждать?

Все ли надписи правильно написаны?

Являются ли цвета подходящим для всех пользователей?

Правильно ли используются изображения или значки, чтобы их было легко понять конечным пользователям?

Есть ли звуковые оповещения?

Может ли пользователь настроить аудио или видео элементы управления?

Может ли пользователь переопределить шрифты по умолчанию для печати и отображения текста?

Может ли пользователь настроить или отключить мигание, вращение или перемещение элементов?

Убедитесь, что цветовое кодирование никогда не используется в качестве единственного средства передачи информации или указания на действие

Видна ли подсветка с инвертированными цветами?

Тестирование цвета в приложении путем изменения контрастности

Правильно ли слышат люди с ограниченными возможностями всё имеющее отношение к аудио и видео?

Протестируйте все мультимедийные страницы без мультимедиа-оборудования.

Предоставляется ли обучение пользователям с ограниченными возможностями, что позволит им ознакомиться с программным обеспечением или приложением?

**34. Что такое тестирование интерфейса?**

Это тип интеграционного теста, который проверяет, правильно ли установлена связь между двумя различными программными системами. Соединение, которое объединяет два компонента, называется интерфейсом. Этот интерфейс в компьютерном мире может быть чем угодно, как API, так и веб-сервисами и т. д. Тестирование этих подключаемых сервисов или интерфейса называется Тестированием интерфейса. На самом деле интерфейс — это программное обеспечение, состоящее из наборов команд, сообщений и других атрибутов, которые обеспечивают связь между устройством и пользователем.

Тестирование интерфейса включает в себя тестирование двух основных сегментов:

Интерфейс веб-сервера и сервера приложений

Интерфейс сервера приложений и базы данных

Во время тестирования интерфейса выполняются различные типы тестирования, которые могут включать:

Workflow: он гарантирует, что механизм интерфейса обрабатывает ваши стандартные рабочие процессы как и ожидалось.

Крайние случаи (Edge cases -unexpected values) — непредвиденные значения: это учитывается, когда тестирование включает дату, месяц и день в обратном порядке.

Тестирование производительности, нагрузки и сети (Performance, load, and network testing): интерфейс с большим объемом может потребовать больше нагрузочного тестирования, чем интерфейс с низким объемом, в зависимости от механизма интерфейса и инфраструктуры подключения

Отдельные системы (Individual systems): это включает в себя тестирование каждой системы в отдельности. Например, биллинговая система и система управления запасами для розничного магазина должны работать отдельно.

**35. Что такое тестирование рабочего процесса/воркфлоу? (Workflow testing)**

Это тип тестирования программного обеспечения, который проверяет, что каждый software workflow точно отражает данный бизнес-процесс. Workflow — это серия задач для получения желаемого результата, которая обычно включает несколько этапов или шагов. Для любого бизнес-процесса тестирование этих последовательных шагов определяется как «WorkFlow testing».

Например, убедитесь, что система может быть установлена на платформе пользователя и работает правильно. Тестирование рабочего процесса проводится поэтапно. Вот как вы будете выполнять Workflow testing:

Начальная фаза (Inception phase): эта фаза включает начальное планирование испытаний и тестирование прототипа

Фаза разработки (Elaboration phase): Эта фаза включает базовую архитектуру тестирования

Фаза строительства (Construction phase): эта фаза включает в себя значительные испытания в каждой сборке

Фаза перехода (Transition phase): Эта фаза включает в себя регрессионные тесты и повторные тесты исправлений

Тестирование workflow выполняется:

Test engineer: планирует цели теста и график. Определяет Test case и процедуры. Оценивает результаты теста.

Component engineer: Разработка тестовых компонентов. Автоматизирует некоторые тестовые процедуры.

Integration Tester: Выполнение интеграционных тестов и выявление дефектов

System Testers: Выполнение системных тестов и отчеты о дефектах.

