

[HOME](#)[ABOUT ME](#)

IT Blog by Ilarion Halushka

Sharing thoughts and experience

[<- BACK TO THE LIST OF ARTICLES](#)

Теория тестирования от А до Я.

🕒 19 mins

[best](#)[testing](#)[qa](#)[interview](#)[Теорія тестування українською UA](#)

Вопросы на собеседованиях Trainee/Junior/Middle Manual QA в среднем на 50% состоят из теории тестирования.

Навигация: [↔](#)

1. [Тестирование. Качество ПО](#)
2. [Валидация vs Верификация](#)
3. [Цели тестирования](#)
4. [Этапы тестирования](#)
5. [Тест-план](#)
6. [Тест-дизайн](#)
7. [Техники тест-дизайна](#)

8. Продвинутые техники тест-дизайна



9. Бонусные и Авторские Техники тест-дизайна

10. Exploratory vs Ad-hoc testing

11. Test Case (Тестовый случай)

12. Check-list (Чек-лист)

13. Bug Report (Баг-репорт)

14. Severity vs Priority

15. Traceability Matrix (Матрица соответствия требований)

16. Defect / Error / Bug / Failure

17. Уровни тестирования (Levels of Testing)

18. Виды / Типы тестирования (Testing Types)

19. Принципы тестирования (Principles of Testing)

20. Статическое и Динамическое тестирование

21. Требования (Requirements)

22. Жизненный цикл бага

23. Жизненный цикл разработки ПО

24. Методологии разработки

1. Тестирование. Качество ПО

Тестирование

— проверка соответствия между реальным и ожидаемым поведением.

Тестирование

— это одна из техник контроля качества, включающая в себя активности по:

- Test Management (планированию работ)
- Test Design (проектирование тестов)
- Test Execution (выполнение тестов)
- Test Analysis (анализ результатов тестирования)

Качество ПО (Software Quality)

— это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

2. Валидация vs Верификация

Верификация (verification)

— оценка соответствия продукта требованиям (спецификации).

Отвечает на вопрос: “Система работает в соответствии с требованиями?”

Валидация (validation)

— оценка соответствия продукта ожиданиям и требованиям пользователей.

Отвечает на вопрос: “Требования удовлетворяют ожидания пользователя?”

3. Цели тестирования

1. Повысить вероятность того, что приложение:
 - будет соответствовать всем описанным требованиям.
 - будет работать правильно при любых обстоятельствах.
2. Предоставление актуальной информации о состоянии продукта на данный момент.

4. Этапы тестирования

1. Анализ продукта
2. Работа с требованиями
3. Разработка тест плана
4. Создание тестовой документации
5. Тестирование
6. Отчет о тестировании (test report)
7. Стабилизация
8. Эксплуатация

5. Тест план

Test Plan — это документ, описывающий весь объем работ по тестированию

Отвечает на вопросы:

- Что?
- Когда?
- Критерии начала/окончания тестирования.
- Окружение (environment) dev/staging/production?
- Подходы/техники/инструменты/виды тестирования?
- Браузеры/версии/OS/разрешения экрана?
- Кто? Обязанности? Ресурсы? Обучение?
- Сроки?
- График?
- Стратегия тестирования.
- Ссылки на документацию.
- Ссылки на требования.

6. Тест дизайн

Test design — это этап процесса тестирования ПО, на котором проектируются и создаются тест кейсы, в соответствии с критериями качества и целями тестирования.

- **Тест аналитик** — определяет «ЧТО тестировать?»
- **Тест дизайнер** — определяет «КАК тестировать?»
- **Реальность** — все делает 1 человек :)

7. Техники тест дизайна

Эквивалентное Разделение (Equivalence Partitioning)

- Как пример, у вас есть диапазон допустимых значений от 1.00 до 10.00 долларов, вы должны выбрать одно любое верное значение внутри интервала, скажем, 5.00, и любые неверные значения вне интервала, например 0.99 и 11.00.

test design equivalence

Анализ Граничных Значений (Boundary Value Analysis)

- Как пример, у вас есть диапазон допустимых значений от 1.00 до 10.00 долларов.
- **Two value (двузначный) BVA**: валидные граничные значения 1.00, 10.00, и невалидные

значения 0.99 и 10.01.

