Geekbrains

Дипломная работа

На тему: «[Организация процесса ручного тестирования веб-приложений на основе исследовательского тестирования в условиях отсутствия технического задания на примере интернет-сайта по изготовлению фотографий.](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D1%82%D0%B5%D0%BC%D0%B0-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0-%D0%BE%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%B0-%D1%80%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE-%D1%82%D0%B5%D1%81%D1%82%D0%)»

Студент: Черленяк Анна Леонидовна

Специальность: Инженер по тестированию

г. Красноярск, 2023г.

**Содержание.**

Введение……………………………………………………………………………………………..1

[Глава 1. Основы ручного тестирования веб-приложений.](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-1-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D1%80%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE-%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%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9).............................................................2

Глава [1.1. Веб-приложение как инструмент развития бизнеса](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#11-%D0%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BA%D0%B0%D0%BA-%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82-%D1%80%D0%B0%D0%B7%D0%B2%D0%B8%D1%82%D0%B8%D1%8F-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81%D0%B0)…………………………………...2

Глава [1.2. Что такое тестирование. Важность ручного тестирования в наше время](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#12-%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%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-%D0%B2%D0%B0%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C-%D1%80%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE-%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%B2-%D0%BD%D0%B0%D1%88%D0%B5-%D0%B2%D1%80%D0%B5%D0%BC%D1%8F10)…………….5

Глава [1.3. Виды тестирования программного продукта.](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#13-%D0%B2%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%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D0%B0-11).................................................................8

[Глава 2. Составление тестовой документации для ручному тестированию веб-приложения](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-3-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%82%D0%B5%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B9-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8-%D0%B4%D0%BB%D1%8F-%D1%80%D1%83%D1%87%D0%BD%D0%BE%D0%BC%D1%83-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8E-%D0%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)…18

Глава [2.1. Составление чек-листов](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#31-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%87%D0%B5%D0%BA-%D0%BB%D0%B8%D1%81%D1%82%D0%BE%D0%B2)………………………………………………………………..18

Глава [2.2. Составление чек-кейсов](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#32-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%87%D0%B5%D0%BA-%D0%BA%D0%B5%D0%B9%D1%81%D0%BE%D0%B2)………………………………………………………………..21

Глава [2.3. Анализ тестирования и тест-дизайн](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#33-%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7-%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%B8-%D1%82%D0%B5%D1%81%D1%82-%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD)…………………………………………………...26

Глава [2.4. Что такое баг-репорт и как его составить](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#34-%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D0%B1%D0%B0%D0%B3-%D1%80%D0%B5%D0%BF%D0%BE%D1%80%D1%82-%D0%B8-%D0%BA%D0%B0%D0%BA-%D0%B5%D0%B3%D0%BE-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%B8%D1%82%D1%8C)……………………………………………..37

[Глава 3. Проведение исследовательского тестирования веб-приложения и составление Технического Задания программного продукта, с учетом ожиданий заказчика(владельца сайта)](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-2-%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%BE%D0%B3%D0%BE-%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%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE-%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D0%B0-%D1%81-%D1%83%D1%87%D0%B5%D1%82%D0%BE%D0%BC-%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B9-%D0%B7%D0%B0%D0%BA%D0%B0%D0%B7%D1%87%D0%B8%D0%BA%D0%B0%D0%B2%D0%BB%D0%B0%D0%B4%D0%B5%D0%BB%D1%8C%D1%86%D0%B0-%D1%81%D0%B0%D0%B9%D1%82%D0%B0)………………………………………………………………………………………….……41

Глава [3.1. Особенности Исследовательского тестирования](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#21-%D0%BE%D1%81%D0%BE%D0%B1%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8-%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%BE%D0%B3%D0%BE-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)……………………………………41

Глава [3.1.1. Идея туров в исследовательском тестировании. Туры Джеймса Виттакера.](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#211-%D0%B8%D0%B4%D0%B5%D1%8F-%D1%82%D1%83%D1%80%D0%BE%D0%B2-%D0%B2-%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%BE%D0%BC-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B8-%D1%82%D1%83%D1%80%D1%8B-%D0%B4%D0%B6%D0%B5%D0%B9%D0%BC%D1%81%D0%B0-%D0%B2%D0%B8%D1%82%D1%82%D0%B0%D0%BA%D0%B5%D1%80%D0%B012)........44

Глава [3.2. Проведение Исследовательского тестирования](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#22-%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%BE%D0%B3%D0%BE-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)……………………………………..53