**36. Что вы знаете о пользовательском приемочном тестировании? (UAT – User Acceptance testing)**

Пользовательское приемочное тестирование (UAT) — это тип тестирования, выполняемый конечным пользователем или клиентом для проверки / принятия ПО перед его перемещением в production. UAT выполняется на заключительном этапе тестирования после выполнения функциональных, интеграционных и системных испытаний. Основной целью UAT является проверка end-to-end business flow. Он не фокусируется на косметических ошибках, орфографических ошибках или тестировании системы. Приемочное тестирование пользователя выполняется в отдельной среде тестирования с настройкой данных, аналогичных производственным. Это своего рода тестирование черного ящика, в котором будут участвовать два или более конечных пользователя. Этапы:

Анализ бизнес-требований

Создать плана тестирования UAT

Определить Test Scenario

Создать Test case UAT

Подготовить Test Data (Production like Data)

Запустить Test case

Записать результаты

Подтвердить бизнес-цели

**37. Что такое эксплуатационное приемочное тестирование? (OAT — Operational Acceptance testing)**

ИСПЫТАНИЕ НА ЭКСПЛУАТАЦИЮ (OAT) — это тип тестирования программного обеспечения, который оценивает операционную готовность программного приложения до его выпуска в производство. Целью эксплуатационного тестирования является обеспечение бесперебойной работы системы в ее стандартной операционной среде (SOE - standard operating environment). Это также называется Оперативное тестирование (Operational testing). Эксплуатационное приемочное тестирование обеспечивает соответствие системы и компонентов в стандартной операционной среде приложения (SOE). Типы OAT:

Installation testing

Load & Performance Test Operation

Backup and Restore testing

Security testing

Code Analysis

Fail over testing

Recovery testing

End-to-End Test Environment Operational testing

Operational Documentation Review

Примеры Test case:

Резервные копии, сделанные на одном сайте, могут быть развернуты на тот же сайт

Резервные копии, сделанные на одном сайте, можно развернуть на другом сайте.

Внедрение любых новых функций в живую производственную среду не должно отрицательно влиять на целостность текущих производственных услуг.

Процесс внедрения может быть воспроизведен с использованием действующей документации

Каждый компонент может быть отключен и успешно запущен в согласованные сроки.

Для оповещений — все критические оповещения должны идти в TEC и ссылаться на документ правильного разрешения.

Оповещения созданы и выдаются при превышении согласованных пороговых значений

Любая документация по восстановлению, созданная или измененная, включая сервисные диаграммы, действительна

Это должно быть передано в соответствующие области поддержки.

Любой компонент, на который влияет сбой, должен показывать рекомендуемый порядок перезапуска, время завершения и т. д.

**38. Что такое инсталляционное тестирование?**

Тестирование инсталляции (установки) направленно на проверку успешной инсталляции и настройки, а также обновления или удаления ПО, как десктопного, так и мобильного.

Тестирование инсталляции в большинстве своем не входит в Веб-тестирование, являясь специализированным тестированием установки приложений на различные операционные системы.

Следующие проверки должны быть выполнены для этапов:

Установка.

Установка должна начаться при клике по кнопке, подтверждающей данное действие

Установки во всех поддерживаемых окружениях и на всех поддерживаемых платформах

Установки в неподдерживаемых окружениях, а также в нужных окружениях с некорректными настройками

Права, которые требует инсталляция (чаще всего они должны быть админскими), проверить установить приложение как гость

Установки в clean state (при отсутствии любых возможных связанных файлов и предыдущих версий)

Подсчитывается ли при установке количество свободного места на диске и выдается ли предупреждение если места недостаточно

Установки загруженного ранее приложения, а также прямая установка с использованием сети/беспроводного соединения

Восстановится ли процесс установки при внезапном его прерывании (отключение устройства, отказ сети, отключение беспроводного соединения)

Установка приложения, его запуск, удаление приложения должны возвращать систему в исходное состояние

Распознается ли наличие в системе приложений/программ, необходимых для корректной работы устанавливаемого приложения

Повторный запуск установки приложения при уже текущем должен выдавать корректное сообщение, двойная установка должна быть исключена

Процесс установки может быть настраиваемый/дефолтный. Убедиться, что оба корректно работают

Наличие кнопки, которая предложит сохранить приложение в определенную папку, а также указывает дефолтное местоположение («C:/programs/.»)

