# 1. qqqqqqq

6 апреля 2023 г.

Гаврилов А. Г. Новоселова О. В.

Современные технологии и средства разработки программного обеспечения (второй семестр )
Раздел 7. Основы тестирования программных продуктов

## Тема 20
## Тестирование: основныепонятия и определения, виды ошибок, стратегии и уровни тестирования

## Содержание



## 2. Жизненный цикл тестирования

*Тестирование программного обеспечения* — процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.

Рассмотрим жизненный цикл тестирования (рис. **??**). Приведенная схема – не догма, и вы легко можете найти альтернативы, но общая суть и ключе-вые принципы остаются неизменными.

**Стадия 1** (общее планирование и анализ требований) объективно необходима как минимум для того, чтобы иметь ответ на такие вопросы,как: что нам пред-стоит тестировать; как много будет работы; какие есть сложности; всё ли необходимое у нас есть и т.п. Как правило, получить ответы на эти вопросы невозможно без анализа требований, т.к. именно требования являются первичным источником ответов.

**Стадия 2**(уточнение критериев приёмки) позволяет сформулировать или уточнить метрики и признаки возможности или необходимости начала тестирования, приостановки и возобновления тестирования, завершения или прекращения тестирования.

**Стадия 3**(уточнение стратегии тестирования) представляет собой ещё одно обращение к планированию, но уже на локальном уровне: рассматриваются и уточняются те части стратегии тестирования, которые актуальны для те-кущей итерации.

**Стадия 4**(разработка тест-кейсов) посвящена разработке, пересмотру, уточнению, доработке, переработке и прочим действиям с тест-кейсами, наборами тест-кейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.

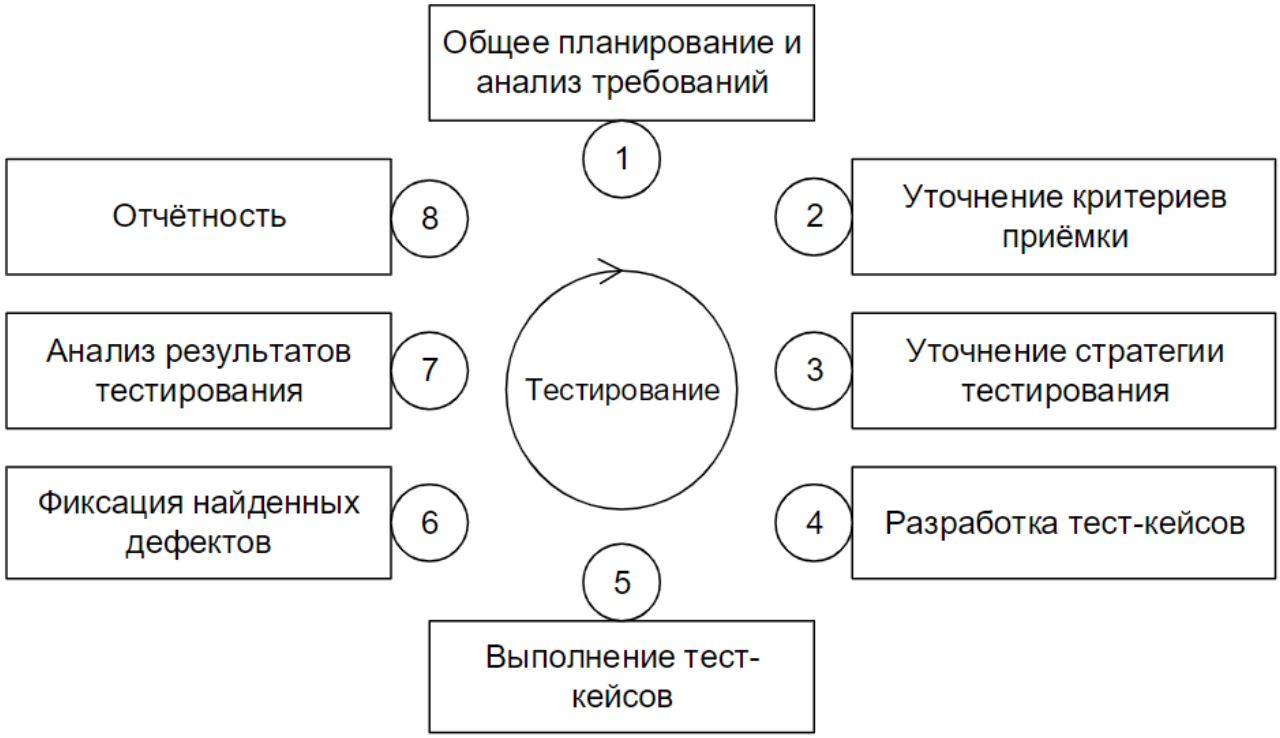


Рис. 1. Жизненный цикл тестирования

**Стадия 5**(выполнение тест-кейсов) и **стадия 6** (фиксация найденных дефектов) тесно связаны между собой и фактически выполняются параллельно: дефекты фиксируются сразу по факту их обнаружения в процессе выполнения тест-кейсов. Однако зачастую после выполнения всех тест-кейсови написания всех отчётов о найденных дефектах проводится явно выделенная стадия уточнения, на которой все отчёты о дефектах рассматриваются повторно с целью формирования единого понимания проблемы и уточнения таких характеристик дефекта,как важность и срочность.