Глава [3.3. Что такое Техническое задание](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#23-%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5-%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5)……………………………………………………….56

Глава [3.4. Составление Технического Задания](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#24-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE-%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F)…………………………………………………..58

[Глава 4. Проведение тестирования веб-приложения и составление рекомендаций по улучшению продукта.](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-4-%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%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%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%80%D0%B5%D0%BA%D0%BE%D0%BC%D0%B5%D0%BD%D0%B4%D0%B0%D1%86%D0%B8%D0%B9-%D0%BF%D0%BE-%D1%83%D0%BB%D1%83%D1%87%D1%88%D0%B5%D0%BD%D0%B8%D1%8E-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D0%B0).......................................................................................................................62

Глава [4.1. Проведение тестирования на основе составленной тестовой документации](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#41-%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%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%BD%D0%B0-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B5-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9-%D1%82%D0%B5%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B9-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8)………62

Глава [4.2. Подведение итогов. Разработка предложений по улучшению работы программного продукта.](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#42--%D0%BF%D0%BE%D0%B4%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B8%D1%82%D0%BE%D0%B3%D0%BE%D0%B2-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0-%D0%BF%D1%80%D0%B5%D0%B4%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9-%D0%BF%D0%BE-%D1%83%D0%BB%D1%83%D1%87%D1%88%D0%B5%D0%BD%D0%B8%D1%8E-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D0%B0)...........................................................................................................................................63

Заключение………………………………………………………………………………………...65

Список использованной литературы и ресурсов………………………………………………..66

Приложение 1. Матрица туров и типов ошибок.

Приложение 2. Test runs. Исследовательское тестирование.

Приложение 3. Jira\_Баг-репорт. Исследовательское Тестирование.

Приложение 4. Техническое задание.

Приложение 5. Test runs. Тестирование на основе тест-кейсов.

Приложение 6. Jira\_Баг-репорт. Тестирование на основе тест-кейсов.

**Введение.**

В настоящее время, с развитием информационных технологий и программного обеспечения, все большую актуальность приобретает веб-разработка. Сайты и веб-приложения становятся неотъемлемой частью жизни современного общества, служа связующим звеном между пользователями и информацией. При этом, критическое значение приобретает качество и надежность разработанных веб-приложений.

Одним из ключевых методов обеспечения качества программного обеспечения является тестирование. Среди различных подходов к тестированию веб-приложений, значительное внимание привлекает исследовательское тестирование. Этот метод предполагает привлечение опытных тестировщиков, способных выявить скрытые ошибки и уязвимости в приложении путем осмысленного исследования его функциональности.

Однако, часто при разработке веб-приложений отсутствует полное и четкое техническое задание, что усложняет проведение тестирования. В таких условиях необходимо организовать процесс ручного тестирования, учитывая особенности и требования исследовательского подхода.

Целью данной дипломной работы является разработка методического подхода к организации процесса ручного тестирования веб-приложений в условиях отсутствия технического задания на примере интернет-сайта. Для достижения этой цели необходимо решить следующие задачи: проанализировать особенности исследовательского тестирования и его применимость в рамках ручного тестирования, изучить технологические и методические основы организации процесса тестирования веб-приложений, разработать модель организации тестирования на основе исследовательского подхода, а также провести экспериментальное исследование на примере реального интернет-сайта.

Ожидается, что результаты данной работы позволят оптимизировать процесс ручного тестирования веб-приложений в условиях отсутствия технического задания, далее его (техническое задание) разработать и провести подробное тестирование интернет-сайта уже с учетом четких требований и ожиданий владельца продукта (заказчика).

[**Глава 1. Основы ручного тестирования веб-приложений.**](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-1-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D1%80%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE-%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%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9)

**Глава** [**1.1. Веб-приложение как инструмент развития бизнеса**](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#11-%D0%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BA%D0%B0%D0%BA-%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82-%D1%80%D0%B0%D0%B7%D0%B2%D0%B8%D1%82%D0%B8%D1%8F-%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81%D0%B0)**.**

Ещё 15–20 лет назад нельзя было предположить, что веб-приложения станут неотъемлемой частью жизни. Сегодня эта разновидность онлайн-инструментов используется для различных задач, включая оптимизацию бизнес-процессов, продажу товаров и услуг, распространение информации, общение пользователей друг с другом.

Веб-приложения — незаменимый инструмент маркетинга, который использует все большее количество компаний. Они решают огромное количество сложных задач, помогая автоматизировать ваш бизнес, и упрощают жизнь сотрудникам.