Правильно ли установлены, сохранены ли в корректных папках файлы приложения

Наличие созданных ярлыков, корректно ли они расположены

После установки в системной вкладке « Программы и компоненты» должны быть доступны: название приложения, иконка, имя издателя, размер приложения, дата установки и номер версии

Настройки переменных сред PATH

Убедиться, что лицензионный ключ сохраняется в Windows Registry library

Поддерживает ли приложение функции 'UnInstall', 'Modify', 'ReInstall' и корректно ли они работают

Работа приложения с уже существующими DLL-файлами, с DLL-файлами приложений, которые необходимы для корректной работы устанавливаемого приложения

Наличие информации/сообщение о том, когда истекает срок действия установленной пробной версии приложения

Обновление:

Поддерживает ли приложение функцию обновления/автообновления

При попытке установить ранее установленную версию приложения система должна ее распознать и выдать корректное сообщение

Сохраняются ли пользовательские настройки при попытке загрузить новую версию/обновить старую версию

При попытке обновить версию должны быть доступны функции удалить приложение и восстановить приложение

Стандартные проверки как при первичной установке приложения

Убедиться, что номер версии приложения сменился новым

Запустить приложение и убедиться, что оно работает корректно

Откат до предыдущей версии:

Попробовать установить старую версию на более новую

Наличие корректного сообщения при попытке отката

Убедиться, что приложение работает корректно

Удаление приложения:

Не остается ли в системе никаких папок/файлов/ярлыков/ключей реестра после полного удаления приложения

Корректно ли работает система после установки и последующего удаления приложения.

**39. Что вы знаете о тестировании безопасности?**

Это тип тестирования ПО, который выявляет уязвимости, угрозы и риски. Целью тестов безопасности является выявление всех возможных лазеек и слабых мест в ПО, которые могут привести к потере информации, доходов, репутации компании, сотрудников или клиентов. Общая стратегия безопасности основывается на трех основных принципах:

Конфиденциальность — сокрытие определенных ресурсов или информации

Целостность - ресурс может быть изменен только в соответствии с полномочиями пользователя

Доступность — ресурсы должны быть доступны только авторизованному пользователю, внутреннему объекту или устройству

Тестирование безопасности обычно выполняет отдельный специалист по безопасности. В ходе тестирования безопасности испытатель играет роль взломщика. Ему разрешено все:

попытки узнать пароль с помощью внешних средств;

атака системы с помощью специальных утилит, анализирующих защиты;

подавление, ошеломление системы (в надежде, что она откажется обслуживать других клиентов);

целенаправленное введение ошибок в надежде проникнуть в систему в ходе восстановления;

просмотр несекретных данных в надежде найти ключ для входа в систему.

При неограниченном времени и ресурсах хорошее тестирование безопасности взломает любую систему. Задача проектировщика системы — сделать цену проникновения более высокой, чем цена получаемой в результате информации.

Типы тестирования безопасности:

Сканирование уязвимостей/оценка защищенности (Vulnerability Scanning) выполняется с помощью автоматизированного ПО для сканирования системы на наличие известных сигнатур уязвимостей.

Сканирование безопасности (Security Scanning) включает в себя выявление слабых сторон сети и системы, а затем предоставляет решения для снижения этих рисков. Это сканирование может быть выполнено как ручным, так и автоматизированным.

Тестирование на проникновение (Penetration testing) — этот тип тестирования имитирует атаку злоумышленника. Это тестирование включает анализ конкретной системы для проверки потенциальных уязвимостей при попытке внешнего взлома.

Оценка рисков (Risk Assessment) тестирование включает анализ рисков безопасности, наблюдаемых в организации. Риски классифицируются как Низкие, Средние и Высокие. Это тестирование рекомендует меры по снижению риска.