- **Three/Full value (трехзначный) BVA:** валидные граничные значения 1.00, 1.01, 10.00, 9.99, и невалидные значения 0.99 и 10.01.

test design BVA

Причина / Следствие (Cause/Effect)

- ввод комбинаций условий (причин), для получения ответа от системы (следствие).
- Например, вы проверяете возможность добавлять клиента:
- **Причина:** необходимо заполнить поля «Имя», «Адрес», «Номер Телефона» и нажать кнопку «Добавить».
- **Следствие:** После нажатия на кнопку «Добавить», система добавляет клиента в базу данных и показывает его номер на экране.

Предугадывание ошибки (Error Guessing)

- использование знаний системы и способность к интерпретации спецификации (требований) на предмет того, чтобы «предугадать» при каких входных условиях система может выдать ошибку.

Например, спецификация говорит: «пользователь должен ввести код».

Тестировщик будет думать: «Что, если я не введу код?», «Что, если я введу неправильный код?»...

test design error guessing

8. Продвинутые техники тест дизайна

Удиви интервьюера. Вааааау...

Попарное тестирование (Pairwise Testing)

- Формирование таких наборов тестовых данных, в которых каждое тестируемое значение каждого из проверяемых параметров хотя бы единожды сочетается с каждым тестируемым значением всех остальных проверяемых параметров.

Звучит сложно, но на практике использовать эту технику очень просто и логично.

- Суть техники — мы не проверяем все сочетания всех значений, но проверяем ВСЕ ПАРЫ значений.

test design pair wise

Таблица принятия решений (Decision table)

- В таблицах решений представлен набор условий, одновременное выполнение которых должно привести к определенному действию/решению.

test design decision table

Диаграмма (граф) состояний-переходов (State Transition diagram)



- диаграмма для описания поведения системы.
- Система имеет конечное число состояний и переходов между состояниями.
- Диаграмма может быть переведена в Таблицу состояний-переходов (или в таблицу принятия решений).

State transition diagram

Use case (пользовательский сценарий)

Это сценарий взаимодействия пользователя с системой для достижения определенной цели.

Use case содержит:

- кто использует систему (например роль админ/покупатель/продавец).
- что пользователь хочет сделать.
- цели пользователя.
- шаги, которые выполняет пользователь.
- описание того, как система реагирует на действия пользователя.

9. Бонусные и авторские техники тест дизайна



Просвети интервьюера. Открой ему глаза.

Семи-Исчерпывающее тестирование (Semi-Exhaustive Testing)

- проверка всех возможных комбинаций входных значений. Как правило, на практике применение техники Exhaustive Testing не представляется возможным. (см. принцип тестирования №2 Исчерпывающее тестирование недостижимо (Exhaustive testing is impossible))

Иногда на практике встречаются случаи, когда стандартные техники не дают достаточного уровня уверенности в работоспособности системы. Например, в системах связанных с медициной или авиа сферами, **иногда** стоит применять Semi-Exhaustive Testing.

Не забываем про принцип тестирования №6 Тестирование зависит от контекста (Testing is context dependent). Думаем головой, когда уместно применение этой техники, а когда нет.

Блок-схема (block scheme/diagram)

Блок-схему можно использовать как технику тест дизайна, составляя тест-кейсы по логике схемы.

test design block schema

Шляпы / роли

Техника “Шляпы / роли” чем-то схожа с техникой составления тест кейсов по Use Case.

Принцип: одеваем шляпу определенной роли пользователя и представляем себя в его роли.

Пример: “одеваем” шляпу Кастинг Директора и размышляем как новый функционал будет работать для этой роли. Представляем, какие могут быть зависимости и особенности системы для Кастинг Директора. Размышляем, какие бизнес цели преследует Кастинг директор в нашей системе и как поведение системы может отличаться от других ролей. Потом “одеваем” шляпу Актера, Агента, Админа...

test design hats roles

Техники тест дизайна, о которых пока нигде не слышал:

Каждый имеет право придумать свою технику тест дизайна. Тестирование – это не бездумное применение всем известных техник. © Илларион

О техниках “Разговорчики-driven”, “Analytics-driven”, “Bug-driven” я пока нигде не слышал.