В отличие от стандартных приложений, речь идет о программах, которые способны работать полноценно даже без установки на устройство. Смартфон, планшет или компьютер получает онлайн-доступ к данным, а пользователю не нужно проводить установку в постоянную память — это и есть главная отличительная черта веб-приложений.

Веб-приложением называют клиент-серверное приложение, которое находится на удаленном сервере. Но в то же время пользовательский интерфейс сервиса отображается в браузере. Именно поэтому в структуре выделяют следующие компоненты:

● клиентская часть — отвечает за действия, выполняемые пользователем;

● серверная часть — отвечает за процессы, происходящие на сервере;

● база данных — структура для упорядоченного хранения информации и доступа к ней.

В зависимости от того, какие задачи ставят перед собой создатели проекта, они используют те или иные средства разработки веб-приложений. Главная задача — обеспечить функциональное взаимодействие между клиентской и серверной частью, доступ к базе данных, корректные возможности по формированию и отправке готовых страниц в ответ на запрос.

Исходя из поставленных задач, разработчики могут создать веб-приложение, к которому удастся получить доступ с любого устройства, или же требовательную среду разработки, для работы с которой подойдут только устройства с определенным уровнем аппаратных возможностей. Могут применяться различные методы разработки веб-приложений, в том числе с открытым доступом к архитектуре, как в «Википедии», или с отсутствием такого доступа для посторонних, как в любом коммерческом или новостном приложении.

**Виды веб-приложений.**

1. **SPA (Single Page Application)**– одностраничное интерактивное приложение. Важно, что оно не только находится на одной странице, но и, подобно полноценной программе, является интерактивным. Так информационный веб-сайт может состоять из одной страницы, но, по существу, не являться SPA. В одностраничном приложении пользователь, переключаясь между вкладками, остается на одной странице. Причем подгружаются и обновляются только нужные части контента, что играет в пользу быстродействия SPA.

Пример одностраничной программы – Gmail. Обратите внимание – когда вы переключаетесь между списками сообщений, адрес страницы не меняется. Это отличительный признак SPA.

2. **MPA (Multi Page Application)** – традиционные многостраничные веб-приложения. Когда пользователь взаимодействует с веб-сайтом, загружаются новые HTTP-страницы. Поэтому обмен данными происходит медленней, чем в SPA. Особенно, если есть проблемы с интернет-соединением или с хостингом веб сайта.

Примеры MPA – интернет-магазины, такие как ОЗОН и Wildberries.

3. **PWA**. Прогрессивное веб-приложение. Разрабатывается с помощью специальных технологий и способно работать без подключения к интернету. Такие приложения доступны на всех устройствах и адаптируются под — любую ширину экрана телефона или компьютера. Их можно устанавливать на главный экран своего смартфона, отправлять уведомления и пользоваться им, как полноценным приложением.

В случае с каждым проектом создания веб-приложения совершается определенный цикл действий, в котором можно выделить следующие этапы:

1. Сбор требований и разработка ТЗ. Заказчик озвучивает как основные задачи, так и более глобальные цели, а также дополняет это своими требованиями, чтобы была возможность ознакомить разработчиков с заданием.

2. Прототипирование. Исполнитель создаёт прототип будущего проекта, где отражены будущие блоки и показано, как они будут взаимодействовать в web-среде. Важно выбрать надёжного и опытного исполнителя. Также на данном этапе определяются необходимые технологии разработки.

3. Создание дизайна. Разработчики создают макет внешнего вида, чтобы согласовать его с заказчиком вслед за функциональным прототипом.

4. Верстка и разработка. Теперь команда приступает к созданию страниц в том виде, в каком они должны быть. Здесь происходит два отдельных процесса: с точки зрения backend важно согласовать выполнение функций, а с точки зрения frontend — реакцию визуальных элементов на действия пользователя.

5. Тестирование. Тестировщикам нужно убедиться, что веб-приложение полностью справляется со своими функциями.

6. Документирование. На основе уже готового проекта создается документация, которая будет необходима пользователям, чтобы как можно быстрее освоить всю функциональность проекта.

С точки зрения коммерции, веб-приложения имеют несколько заметных преимуществ:

1. Безопасность. Минимальный доступ к серверным элементам и базам данных. А значит, меньше всего можно опасаться взлома и других негативных последствий.

2. Доступ с разных устройств. Современные движки позволяют получать доступ к веб-приложению параллельно с разных платформ, например, с компьютера или ноутбука на Windows, с мобильных устройств на Android и Apple.