Аудит безопасности (Security Auditing) - внутренняя проверка приложений и операционных систем на наличие уязвимостей. Аудит также может быть выполнен путем построчной проверки кода

Этический взлом (Ethical hacking) — совершается с целью выявления проблем безопасности в системе. Это делается White Hat хакерами - это специалисты по безопасности, которые использует свои навыки законным способом для помощи в выявлении дефектов системы, в отличии от Black Hat (преступников) или Gray Hat (что-то между).

Оценка состояния (Posture Assessment) объединяет сканирование безопасности, этический взлом и оценки рисков, чтобы показать общее состояние безопасности организации.

**40. Что такое конфигурационное тестирование?**

Конфигурационное тестирование (Configuration testing) — специальный вид тестирования, направленный на проверку работы ПО при различных аппаратных и программных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т. д. )

В зависимости от типа проекта конфигурационное тестирование может иметь разные цели:

Проект по профилированию работы системы Цель Тестирования: определить оптимальную конфигурацию оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы.

Проект по миграции системы с одной платформы на другую Цель Тестирования: Проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, операционными системами и программными продуктами третьих фирм.

Для клиент-серверных приложений конфигурационное тестирование можно условно разделить на два уровня (для некоторых типов приложений может быть актуален только один):

Серверный

Клиентский

На первом (серверном) уровне, тестируется взаимодействие выпускаемого ПО с окружением, в которое оно будет установлено:

Аппаратные средства (тип и количество процессоров, объем памяти, характеристики сети / сетевых адаптеров и т. д.)

Программные средства (ОС, драйвера и библиотеки, стороннее ПО, влияющее на работу приложения и т. д.)

Основной упор здесь делается на тестирование с целью определения оптимальной конфигурации оборудования, удовлетворяющего требуемым характеристикам качества (эффективность, портативность, удобство сопровождения, надежность).

На следующем (клиентском) уровне, ПО тестируется с позиции его конечного пользователя и конфигурации его рабочей станции. На этом этапе будут протестированы следующие характеристики: удобство использования, функциональность. Для этого необходимо будет провести ряд тестов с различными конфигурациями рабочих станций:

Тип, версия и битность операционной системы (подобный вид тестирования называется кроссплатформенное тестирование)

Тип и версия Web барузера, в случае если тестируется Web приложение (подобный вид тестирования называется кросс-браузерное тестирование)

Тип и модель видео адаптера (при тестировании игр это очень важно)

Работа приложения при различных разрешениях экрана

Версии драйверов, библиотек и т. д. (для JAVA приложений версия JAVA машины очень важна, тоже можно сказать и для .NET приложений касательно версии .NET библиотеки)

и т. д.

Перед началом проведения конфигурационного тестирования рекомендуется:

создавать матрицу покрытия (матрица покрытия - это таблица, в которую заносят все возможные конфигурации),

проводить приоритезацию конфигураций (на практике, скорее всего, все желаемые конфигурации проверить не получится),

шаг за шагом, в соответствии с расставленными приоритетами, проверять каждую конфигурацию.

Уже на начальном этапе становится очевидно, что чем больше требований к работе приложения при различных конфигурациях рабочих станций, тем больше тестов нам необходимо будет провести. В связи с этим, рекомендуем, по возможности, автоматизировать этот процесс, так как именно при конфигурационном тестировании автоматизация реально помогает сэкономить время и ресурсы. Конечно же автоматизированное тестирование не является панацеей, но в данном случае оно окажется очень эффективным помощником.

В итоге: конфигурационным называется тестирование совместимости выпускаемого продукта (ПО) с различным аппаратным и программным средствами.

Основные цели — определение оптимальной конфигурации и проверка совместимости приложения с требуемым окружением (оборудованием, ОС и т. д.). Автоматизация конфигурационного тестирования позволяет избежать лишних расходов

Примечание: в ISTQB вообще не говорится о таком виде тестирования как конфигурационное. Согласно глоссарию, данный вид тестирования рассматривается там как тестирование портируемости: configuration testing: See portability testing. portability testing: The process of testing to determine the portability of a software product.