⚠ Интервьюеры могут быть отличниками, которые ограничиваются только книжными понятиями и не выходят за рамки (thinking out of the box). Поэтому будьте аккуратны с озвучиванием этих техник интервьюеру, особенно, если у вас проблемы с объяснением и примерами)) Не ограничивайте себя существующими техниками, думайте, фантазируйте.

Разговорчики-driven (talks-driven)

Собираем в одной комнате/звонке одного или нескольких программистов, менеджеров, клиентов, тестировщиков и тд. И начинаем допрос о конкретной функции или всей системе.

Если фантазия не работает, то задаем Wh-вопросы:

what, when, where, who, whom, which, whose, why and how - что, когда, где, кто, кому, какой, чей, почему, как

Для продвинутых: сначала собираем всех по одному, а потом по несколько человек. Не выпускаем, пока не получим все ответы и не решим какие тесты проектировать.

Аналитика-driven (analytics-driven)

Если на проекте используется аналитика, например при кликах на кнопки или при открытии страниц отправляются ивенты (events) в систему для аналитики, то можно использовать данные аналитики для составления тест кейсов.

Мы знаем куда пользователи чаще всего кликают, на каких страницах проводят больше всего времени. Почему бы основываясь на этих данных не составить тест кейсы?

Баг-driven (bugs-driven)

Принцип тестирования №4 Скопление дефектов (Defects clustering) гласит, что “большая часть дефектов содержится в небольшом количестве модулей”.

Основываясь на найденных ранее багах и на обращениях клиентов в службу поддержки, можно определить “больные” места системы и сконцентрировать тест кейсы на этих модулях системы.

Дополнительно можно посидеть над найденными багами и подумать “а может ли аналогичный баг быть в другой части системы?”.

10. Exploratory vs Ad-hoc testing

Исследовательское тестирование (exploratory testing)

- это одновременное изучение системы, проектирование тестов (тест дизайн) и непосредственно тестирование.
- Данная техника базируется на опыте тестировщика (experience based).
- Пример: приходит тестировщик на новый проект и начинает одновременно изучать сайт, писать чек-лист и проходить этот чек-лист (тестировать).

Ad-hoc тестирование

- Перевод от автора статьи - “тестирование от балды”.

- Вид тестирования, который выполняется без подготовки к тестам, без определения ожидаемых результатов, без проектирования тестовых сценариев.
- Неформальное, импровизационное тестирование.

11. Test Case (тестовый случай)

Test Case

— это тестовый артефакт/документ, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки тестируемой функции.

Test Case

— это описание проверки работы системы, которое может выполнить любой человек из команды.

Test Case

— это описание проверки системы на соответствие требованиям.

Тест кейс состоит из:

- ID (идентификатор)
- Title (название)
- Type (тип)
- Priority (приоритет)
- Preconditions (предусловия)
- Steps (шаги)
- Expected Result (ожидаемый результат)
- Post conditions (пост условия) - например очистка данных или возвращение системы в первоначальное состояние.

Тест кейсы разделяются на позитивные и негативные:

- **Позитивный тест кейс** использует только корректные данные и проверяет, что приложение правильно выполнило вызываемую функцию.
- **Негативный тест кейс** оперирует как корректными, так и некорректными данными (минимум 1 некорректный параметр) и ставит целью проверку исключительных ситуаций (срабатывание валидаторов), а также проверяет, что вызываемая системой функция не выполняется при срабатывании валидатора.

Примеры и лучшие практики создания тест кейсов - 

12. Check-list (Чек-лист)

Check list

— это документ, описывающий, **что** должно быть протестировано.

- Чек-лист может быть абсолютно разного уровня детализации.
- Как правило, чек-лист содержит только действия (шаги) без ожидаемого результата.
- Чек-лист менее формализован чем тест кейс.
- Чек-лист намного легче поддерживать, чем тест кейсы.
- Пункты чек листа отвечают на вопрос “что тестировать?”, а конкретные шаги и детали “как тестировать?” описывают в тест кейсах.

Примеры и лучшие практики создания чек-листов - 

13. Bug report (баг репорт)



Bug Report

— это документ, описывающий последовательность действий, которые привели к некорректной работе системы, с указанием причин и ожидаемого результата.