3. Отсутствие клиентского ПО. Не нужно расходовать лишние ресурсы — место и память на установку клиентского ПО на устройство.

4. Масштабируемость — веб-приложение способно справляться с нужным объёмом задач в зависимости от их количества.

Создание веб-приложения — это на сегодня однозначно один из самых прогрессивных путей инвестирования времени и ресурсов в развитие компании, ведь его внедрение позволяет:

○ автоматизировать бизнес-процессы и для сотрудников, и для персонала;

○ привлечь внимание целевой аудитории и выделиться на фоне конкурентов;

○ сделать решение востребованных задач проще, надежнее и безопаснее.

**Глава 1.2. Что такое тестирование. Важность ручного тестирования в наше время.**

Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.

Цель тестирования – проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

Тестирование ПО проводится для:

1. проверки соответствия требованиям;

2. обнаружения проблем на более ранних этапах разработки и предотвращение повышения стоимости продукта;

3. обнаружения вариантов использования, которые не были предусмотрены при разработке, а также взгляд на продукт со стороны пользователя;

4. повышения лояльности к компании и продукту, т.к. любой обнаруженный дефект негативно влияет на доверие пользователей.

**Принципы тестирования.**

Как определить, что во время тестирования вы придерживаетесь правильной стратегии? Нужно следовать семи принципам тестирования, которые определяет стандарт ISTQB. Они оптимизируют количество и качество работ в QA.

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

Тестирование только снижает вероятность наличия дефектов, которые находятся в программном обеспечении, но не гарантирует их отсутствия.

**Принцип 2** – Исчерпывающее тестирование невозможно (Exhaustive testing is impossible).

Полное тестирование с использованием всех входных комбинаций данных, результатов и предусловий физически невыполнимо (исключение — тривиальные случаи).

**Принцип 3** – Раннее тестирование (Early testing).

Следует начинать тестирование на ранних стадиях жизненного цикла разработки ПО, чтобы найти дефекты как можно раньше.

**Принцип 4** – Скопление дефектов (Defects clustering).

Большая часть дефектов находится в ограниченном количестве модулей.

**Принцип 5** – Парадокс пестицида (Pesticide paradox).

Если повторять те же тестовые сценарии снова и снова, в какой-то момент этот набор тестов перестанет выявлять новые дефекты.

**Принцип 6** – Тестирование зависит от контекста (Testing is context depending). Тестирование проводится по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем новостной портал.

**Принцип 7** – Заблуждение об отсутствии ошибок (Absence-of-errors fallacy). Отсутствие найденных дефектов при тестировании не всегда означает готовность продукта к релизу. Система должна быть удобна пользователю в использовании и удовлетворять его ожиданиям и потребностям.

Сегодня есть множество фреймворков для тестирования, поддерживающих практически все существующие языки. Казалось бы — можно брать и автоматизировать. Но даже сейчас ручные тесты важны.

Одно из объяснений их необходимости заключается в том в том, что при ручном тестировании функционала мы можем гораздо быстрее получить информацию о состоянии продукта, который анализируем, о качестве разработки. Кроме того, при автоматизации предварительно разработанные кейсы часто приходится менять и актуализировать, а на написание автотестов требуется определённое время.

При этом в процессе разработки может прийти обратная связь от заказчика, когда он увидит в готовом продукте какую-то функцию, которую решит изменить до релиза — и, если вы уже подготовили для неё программные тесты, их придётся переписать. Обновление кейсов, автотестов и их проверка отнимут ценное время, которое можно было бы использовать на само обновление этой фичи.

Всё это означает, что главная цель ручных тестов — предварительно убедиться в том, что заявленный функционал работоспособен, не имеет ошибок и выдаёт ожидаемые, запланированные результаты. Без них нельзя быть уверенным в том, что можно работать дальше. Особенно это актуально для функций, на реализацию которых завязана последующая разработка. В таком случае возня с созданием автотестов на эти фичи становится блокирующим фактором для всей разработки продукта, сдвигая сроки и срывая дедлайны.

В дополнение к этому, на первых этапах разработки приложения автоматизация может оказаться довольно дорогой. Вам потребуется специалист, обладающий специфической квалификацией (и, возможно, не один). Постоянное поддержание тестов в актуальном состоянии требует затрат ресурсов вплоть до релиза фичи. А месяцы простоя, посвященные вылизыванию автотеста ударят по мотивации команды.

Если вы хотите регулярно добавлять новый функционал и успевать за действиями конкурентов, то перед тем, как создавать автотесты всегда проверяйте возможности продукта вручную. Просто потому что ручное тестирование ускоряет ваши процессы.