**41. Объясните, что такое тестирование N+1**

Вариант регрессионного тестирования представлен как N+1. В этом методе тестирование выполняется в несколько циклов, в которых ошибки, обнаруженные в тестовом цикле «N», устраняются и повторно тестируются в тестовом цикле N + 1. Цикл повторяется, пока не будет найдено ни одной ошибки.

**42. Что вы знаете о тестировании сборки? (Build Verification Test)**

Тестирование, направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования. По своим целям является аналогом Дымового Тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

**43. Что такое тестирование потоков? (Thread testing**)

Тестирование потоков определяется как тип тестирования программного обеспечения, который проверяет основные функциональные возможности конкретной задачи (потока). Обычно проводится на ранней стадии фазы интеграционного тестирования. Тестирование на основе потоков является одной из дополнительных стратегий, принятых в ходе тестирования системной интеграции. Поэтому его, вероятно, следует более правильно назвать «тестом взаимодействия потоков» (thread interaction test).

Тестирование на основе потоков подразделяется на две категории:

Однопоточное тестирование включает одну транзакцию приложения за раз

Многопоточное тестирование включает одновременно несколько активных транзакций

Как производить:

Тестирование на основе потоков является обобщенной формой тестирования на основе сеансов (session-based testing), в котором сеансы являются формой потока, но поток не обязательно является сеансом.

Для тестирования потока, поток или программа (небольшая функциональность) интегрируются и тестируются постепенно как подсистема, а затем выполняются для всей системы.

На самом низком уровне оно предоставляет интеграторам лучшее представление о том, что тестировать.

Вместо непосредственного тестирования программных компонентов требуется, чтобы интеграторы сосредоточились на тестировании логических путей выполнения в контексте всей системы.

**44. Что вы знаете о мутационном тестировании? (Mutation testing)**

Mutation testing — это тип тестирования программного обеспечения, в котором мы мутируем (меняем) определенные выражения в исходном коде и проверяем, способны ли Test case найти ошибки. Это тип тестирования белого ящика, который в основном используется для модульного тестирования. Изменения в мутантной программе сохраняются крайне небольшими, поэтому это не влияет на общую цель программы. Цель Mutation testing - оценить качество Test case, которые должны быть достаточно надежными, чтобы не выполнять мутантный код. Этот метод также называется стратегией тестирования на основе ошибок, так как он включает в себя создание ошибки в программе.

Шаг 1: Ошибки вводятся в исходный код программы путем создания множества версий, называемых мутантами. Каждый мутант должен содержать одну ошибку, и цель состоит в том, чтобы заставить версию мутанта потерпеть неудачу, что демонстрирует эффективность Test case.

Шаг 2: Test case применяются к исходной программе, а также к программе мутанта.

Шаг 3: Сравните результаты оригинальной и мутантной программы.

Шаг 4: Если исходная программа и программы-мутанты генерируют разные выходные данные, то этот мутант уничтожается by the Test case. Следовательно, Test case достаточно хорош, чтобы обнаружить изменение между оригинальной и мутантной программой.

Шаг 5: Если исходная программа и программа-мутант генерируют одинаковые выходные данные, мутант остается в живых. В таких случаях необходимо создать более эффективные Test case, которые убивают всех мутантов.

Что изменить в программе мутантов? Есть несколько методов, которые могут быть использованы для создания мутантных программ:

Операторы замены операндов (Operand replacement operators) - например, в условии if (x> y) поменять местами значения x и y

Операторы модификации выражений (Expression Modification Operators) - например, в условии if (х == у) Мы можем заменить == на >=

Операторы модификации операторов (Statement modification Operators) - например, удалить часть else в конструкции if-else или удалить целиком конструкцию if-else, чтобы проверить, как ведет себя программа

Оценка мутации = (убитые мутанты / общее количество мутантов) \* 100

**45. В чем разница между Baseline и Benchmark testing?**

Baseline предназначено для оценки производительности приложения. Benchmark сравнивает производительность приложения с отраслевым стандартом.

