1. **Среда тестирования**

**Среда тестирования** — это настройка ПО для групп тестирования для выполнения тестовых случаев.  Тестовая среда настраивается в соответствии с требованиями тестируемого приложения. Настройка правильной среды тестирования гарантирует успех тестирования программного обеспечения. Любые недостатки в этом процессе могут привести к дополнительным затратам и времени для клиента.

В настройке тестовой среды учавствуют:

* Системные администраторы,
* Разработчики
* Тестеры
* Иногда пользователи или технари со сродством к тестированию.

Каждый тест не может быть выполнен на локальной машине. Возможно, потребуется установить тестовый сервер, который может поддерживать приложения.

Область для настройки тестовой среды включает в себя:

* система и приложения;
* тестовые данные;
* сервер базы данных;
* фронтальная рабочая среда;
* клиентская операционная система;
* браузер;
* аппаратное обеспечение включает операционную систему сервера;
* сеть;
* необходимая документация, такая как справочные документы / руководства по конфигурации / руководства по установке / руководства пользователя.

**2. Понятия build от release**

**Главное отличие** между сборкой и выпуском в тестировании программного обеспечения является то, что **Build – это версия программного обеспечения, которую команда разработчиков передает команде тестирования для целей тестирования, а Release – это программное обеспечение, которое команда тестирования передает клиенту.**

**release**

**build**

*Рисунок 1 – Наглядное изображение этапов ПО*

После разработки программного модуля разработчики преобразуют исходные коды в автономную форму или исполняемый код. После чего команда разработчиков передает сборку команде тестирования для выполнения тестирования. После чего сборка находится в стадии тестирования.

Команда тестирования программного обеспечения проверяет эту сборку. Если он состоит из нескольких ошибок и не соответствует требованиям, команда тестирования программного обеспечения отклоняет эту сборку. Сборки происходят до релиза, и они генерируются чаще.

Релиз является окончательным приложением после завершения разработки и тестирования. После тестирования сборки группа тестирования сертифицирует это программное обеспечение и доставляет его заказчику. Для одного выпуска возможно иметь несколько сборок.

Таким образом, это программное обеспечение доставляется заказчику после завершения этапов разработки и тестирования. Более того, релиз основан на сборках и может иметь несколько сборок.

Также, сктоит отметить, что под сборкой понимается автономный программный артефакт, сгенерированный после преобразования исходного кода в исполняемый код, который можно запустить на компьютере. Релиз, с другой стороны, является распространением окончательной версии приложения. Таким образом, эти определения объясняют принципиальную разницу между сборкой и выпуском.

Различие между сборкой и выпуском заключается в том, что сборка происходит часто, а выпуск – время от времени.

**Вывод:** можно сказать, что существует четкое различие между сборкой и выпуском, хотя эти два слова часто используются взаимозаменяемо. Основное различие между Build и Release в тестировании ПО заключается в том, что Build – это версия программного обеспечения, которую команда разработчиков передает команде тестирования для целей тестирования, а Release – это программное обеспечение, которое команда тестирования передает клиенту.

**3. Основные фазы тестирования**

Необходимость проведения тестирования в несколько этапов определяется тем, что разные типы ошибок проще всего обнаруживаются на разных этапах. Основными фазами тестирования являются:

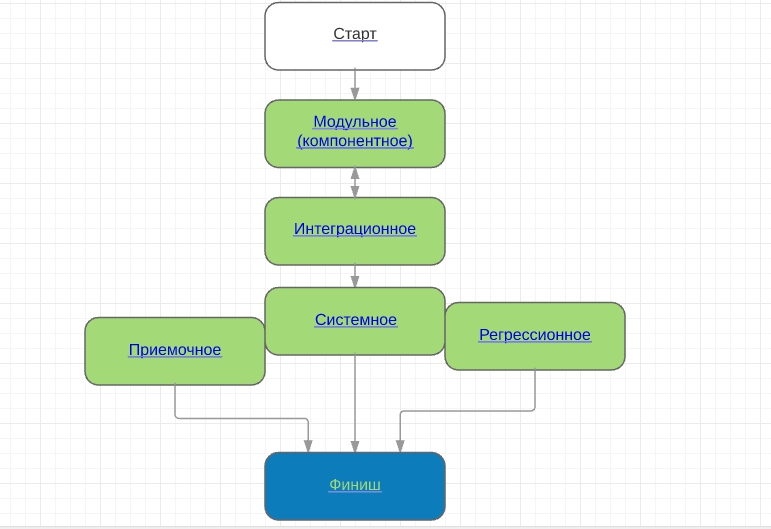
– модульное тестирование;

– интеграционное тестирование;

системное тестирование (в том числе адаптационное, функциональное и нагрузочное);

– регрессионное тестирование;

– приемочное тестирование.



Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. На стадии модульного тестирования достаточно просто выявляются просчеты в логике построения модуля.

Следующим этапом явлется 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 интерпредтации данных, использовании ресурсов, средств коммуникации.

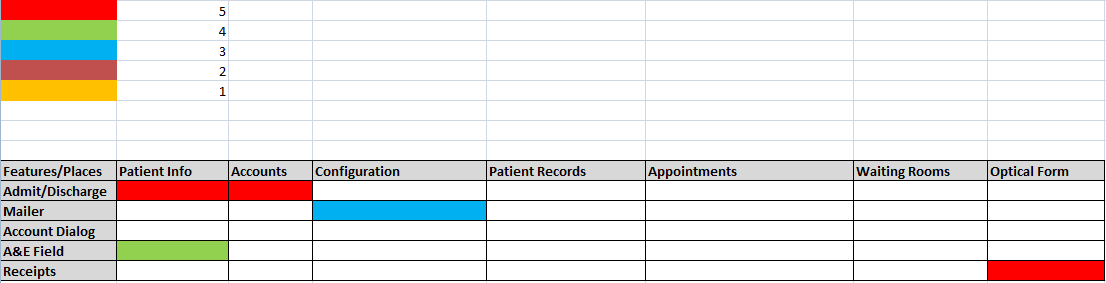
Третьим этапом является системное тестирование, которое существенным образом отличается от модульного и интеграционного. Оно рассматривает систему как черный ящик и работает исключительно с пользовательскими интерфейсами. Основной целью системного тестирования является проверка стабильности и корректности 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.

**4. Импакт анализ**

Импакт анализ — еще один инструмент в разработке ПО, который применяется программистами улучшения тестирования. Основная идея импакт анализа — это указание затронутых мест в проекте при разработке новой или изменении старой функциональности. Во время разработки часто возникают ситуации, когда вносишь изменения в одном месте, а это влияет либо на весь проект либо на несколько мест в проекте. При таких изменениях сложно вспомнить и протестировать все необходимые места, особенно если они не покрыты тестами. В результате этого, после сдачи очередного релиза, клиенты начинают обращаться с жалобами на то, что что-то поломалось или не работает.



В самом начале виден разноцветный список приоритетов — важность или сложность (кому как удобно) того, что будет тестироваться. По вертикали вписываем краткие названия функциональностей, так чтобы было понятно всем о чем идет речь. По горизонтали выписываем те места в проекте, на которые повлияли изменения в функциональности.

Далее начинаем красить таблицу, желательно заранее договориться от критериях важности, чтобы не возникало вопросов по поводу поставленного цвета, а еще лучше проставлять цвета вместе с тем кто разрабатывал и тем кто будет тестировать. Т.к. программист не знает, сколько времени уйдет у тестера на проверку измений, а тестер не знает, сколько программисту понадобится времени на исправление ошибки.

Заполнение таблицы нужно делать с программистом, т.к. тестировщику не известно на какие вкладки могло повлиять то или иное изменение. Программист может по коду просмотреть все места использования измененного/внедренного им метода или класса.

Плюс импакт анализа в том, что вернувшись к тестированию функциональности через длительный промежуток времени, или отдав тестирование в руки нового человека, стоит просто взглянуть на таблицу в которой будут расставлены приоритеты. И уже не нужно будет тратить и без того драгоценное время на воспоминания о том, где нужно тестировать и куда смотреть в первую очередь.

Еще Excel позволяет ставить комментарий на ячейку, с указанием логина того, кто поставил. В комментарий можно дописать либо номер тикета, либо пожелание по тестированию или вставить сслыку на полезный в этом месте документ.