**Стадия 7**(анализ результатов тестирования) и **стадия 8** (отчётность) также тесно связаны между собой и выполняются практически параллельно. Формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приёмки и уточнённой стратегии, полученных на стадиях 1, 2 и 3. Полученные выводы оформляются на стадии 8 и служат основой для стадий 1, 2 и 3 следующей итерации тестирования. Таким образом, цикл замыкается.

# 3. Основные принципы тестирования

**Тестирование показывает наличие дефектов, а не их отсутствие**

Очень сложно обнаружить нечто, относительно чего мы не знаем — ни «где оно», ни «как оно выглядит», ни даже «существует ли оно вообще». Это отчасти напоминает попытки«вспомнить, не забыл ли я что-то».

**Исчерпывающее тестирование невозможно**

Исчерпывающее тестирование в теории призвано проверить приложение «со всеми возможными комбинациями всех возможных входных данных во всех возможных условиях выполнения». Но это невозможно физически.

**Тестирование тем эффективнее, чем раньше оно выполняется**

Этот принцип призывает не откладывать тестирование «на потом» и «на последний мо-

мент». Конечно, чрезмерно раннее тестирование может оказаться не-эффективным и даже привести к необходимости повторно выполнять большой объём работы, но начатое вовремя(без промедления) тестирование даёт наибольший эффект.

**Кластеризация дефектов**

Дефекты не возникают «просто так». И уже тем более «просто так» не появляется много дефектов в какой-то «проблемной» области приложения. «Группировка» дефектов по какому-то явному признаку является хорошим поводом к продолжению исследования данной области программного продукта: скорее всего, именно здесь будет обнаружено ещё больше дефектов.

**Парадокс пестицида**

Название данного принципа происходит от общеизвестного явления в сельском хозяйстве: если долго распылять один и тот же пестицид на посевы, у насекомых вскоре вырабатывается иммунитет, что делает пестицид неэффективным.

То же самое верно и для тестирования программного обеспечения, где парадокс пестицида проявляется в повторении одних и тех же (или просто однотипных) проверок снова и снова: со временем эти проверки перестанут обнаруживать новые дефекты.

**Тестирование зависит от контекста**

Набор характеристик программного продукта влияет на глубину тестирования, используемый набор техник и инструментов, принципы организации работы тестировщиков и т.д. Основная идея данного принципа состоит в том, что невозможно выработать некий «универсальный подход к тестированию» на все случаи жизни, и даже просто бездумное копирование подходов к тестированию с одних проектов на другие часто не заканчивается ничем хорошим.

**Отсутствие дефектов — не самоцель**

Программный продукт должен не только быть избавлен от дефектов настолько, насколько это возможно, но и удовлетворять требованиям заказчика и конечных пользователей—в противном случае он станет непригодным для использования. Если объединить этот принцип с предыдущим, получается: именно понимание контекста продукта и потребностей пользователей позволяет тестировщикам выбрать наилучшую стратегиюи добиться наилучшего результата.

# 4. Виды тестирования (классификация)

## 4.1. Классификация по запуску кода на исполнение

Далеко не всякое тестирование предполагает взаимодействие с работающим приложением. Потому в рамках данной классификации выделяют:

- **Статическое тестирование** — тестирование без запуска кода на исполнение.В рамках этого подхода тестированию могут подвергаться:
  - Документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.).
  - Графические прототипы (например, эскизы пользовательского интерфейса).
  - Код приложения (что часто выполняется самими программистами в рамках аудита кода (codereview), являющегося специфической вариацией взаимного просмотра в применении к исходному коду).
  - Код приложения также можно проверять с использованием техник тестирования на основе структур кода.
  - Параметры (настройки) среды исполнения приложения.
  - Подготовленные тестовые данные

- **Динамическое тестирование** — тестирование с запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование), так и код нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное или компонентное тестирование) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

## 4.2. Классификация по доступу к коду и архитектуре приложения

- **Метод белого ящика** — у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного. Выделяют даже сопутствующую тестированию по методу белого ящика глобальную технику — тестирование на основе дизайна (design-basedtesting). Для более глубокого изучения сути метода белого ящика рекомендуется ознакомиться с техниками исследования потока управления или потока данных, использования диаграмм состояний. Некоторые авторы склонны жёстко связывать этот метод со статическим тестированием, но ничто не мешает тестировщику запустить код на выполнение и при этом периодически обращаться к самому коду(а модульное тестирование и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком»).