Baseline тестирование использует данные, собранные для повышения производительности. Benchmark возвращает информацию о целевом приложении по сравнению с другими приложениями.

Baseline тестирование сравнивает текущую производительность с предыдущей производительностью приложения, тогда как Benchmark сравнивает производительность нашего приложения с производительностью конкурентов.

**46. Что такое A/B тестирование?**

A / B-тестирование также называется сплит-тестированием (split). При тестировании AB мы создаем и анализируем два варианта приложения, чтобы найти, какой вариант работает лучше с точки зрения пользовательского опыта, потенциальных клиентов, конверсий или любой другой цели, а затем в конечном итоге сохранить наиболее эффективный вариант.

Давайте попробуем понять это на примере. Предположим, у нас есть интернет магазин и каталог отображается определенным образом. В какой-то момент (новые маркетинговые исследования/пожелания клиента и т. д.) решено изменить дизайн выдачи товаров в каталоге. Независимо от того, сколько проведено анализа, выпуск нового пользовательского интерфейса будет большим изменением и может иметь неприятные последствия.

В этом случае мы можем использовать A / B-тестирование. Мы создадим интерфейс нового варианта и выпустим его для некоторого процента пользователей. Например — мы можем распределить пользователей в соотношении 50:50 или 80:20 между двумя вариантами - A и B. После этого в течение определенного периода времени мы будем наблюдать эффективность обоих вариантов. Таким образом, тестирование A/B помогает принять решение о выборе лучшего варианта.

**47. Что означает сквозное тестирование? (E2E — End–to–End)**

Сквозное тестирование — это стратегия тестирования для выполнения тестов, которые охватывают все возможные потоки приложения от его начала до конца; проверяет программную систему вместе с ее интеграцией с внешними интерфейсами. Целью сквозного тестирования является создание полного производственного сценария, выявление программных зависимостей и утверждение, что между различными программными модулями и подсистемами передается правильный ввод. Сквозное тестирование обычно выполняется после функционального и системного тестирования. Оно использует реальные данные, такие как данные и тестовая среда, для имитации настроек в реальном времени. Сквозное тестирование также называется цепным тестированием (Chain testing).

Для чего оно нужно? Современные программные системы являются сложными и взаимосвязаны с несколькими подсистемами. Подсистема может отличаться от текущей системы или может принадлежать другой организации. Если какая-либо из подсистем выйдет из строя, вся система программного обеспечения может рухнуть. Это серьезный риск, и его можно избежать путем сквозного тестирования.

**48. В чем разница между E2E и системным тестированием?**

End to End testing Проверяет программную систему, а также взаимосвязанные подсистемы

System testing Проверяет только программную систему в соответствии со спецификациями требований.

Проверяет весь E2E flow

System testing Проверяет функциональные возможности и функции системы.Все интерфейсы, бэкэнд-системы

Функциональное и нефункциональное тестирование

Выполняется после завершения System testing Выполняется после завершения Integration testing Сквозное тестирование включает проверку внешних интерфейсов, которые могут быть сложными для автоматизации. Следовательно, ручное тестирование является предпочтительным.Как ручное, так и автоматическое могут быть выполнены для тестирования системы

**49. Что такое параллельное тестирование? (Parallel testing)**

Это тип тестирования ПО, который одновременно проверяет несколько приложений или подкомпонентов одного приложения, чтобы сократить время выполнения теста. При параллельном тестировании тестер запускает две разные версии программного обеспечения одновременно с одним и тем же вводом. Цель состоит в том, чтобы выяснить, ведут ли себя прежняя система и новая система одинаково или по-разному. Это гарантирует, что новая система достаточно способна для эффективной работы программного обеспечения.

Пример: когда какая-либо организация переходит от старой системы к новой, legacy является важной частью. Передача этих данных является сложным процессом. При тестировании программного обеспечения проверка совместимости вновь разработанной системы со старой системой осуществляется посредством «параллельного тестирования».