Автотесты не позволяют проверить удобно ли использовать новые возможности приложения – проверка юзабилити всегда осуществляется вручную.

В ручных тестах можно импровизировать, создавая безумные сочетания действий, которые никогда не придут в голову пользователю, но могут быть совершены им случайно. Это позволяет создавать новые кейсы.

Новые кейсы появляются еще и потому, что тестировщик постоянно задает себе вопрос «а что, если?». Так он находит оригинальные способы взаимодействия с приложением — даже если их не было в базовых сценариях.

**Глава 1.3. Виды тестирования программного продукта.**

Тестирование программного продукта с точки зрения классификации по целям должно делиться на два класса:

* **Функциональное тестирование**
* **Нефункциональное тестирование**

Под **функциональным тестированием** понимается проверка соответствия программного продукта функциональным требованиям, указанным в техническом задании на создание это продукта. Если говорить проще, то при функциональном тестировании проверяется выполняет ли программный продукт все функции, которые должен.

Итак, Вы, таки решились провести функциональное тестирование. Вы заглядываете в техническое задание, читаете функциональные требования и понимаете, что по крайней мере они расположены не в том порядке, в каком можно производить тестирование. Вы будете удивлены, что еще достаточно давно другие уже заметили это несоответствие и придумали как его преодолеть.

Для проведения функционального тестирования персоналом отдела технического контроля разрабатывается документ программа и методика испытаний функционала приложения (ПМИ). Документ ПМИ содержит перечень сценариев тестирования программного продукта (test cases) с подробным описанием шагов. Каждый шаг сценария тестирования характеризуется действиями пользователя (специалиста по тестированию) и ожидаемыми результатами – ответной реакции программы на эти действия. Программа и методика испытаний обязана имитировать эксплуатацию программного продукта в реальном режиме. Это означает, что сценарий тестирования должен быть построен на основе анализа операций, которые будут выполнять будущие пользователи системы, а не быть искусственно составленной последовательностью понятных только разработчику манипуляций.

**Виды функционального тестирования.**

* **По доступу к внутренней структуре ПО:**

1. Метод черного ящика
2. Метод белого ящика
3. Метод серого ящика

**Чёрный ящик** – это система, внутреннее устройство и механизм работы которой сложны, неизвестны или не важны в рамках решения задачи.

Метод чёрного ящика — метод исследования, при котором вместо свойств и взаимосвязей составных частей системы изучается реакция системы как целого на изменяющиеся условия.

У чёрного ящика есть «вход» для ввода информации и «выход» для отображения результатов работы. При этом происходящие процессы во время работы системы наблюдателю неизвестны. Состояние выходов функционально зависит от состояния входов.

Тестирование чёрного ящика (black box testing) — тестирование, основанное на анализе функциональной или нефункциональной спецификации системы без знания внутренней структуры.

У тестировщика нет доступа к коду, он видит приложение как пользователь. Тестирование проводится через интерфейс приложения. Это ручное тестирование без знания, что находится «за кулисами» интерфейса.

Тестирование чёрного ящика находит ошибки:

1. Функции неправильно реализуются или их нет.

2. Интерфейс отличается от макетов.

3. Данные не записываются в базы или записываются неверно.

4. Недостаточная производительность системы.

Таким образом, тестировщик концентрируется на том, что программа делает, а не как.

Преимущества чёрного ящика:

1. Тестирование производится с позиции пользователя, обнаруживает неточности и противоречия в поведении ПО.

2. Тестировщику не обязательно знать языки программирования.

3. Тестирование проводят независимые специалисты, что помогает избежать предвзятого отношения.

4. Тест-кейсы пишутся, как только готова спецификация.

Недостатки чёрного ящика

1. Тестируется ограниченное количество сценариев.

2. Без чёткой спецификации трудно составить эффективные тест-кейсы.

3. Тесты избыточны, если их уже проверил разработчик на уровне модульного тестирования.

**Метод белого ящика**

Тестирование белого ящика (white box testing) основано на анализе внутренней структуры системы, на знании и понимании исходного кода. У тестировщика есть полный доступ к исходному коду.

Для тестирования методом белого ящика нужно знать язык программирования, на котором написано приложение. Обычно этот вид тестирования применяют разработчики при написании юнит-тестов. Входные значения отбираются на основе кода, который будет их обрабатывать.

Техника белого ящика применяется на разных уровнях тестирования, но главным образом для модульного тестирования компонентов.