Основные составляющие Bug report:

- ID (идентификатор)
- Название (Title)
- Короткое описание (Summary)
- Проект (Project)
- Компонент приложения (Component)
- Номер версии (Version)
- Серьезность (Severity)
- Приоритет (Priority)
- Статус (Status)
- Автор (Author)
- Назначен на (Assignee)
- Окружение (Environment: dev/test/staging/prod/etc.)
- App/build version (версия билда/приложения)
- Шаги воспроизведения (Steps to Reproduce)
- Фактический Результат (Actual Result)
- Ожидаемый результат (Expected Result)

Дополнительные составляющие Bug report:

- Screenshots (скриншоты) 
- Video (видео)
- Credentials (login + password)
- Browser console errors (логи с браузера)
- Mobile app logs (логи с мобилки)
- Server logs (логи с сервера)
- API Requests (апи запросы)
- Analytics events (ивенты с аналитики)
- Database data (данные из базы данных)
- Database queries (запросы в базу)
- Date and time (дата и время)
- Comments/Notes (комментарии/заметки)
- Link tasks/bugs (подвязка других задач/багов к текущему)
- HAR archive - архив со всеми запросами в Network 

14. Severity vs Priority

Серьезность (Severity)

— это атрибут, характеризующий влияние дефекта на работоспособность приложения.

В теории Severity выставляется тестировщиком.

Градация Severity:

- S1 Блокирующая (Blocker)
- S2 Критическая (Critical)
- S3 Значительная (Major)
- S4 Незначительная (Minor)
- S5 Тривиальная (Trivial)

Приоритет (Priority)

— это атрибут, указывающий на очередность выполнения задачи или устранения дефекта.

Чем выше приоритет, тем быстрее нужно исправить дефект.


В теории Priority выставляется менеджером, тимлидом или заказчиком.

Градация Priority:

- P1 Высокий (High)
- P2 Средний (Medium)
- P3 Низкий (Low)

Реальность: на всех проектах, где я работал, был только priority :)

Реальность: на разных проектах разные градации.

Пример вопроса на собеседовании про Severity / Priority - 

15. Traceability matrix (Матрица соответствия требований)

Traceability matrix - это двумерная таблица, содержащая соответствие функциональных требований и тест кейсов.

В заголовках колонок таблицы расположены требования, а в заголовках строк — ID тест кейсов.

На пересечении — отметка, означающая, что требование текущей колонки покрыто тестовым сценарием текущей строки.

16. Defect / Error / Bug / Failure

Дефект (он же баг)

— это несоответствие фактического результата ожидаемому результату, описанному в требованиях.

Bug (defect)

— ошибка программиста (или другого члена команды), то есть когда в программе что-то идёт не так как планировалось и программа выходит из-под контроля.

Например, когда никак не контролируется ввод пользователя, в результате неверные данные вызывают краши (crash) или иные «приколы» в работе программы. Либо программа построена так, что изначально не соответствует тому, что от неё ожидается.

Error (ошибка)

— действие, которое привело к неправильному результату.

Пример 1 — ввод букв в поля, где требуется вводить цифры (возраст, количество товара и т.п.). Error: “поле должно содержать только цифры”.

Пример 2 - регистрация с уже существующим в системе емейлом. Error: “этот емейл уже используется”.

Failure

— сбой (причём необязательно аппаратный) в работе компонента, всей программы или системы.


То есть, существуют такие дефекты, которые приводят к сбоям. И существуют такие, которые не приводят. UI-дефекты например. Но аппаратный сбой, никак не связанный с software, тоже является failure.

17. Уровни Тестирования (Levels of testing)

1. Модульное тестирование (Unit Testing)

Тестирование кода классов, функций, модулей в коде. Обычно выполняется программистами.

2. Интеграционное тестирование (Integration Testing)

Тестирование взаимодействия между несколькими классами, функциями, модулями. Например тестирование API через Postman. 

3. Системное тестирование (System Testing)

Проверка как функциональных, так и нефункциональных требований к системе.

4. Приемочное тестирование (Acceptance Testing)

Проверка соответствия системы требованиям и проводится с целью:

- определения удовлетворяет ли система приемочным критериям;
- вынесения решения заказчиком/менеджером принимается приложение или нет.