Это Parallel testing Это НЕ Parallel testing

Тестирование обновленного приложения по сравнению с предыдущим приложением.

Запуск старого сценария с новым программным обеспечением с зарезервированными условиями ввода.

Цель состоит в том, чтобы узнать, соответствует ли результат предыдущей системе.

Должен иметь знания о старой и недавно разработанной системе

Тестирование только одного ПО

Кросс-браузерное или кроссплатформенное тестирование.

Цель состоит в том, чтобы выяснить проблему проектирования.

Знать разницу не обязательно.

**50. Чем AdHock тестирование отличается от Исследовательского тестирования?**

Часто его путают с другим видом тестирования «Exploratory testing» - «Исследовательское тестирование».

Свободное тестирование (ad-hoc testing) - это вид тестирования, который выполняется без подготовки к тестированию продукта, без определения ожидаемых результатов, проектирования тестовых сценариев. Это неформальное, импровизационное тестирование. Оно не требует никакой документации, планирования, процессов, которых следует придерживаться при выполнении тестирования. Такой способ тестирования в большинстве случаев дает большее количество заведенных отчётов об ошибке. Это обусловлено тем, что тестировщик на первых шагах приступает к тестированию основной функциональной части продукта и выполняет как позитивные, так и негативные варианты возможных сценариев.

Чаще всего такое тестирование выполняется, когда владелец продукта не обладает конкретными целями, проектной документацией и ранее поставленными задачами. При этом тестировщик полагается на свое общее представление о продукте, сравнение с похожими продуктами, собственный опыт. Однако при тестировании ad-hoc имеет смысл владеть общей информацией о продукте, особенно если проект очень сложный и большой. Поэтому нужно хорошее представление о целях проекта, его назначении и основных функциях, и возможностях. А дальше уже можно приступать к ad-hoc тестированию.

Виды свободного тестирования (ad-hoc testing):

Buddy testing - процесс, когда 2 человека, как правило разработчик и тестировщик, работают параллельно и находят дефекты в одном и том же модуле тестируемого продукта. Такой вид тестирования помогает тестировщику выполнять необходимые проверки, а разработчику исправлять множество дефектов на ранних этапах.

Pair testing - процесс, когда 2 тестировщика проверяют один модуль и помогают друг другу. К примеру, один может искать дефекты, а второй их документировать. Таким образом, у одного тестера будет функция, скажем так, обнаружителя, у другого - описателя.

Monkey testing - произвольное тестирование продукта с целью как можно быстрее, используя различные вариации входных данных, нарушить работу программы или вызвать ее остановку (простыми словами - сломать).

Различия между Buddy testing и Pair testing:

Buddy testing (Совместное тестирование) - это сочетание модульного тестирования и системного тестирования между разработчиком и тестировщиком.

Pair testing (Парное тестирование) - выполняется только тестировщиками с разным уровнем знаний и опыта (такое сочетание поможет поделиться взглядами и идеями).

Основные преимущества ad-hoc testing:

нет необходимости тратить время на подготовку документации;

самые важные дефекты зачастую обнаруживаются на ранних этапах;

часто применяется, когда берут нового сотрудника. С помощью этого метода, человек усваивает за 3 дня то, что, разбираясь тестовыми случаями, разбирал бы неделю - это называется форсированное обучение новых сотрудников;

возможность найти трудновоспроизводимые и трудноуловимые дефекты, которые невозможно было бы найти, используя стандартные сценарии проверок.

Если нам нужно провести ad-hoc тестирование интернет-магазина, то этот краткий список может помочь с тем, что нужно проверить:

все возможности сайта доступны без регистрации;

корректность отображения анимаций и картинок;

все возможности сайта доступны после регистрации;

процесс регистрации;

процесс добавления/удаления из корзины;

процесс оплаты покупок;

удобство в пользовании для новичков, простота, подсказки, помощь.