Преимущества:

1. Тестирование производится на ранних этапах: пользовательский интерфейс не требуется.

2. Тестирование более тщательное, с покрытием путей выполнения программы (условий и операторов).

Недостатки:

1. Для тестирования нужны специальные знания. В первую очередь — язык программирования.

**Метод серого ящика**

Тестирование серого ящика (gray box testing) — тестирование в условиях, когда часть внутренней структуры программы известна. Тестировщик работает не с кодом приложения, а с часть его внутренней структуры: проверяет запись в базе данных, лог-файлы, коды ответа от сервера.

Для тестирования веб-приложений методом серого ящика тестировщик использует инструменты разработчика, например, Chrome DevTools.

Техника серого ящика применяется на интеграционном уровне для проверки взаимодействия компонентов программы, например, API-интерфейса и базы данных.

Методы тестирования серого ящика:

1. Матричное тестирование — определение всех переменных, которые есть в программе.

2. Тестирование ортогональных массивов обеспечивает максимальное покрытие кода с минимальным количеством тестов.

3. Pattern Testing выполняется на данных истории предыдущих дефектов системы.

* **Связанные с изменениями в коде:**

На протяжении всей разработки в код приложения вносят изменения: при добавлении новых функций и при исправлении дефектов. В результате тестировщик снова проводит тестирование той части приложения, которая уже была проверена, но подверглась изменениям. В зависимости от изменений выделяют регрессионное и повторное тестирование.

1. **Повторное** – исполняются тестовые сценарии, выявившие ошибки во время последнего запуска.

Цель – подтвердить, что ошибки исправили, и приложение работает в соответствии с требованиями.

Повторное тестирование – обязательный этап. Тестировщик должен проверить, исправлен ли дефект, повторив сценарий, который выявил ошибку.

1. **Регрессионное** – тестирование уже проверенной функциональности после изменений в коде.

Цель – убедиться, что эти изменения не добавили или не активизировали ошибки в изменённых областях.

* **По запуску кода на исполнение:**

1. **Статическое** – тестирование системы на уровне спецификации или реализации без исполнения кода.

Статическое тестирование проводится для проверки:

○ документации — требований, схем баз данных, тест-кейсов;

○ кода приложения — проверка кода перед запуском специалистом, не участвовавшем в его написании или изменении, то есть аудит кода, или code review;

○ параметров настройки среды приложения;

○ подготовленных тестовых данных;

○ прототипов пользовательского интерфейса.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и продолжается на протяжении всей разработки.

1. **Динамическое** – тестирование во время выполнения программного обеспечения, компонента или системы. Проверка – реальное поведение ПО во время его работы.

Чтобы выполнить динамическое тестирование нужно, чтобы код программы запустился. Тестируется как система в целом, так и отдельные компоненты. Все виды функционального тестирования – динамические.

* **По характеру сценария:**

Этот подход определяет поведение системы в привычных и экстремальных условиях.

1. Позитивное – оценка ожидаемого поведения. Это тестирование проводится в первую очередь, ведь позволяет определить корректность работы программы.

Тестировщик полностью следует требованиям и инструкции по работе с приложением.

1. Негативное – определение устойчивости системы в нестандартной ситуации. Например, неожиданный сценарий взаимодействия пользователя с интерфейсом.

Эти типы тестирования нередко проводятся параллельно. Ведь работая над некоторой функциональностью, тестировщику проще оценить её поведение и в стандартных, и в нестандартных условиях.

* **По исполнителю:**

От объекта тестирования движемся к его субъекту. Вы могли слышать об альфа- и бета-тестировании. А поучаствовать в одном из них можно, даже не будучи тестировщиком.

Итак, по исполнителю тестирование делится на:

1. **Альфа-тестирование** - проверка программного продукта на поздней стадии разработки. Проводится разработчиками или тестировщиками, иногда — с частичным привлечением пользователей.

Альфа-тестирование проводится после модульного, интеграционного и системного тестирования, когда продукт уже частично готов к выпуску на рынок, но нужно его доработать. Представляет собой имитацию реального использования, но выполняется либо командой тестирования, либо другими сотрудниками компании-разработчика в тестовой среде. Например, на тестовых стендах, недоступных внешним пользователям.

После внутреннего альфа-тестирования выпускают бета-версию продукта и передают её на внешнее (публичное) бета-тестирование.

1. **Бета-тестирование** – оценка ПО перед выходом на рынок в фокус-группе или добровольцами. Отзывы собираются, анализируются и учитываются при внесении правок.