**5. Критерии выбора тестов**

Требования к идеальному критерию тестирования:

Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.

1. Критерий должен быть полным, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.
2. Критерий должен быть надежным, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы.
3. 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.

Классы критериев:

* Структурные критерии используют информацию о структуре программы (критерии так называемого "белого ящика").
* Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого "черного ящика").
* Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы.
* Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

**6. Принципы тестирования**

1. Тестирование демонстрирует наличие дефектов
2. Исчерпывающее тестирование недостижимо
3. Раннее тестирование
4. Скопление/кластеризация дефектов
5. Парадокс пестицида
6. Тестирование зависит от контекста
7. Заблуждение об отсутствии ошибок
8. Garbage in, garbage out (GIGO)

**Принцип 1. Тестирование показывает наличие дефектов**

Тестирование может показать, что дефекты присутствуют, но не может доказать, что дефектов нет. Сколько бы успешных тестов вы не провели, вы не можете утверждать, что нет таких тестов, которые не нашли бы ошибку. Но если мы нашли хотя бы один дефект, мы уже можем утверждать, что в данном ПО присутствуют дефекты.

**Принцип 2. Исчерпывающее тестирование невозможно**

Вместо попыток «протестировать все» нам нужен некий подход к тестированию (стратегия), который обеспечит правильный объем тестирования для данного проекта, данных заказчиков (и других заинтересованных лиц) и данного продукта.

При определении, какой объем тестирования достаточен, необходимо учитывать уровень риска, включая технические риски и риски, связанные с бизнесом, и такие ограничения проекта как время и бюджет. Оценка и управление рисками – одна из наиболее важных активностей в любом проекте.

**Принцип 3. Раннее тестирование**

Тестовые активности должны начинаться как можно раньше в цикле разработки и быть сфокусированы на определенных целях. Этот принцип связан с понятием «цена дефекта» (cost of defect). Цена дефекта существенно растет на протяжении жизненного цикла разработки ПО. Чем раньше обнаружен дефект, тем быстрее, проще и дешевле его исправить. Дефект, найденный в требованиях, обходится дешевле всего.

Еще одно важное преимущество раннего тестирования – экономия времени. Тестовые активности могут начинаться еще до того, как написана первая строчка кода. По мере того, как готовятся требования и спецификации, тестировщики могут приступать к разработке и ревью тест-кейсов. И когда появится первая тестовая версия, можно будет сразу приступать к выполнению тестов.

**Принцип 4. Скопление дефектов**

Небольшое количество модулей содержит большинство дефектов, обнаруженных на этапе предрелизного тестирования, или же демонстрируют наибольшее количество отказов на этапе эксплуатации.  
Многие тестировщики наблюдали такой эффект – дефекты «кучкуются». Это может происходить потому, что определенная область кода особенно сложна и запутана, или потому, что внесение изменений производит «эффект домино». Это знание часто используется для оценки рисков при планировании тестов – тестировщики фокусируются на известных «проблемных зонах».

Также полезно проводить анализ первопричин (root cause analysis), чтобы предотвратить повторное появление дефектов, обнаружить причины возникновения скоплений дефектов и спрогнозировать потенциальные скопления дефектов в будущем.

**Принцип 5. Парадокс пестицида**

Если повторять те же тесты снова и снова, в какой-то момент этот набор тестов перестанет выявлять новые дефекты. Повторное применение тех же тестов и тех же методик приводит к тому, что в продукте остаются именно те дефекты, против которых эти тесты и эти методики неэффективны.  
Чтобы преодолеть «парадокс пестицидов», необходимо регулярно пересматривать существующие тест-кейсы и создавать новые, разнообразные тесты, которые будут выполняться на различных частях системы.

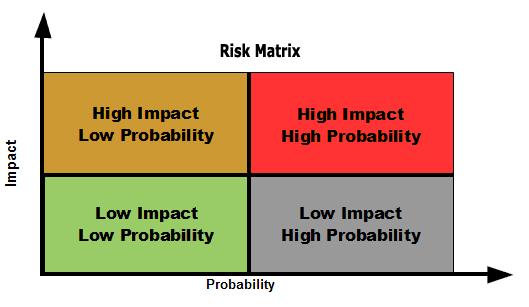
**Принцип 6. Тестирование зависит от контекста**

Тестирование выполняется по-разному, в зависимости от контекста. Например, тестирование систем, критических с точки зрения безопасности, проводится иначе, чем тестирование сайта интернет-магазина.