18. Виды / типы тестирования (Testing types)



18.1. Функциональные виды тестирования

- Функциональное тестирование (Functional testing)
- Тестирование пользовательского интерфейса (GUI Testing)

- Тестирование безопасности (**Security and Access Control Testing**)
- Тестирование взаимодействия (**Interoperability Testing**)

18.2. Нефункциональные виды тестирования

- Все виды тестирования производительности (**Performance**):
 - нагрузочное тестирование (**Load Testing**) - много пользователей.
 - стрессовое тестирование (**Stress Testing**) - очень много данных и/или пользователей (пиковые значения).
 - объемное тестирование (**Volume Testing**) - много данных.
 - тестирование стабильности или надежности (**Stability / Reliability Testing**)
- Тестирование установки (**Installation testing**)
- Тестирование удобства пользования (**Usability Testing**)
- Тестирование на отказ и восстановление (**Failover and Recovery Testing**)
- Конфигурационное тестирование (**Configuration Testing**)

18.3. Связанные с изменениями виды тестирования

- Дымовое тестирование (**Smoke Testing**)
- Регрессионное тестирование (**Regression Testing**)
- Повторное тестирование (**Re-testing**)
- Тестирование сборки (**Build Verification Test**)
- Санитарное тестирование или проверка согласованности/исправности (**Sanity Testing**)

Testing types

19. Принципы тестирования (Principles of testing)



1. Тестирование демонстрирует наличие дефектов (Testing shows presence of defects)

Тестирование может показать, что дефекты присутствуют в системе, но не может доказать, что их нет.

2. Исчерпывающее тестирование недостижимо (Exhaustive testing is impossible)

Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев.

3. Раннее тестирование (Early testing)

Чтобы найти дефекты как можно раньше, активности по тестированию должны быть начаты как можно раньше в жизненном цикле разработки.

4. Скопление дефектов (Defects clustering)

Как правило, большая часть дефектов, обнаруженных при тестировании, содержится в небольшом количестве модулей.

5. Парадокс пестицида (Pesticide paradox)

Если одни и те же тесты будут прогоняться много раз, в конечном счете этот набор тестовых сценариев больше не будет находить новых дефектов.

6. Тестирование зависит от контекста (Testing is context dependent)

Тестирование выполняется по-разному в зависимости от контекста.

7. Заблуждение об отсутствии ошибок (Absence-of-errors fallacy)

Обнаружение и исправление дефектов не помогут, если созданная система не подходит пользователю и не удовлетворяет его ожиданиям и потребностям.

20. Статическое и динамическое тестирование



Статическое (static) тестирование

Производится **БЕЗ** запуска кода программы.

Примеры: тестирование требований/документации, код ревью, статические анализаторы кода.

Динамическое (dynamic) тестирование

Производится **С** запуском кода программы.

21. Требования (requirements)

Требования - это спецификация (описание) того, что должно быть реализовано.

Требования описывают то, что необходимо реализовать, без детализации технической

стороны решения. “Что”, а не “как”.

Требования к требованиям:

1. корректность
2. недвусмысленность
3. полнота
4. непротиворечивость
5. упорядоченность по важности и стабильности
6. проверяемость (тестопригодность)
7. модифицируемость
8. трассируемость
9. понимаемость

22. Жизненный цикл бага

Bug lifecycle


23. Жизненный цикл разработки ПО


Software Development Life Cycle (SDLC):


1. Идея (Idea)
2. Сбор и анализ требований (Planning and Requirement Analysis)
3. Документирование требований (Defining Requirements)
4. Дизайн (Design Architecture)
5. Разработка (Developing)
6. Тестирование (Testing)
7. Внедрение/развертывание (Deployment)
8. Поддержка (Maintenance)
9. Смерть (Death)


24. Методологии разработки

Waterfall 



V-model 

Spiral 

Kanban 

Scrum 

Предложения и пожелания

Всегда рад получать конструктивную критику по контенту лекций и этой статье. Не стесняйтесь  

[Теорія тестування українською UA](#)

Источники: статья на доу <https://dou.ua/forums/topic/13389> www.protesting.ru, bugscatcher.net, qalight.com.ua, thinkingintests.wordpress.com, книга ISTQB, www.quizful.net, bugsclock.blogspot.com, www.zeelabs.com, devopswiki.net, hvorostovoz.blogspot.com.

Share this article:

 [facebook](#)

 [twitter](#)

 [reddit](#)

 [linkedin](#)

 [email](#)

Maintained by [Ilarion Halushka](#)