Продукт должен быть стабилен, но не исключено появление проблем и выявление недостатков. Поэтому сначала доступ открывают для небольшой группы лояльных пользователей, чтобы проверить работоспособность и получить обратную связь.

Иногда бета-версия размещается в конкретной стране или регионе, чтобы собрать статистику или получить обратную связь прежде, чем полность вывести продукт на рынок.

* **По наличию тест-кейсов:**

1. **На основе тест-кейсов** – формализованный подход, при котором тестирование проводится на основе заранее подготовленных тест-кейсов. Часто используется на проектах по разработке ПО, так как позволяет структурировать процесс тестирования и сделать его контролируемым.
2. **Исследовательское тестирование** - применяется в быстрорастущих проектах с гибкими методиками разработки, где нет четкой документации и требований.

Исследовательское тестирование - высший пилотаж в тестирование программного обеспечения. Качественное тестирование доступно специалистам с высшей квалификацией и практически полностью зависит от исполнителя, его опыта, знаний (как в предметной области, так и в методоиках тестирования), способности быстро проникать в суть.

Во время исследовательского тестирования неформальные (не созданные заранее) тестовые сценарии разрабатываются, выполняются, анализируются и оцениваются динамически. Результаты тестирования используют для изучения компонента или системы и последующей разработки тестовых сценариев для непокрытых областей.

Исследовательское тестирование проводится сессиями. Сессия – это выделенный промежуток времени, в котором тестировщик исследует программу, ориентируясь на поставленную цель. Например, требуется проверить все поля ввода на странице. Во время сессии ведётся протокол, а тестировщик фиксирует действия и результаты.

* **По уровню проводимого тестирования:**

Это определяет объект тестирования.

1. **Компонентное (модульное или юнит-тестирование) тестирование**. Тестирование отдельных компонентов программного продукта, сфокусированное на их специфике, назначении и функциональных особенностях.
2. **Интеграционное тестирование**. Если в вашем проекте более одной компоненты, он нуждается в интеграционном тестировании. Данный вид тестирования проводится после компонентного тестирования и направлен на выявление дефектов взаимодействия различных подсистем на уровне потоков управления и обмена данными. Поэтому очень важно учитывать опыт компонентного тестирования, при этом соблюдая бизнес-ориентацию тест-кейсов.
3. **Системное** – проверка работы приложения целиком.
4. **Приёмочное** – оценка соответствия заявленным требованиям к программному продукту.

* **По уровню функционального тестирования:**

1. **Smoke-тестирование**

**Smoke-тестирование или Дымовое тестирование** – предварительная проверка самой важной функциональности программного продукта.

Почему дымовое тестирование так называется, есть несколько версий. Наиболее правдоподобная версия: термин инженеров-электротехников. Когда на электрическую цепь впервые подавалось питание, она могла дымиться если пайка была небрежной, или если в устройстве были дефекты.

Дымовые тесты обычно запускаются после сборки, чтобы убедиться, что приложение может быть запущено и работают самые основные функции, прежде чем будет выполнено более всестороннее тестирование.

Дымовое тестирование может быть как ручным, так и автоматизированным.

Ручное смок-тестирование – это процесс проверки ключевых функций на явные дефекты. Чаще всего этим и ограничиваются, особенно если приложение небольшое.

Автоматизированное смок-тестирование – пишутся скрипты, проверяющие ключевые функции. Иногда это бывает целесообразно, если действия стандартные и повторяемые.

Может быть и гибридное тестирование: сочетание ручного и автоматизированного.

Этапы дымового теста могут варьироваться в зависимости от конкретного приложения и целей теста, но обычно включают следующее:

○ Проверка сборки: убедитесь, что сборка завершена и может быть установлена в целевой среде.

○ Настройка среды: убедитесь, что тестовая среда настроена правильно и соответствует необходимым требованиям.

○ Выполнение теста: запустите наиболее важные и часто используемые функции приложения, чтобы определить, работают ли они должным образом.

○ Оценка результатов испытаний: проанализируйте результаты дымовых испытаний и определите, произошли ли какие-либо сбои.

○ Отчет о дефектах: задокументируйте любые сбои или дефекты, обнаруженные во время дымового теста.

○ Очистка: удалите все временные файлы и восстановите тестовую среду до исходного состояния.

Таким образом, Smoke-тестирование или проверка сборки проводится для того, чтобы до запуска продукта убедиться, что всё работает стабильно и отвечает требованиям заказчика. Оно проводится при каждой новой сборке. У дымового тестирования много преимуществ: оно помогает заметить дефекты на раннем этапе, повысить качество системы и экономит время команды.