Этот принцип тесно связан с понятием риска. Что такое риск? Риск – это потенциальная проблема. У риска есть вероятность (likelihood) – она всегда выше 0 и ниже 100% – и есть влияние (impact) – те негативные последствия, которых мы опасаемся. Анализируя риски, мы всегда взвешиваем эти два аспекта: вероятность и влияние.

То же можно сказать и о мире ПО: разные системы связаны с различными уровнями риска, влияние того или иного дефекта также сильно варьируется. Одни проблемы довольно тривиальны, другие могут дорого обойтись и привести к большим потерям денег, времени, деловой репутации, а в некоторых случаях даже привести к травмам и смерти.

Уровень риска влияет на выбор методологий, техник и типов тестирования.



**Принцип 7. Заблуждение об отсутствии ошибок**

Нахождение и исправление дефектов бесполезно, если построенная система неудобна для использования и не соответствует нуждам и ожиданиям пользователей.

Заказчики ПО – люди и организации, которые покупают и используют его, чтобы выполнять свои повседневные задачи – на самом деле совершенно не интересуются дефектами и их количеством, кроме тех случаев, когда они непосредственно сталкиваются с нестабильностью продукта. Им также неинтересно, насколько ПО соответствует формальным требованиям, которые были задокументированы. Пользователи ПО более заинтересованы в том, чтобы оно помогало им эффективно выполнять задачи. ПО должно отвечать их потребностям, и именно с этой точки зрения они его оценивают.

Даже если вы выполнили все тесты и ошибок не обнаружили, это еще не гарантия того, что ПО будет соответствовать нуждам и ожиданиям пользователей.

Иначе говоря, верификация != валидация.

**\* Принцип 8. GIGO.**

В компьютерной науке «garbage in – garbage out» (GIGO) — это концепция, в которой ошибочные или бессмысленные входные данные создают бессмысленный вывод или «мусор», т.е. при неверных входящих данных будут получены неверные результаты, даже если сам по себе алгоритм правилен.

1. **Независимое тестирование**

Независимое тестирование соответствует независимой команде, которая участвует в тестировании, отличном от разработчика, чтобы избежать предвзятости автора и часто более эффективна в поиске дефектов и сбоев.

## Уровни независимого тестирования

В следующем списке показаны возрастающие уровни независимости для тестирования:

* тестирование сделано самим разработчиком;
* независимые тестеры передаются команде разработчиков;
* независимая команда тестирования в организации;
* независимые тестеры разных организаций;
* аутсорсинг членов команды тестирования другой организации.

1. **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 (перемещается влево на временной шкале проекта). Это первая половина изречения «Проверяйте рано и часто». Он был придуман Ларри Смитом в 2001 году.

Тестирование со сдвигом влево важно, поскольку оно помогает предотвратить следующие виды вреда из-за позднего тестирования:

* Тестировщики могут меньше участвовать в первоначальном планировании, что часто приводит к выделению недостаточных ресурсов на тестирование.
* Дефекты в требованиях, архитектуре и дизайне остаются необнаруженными, а на их реализацию тратятся значительные усилия.
* Отладка (включая выявление, локализацию, исправление и регрессионное тестирование дефектов) становится сложнее по мере создания и интеграции большего количества программного обеспечения.
* Инкапсуляция затрудняет тестирование методом белого ящика, уменьшая покрытие кода во время тестирования. У вас меньше времени на исправление дефектов, обнаруженных при тестировании, что увеличивает вероятность того, что они будут отложены до более поздних инкрементов или версий системы. Это создает «изгибающуюся волну» технического долга, который может поглотить проекты, если он станет слишком большим.

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

Каждый этап испытаний имеет свое назначение. Проще управлять поэтапно. Мы можем запустить разные тесты в разных средах. Производительность и качество тестирования улучшаются с помощью поэтапного тестирования.

**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.