- **Метод чёрного ящика** — у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования. Тестировщик оказывает на приложение воздействия (и проверяет реакцию) тем же способом, каким при реальной эксплуатации приложения на него воздействовали бы пользователи или другие приложения. В рамках тестирования по методу чёрного ящика основной информацией для создания тест-кейсов выступает документация(особенно — требования) и общий здравый смысл (для случаев, когда поведение приложения в некоторой ситуации не регламентировано явно; иногда это называют «тестированием на основе не-явных требований», но канонического определения у этого подхода нет).

- **Метод серого ящика** — комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет. Крайне редкий случай: обычно говорят о методах белого или чёрного ящика в применении к тем или иным частям приложения, при этом понимая, что «приложение целиком» тестируется по методу серого ящика.

## 4.3. Классификация по уровню детализации приложения(по уровню тестирования)

- **Модульное (компонентное) тестирование** направлено на проверку отдельных небольших частей приложения, которые (как правило) можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы, взаимодействие классов, небольшие библиотеки, отдельные части приложения. Часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

  Из-за особенностей перевода на русский язык теряются нюансы степени детализации: «юнит-тестирование», как правило, направлено на тестирование атомарных участков кода, «модульное» — на тестирование классов и небольших библиотек, «компонентное» —

на тестирование библиотек и структурных частей приложения. Но эта классификация не стандартизирована, и у различных авторов можно встретить совершенно разные взаимоисключающие трактовки.

- **Интеграционное тестирование** направлено на проверку взаимодействия между несколькими частями приложения(каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования).К сожалению, даже если мы работаем с очень качественными отдельными компонентами, «на стыке» их взаимодействия часто возникают проблемы. Именно эти проблемы и выявляет интеграционное тестирование.

- **Системное тестирование** направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях. Здесь не только выявляются дефекты «на стыках» компонентов, но и появляется возможность полноценно взаимодействовать с приложением с точки зрения конечного пользователя, применяя множество других видов тестирования.

Выделенные уровни схематично представлены на рисунке **??**.

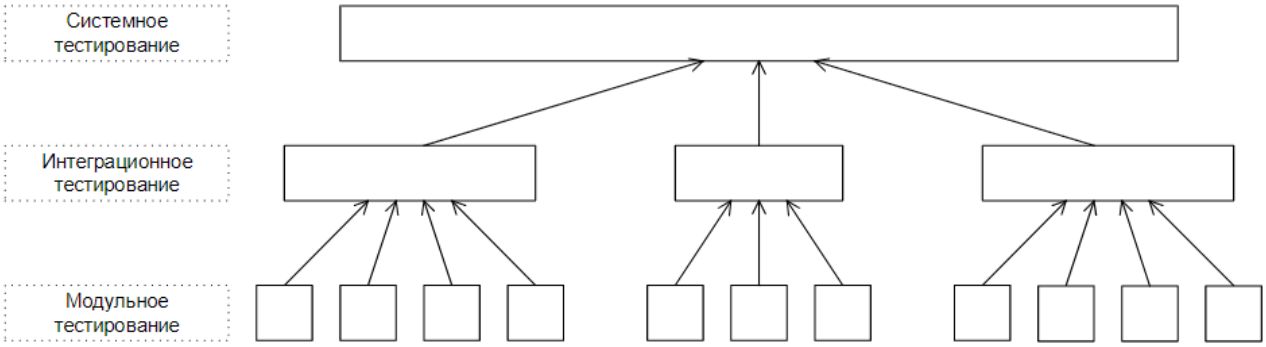


Рис. 2. Схематичное представление классификации тестирования по уровню детализации приложения

## 4.4. Классификация по привлечению конечных пользователей

- **Альфа-тестирование** выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей. Может являться формой внутреннего приёмочного тестирования. В некоторых источниках отмечается, что это тестирование должно проводиться без привлечения команды разработчиков, но другие источники не выдвигают такого требования. Суть этого вида вкратце: продукт уже можно периодически показывать внешним пользователям, но он ещё достаточно «сырой», потому основное тестирование выполняется организацией-разработчиком.

- **Бета-тестирование** — выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования. Суть этого вида вкратце: продукт уже можно открыто показывать внешним пользователям, он уже достаточно стабилен, но проблемы всё ещё могут быть, и для их выявления нужна обратная связь от реальных пользователей.

- **Гамма-тестирование** — финальная стадия тестирования перед выпуском продукта, направленная на исправление незначительных дефектов, обнаруженных в бета-тестировании. Как правило, также выполняется с максимальным привлечением конечных пользователей/заказчиков. Может являться формой внешнего приёмочного тестирования. Суть этого

вида вкратце: продукт уже почти готов, и сейчас обратная связь от реальных пользователей используется для устранения последних недоработок.

С учётом предыдущей классификации полный цикл тестирования программного продукта можно сформулировать в виде схемы, представленной на рисунке **??**.


Рис. 3. Полный вариант классификации тестирования по уровню

# Список литературы

[1] Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — 3-е изд. — Минск: Четыре четверти, 2020. — 312 с. ISBN 978-985-581-362-1.