1. **Тестирование критического пути** (critical path) — направлено для проверки функциональности, используемой обычными пользователями во время их повседневной деятельности.
2. **Расширенное тестирование** (extended) — направлено на исследование всей заявленной в требованиях функциональности.

**Нефункциональное тестирование** оценивает такие качества программного продукта, как, например, эргономику или производительность.

Думаю, важность данного вида тестирования понятна и не требует обоснования. Ведь всем понятно, что если, к примеру, производительность системы не достаточна, то пользователям придется по пол дня ждать отклика на свои действия, что может привести к их массовой спячке.

Как следует из названия, при нефункциональном тестировании проверяется соответствие программного продукта нефункциональным требованиям из технического задания на его создание. И, как в случае с функциональным тестированием, для нефункционально разрабатывается программа и методика испытаний.

**Виды нефункционального тестирования.**

1. **Тестирование производительности:**

○ **Нагрузочное тестирование**

**○ Тестирование масштабируемости**

**○ Объемное тестирование**

**○ Стрессовое тестирование**

Тестирование производительности помогает определить работоспособность, стабильность, потребление ресурсов в условиях разных сценариев использования и нагрузок.

Задача системы – обрабатывать нужное количество данных за установленное время. В случае превышения запланированных объёмов входных данных, система восстанавливается после отказа без потери данных.

Нагрузочное тестирование – процесс анализа производительности тестируемой системы под воздействием нагрузок. Цель нагрузочного тестирования- определить способность приложения к внешним нагрузкам. Обычно испытания проводятся в несколько этапов.

1. Генерация тестовых сценариев

Для эффективного анализа сценарии должны быть наиболее близки к реальным сценариям использования. Важно понимать, что всегда возможны исключения, и даже самый подробный план тестирования может не покрыть отдельно взятого случая.

1. Разработка тестовой конфигурации

Имея сценарии тестирования, важно распределить порядок возрастания нагрузки. Для успешного анализа необходимо выделить критерии оценки производительности (скорость отклика, время обработки запроса и т.д.).

1. Проведение тестового испытания

При проведении тестов важно своевременно следить за исполнением сценариев и откликом тестируемой системы. Для эмуляции высоких нагрузок требуется серьезная аппаратная и программная инфраструктура. В некоторых случаях для удешевления работ применяются методы математического моделирования. За основу берутся данные, полученные при низких нагрузках, и аппроксимируются. Чем выше уровень моделируемой нагрузки, тем ниже точность оценки. Однако подобный способ существенно сокращает расходы.

Тестирование масштабируемости – тестирование для измерения возможностей вертикального и горизонтального масштабирования с точки зрения любой из нефункциональных возможностей: увеличение количества пользователей, рост количества транзакций, увеличение объёма данных.

Вертикальное масштабирование – это увеличение производительности каждого компонента системы для повышения общей производительности. Например, увеличивается объём оперативной памяти на сервере, чтобы он быстрее обрабатывал запросы. Это повысит производительность всей системы.

Горизонтальное масштабирование – разбиение системы на структурные компоненты и разнесение их по отдельным физическим машинам. А также увеличение количества серверов, параллельно выполняющих одну и ту же функцию. Например, увеличивается количество серверов, но каждый выполняет одну и ту же задачу: принимает одни и те же запросы и отвечает на них. Если разработчики заранее не подумают, как они увеличат ресурсы при росте популярности, они потеряют значительную часть прибыли.

Объёмное тестирование – тестирование на больших объёмах данных. Например, тестируется поведение приложения при попытке загрузить в его базу данных нескольких файлов очень большого размера.

Стресс тестирование. У любой системы есть предел нормального функционирования. При превышении предела система попадает в состояние стресса и значительно меняет свое поведение. Стресс тестирование проверяет работу приложения в условиях превышения пределов нормального функционирования. Особенно это важно для "критичных" программ: банковского ПО, программ авиационной отрасли, медицины. Стресс тестирование проводят не только на стадии разработки программного обеспечения, но и на протяжении всего цикла функционирования с целью получения и обработки данных поведения системы за долгий период времени.

1. **Инсталляционное тестирование**

Оценка вероятности возникновения проблем при установке, удалении, а также обновлении ПО при различном программном и аппаратном окружении. Оно позволяет оценить работоспособность системы после завершения работы инсталлятора.

1. **Тестирование пользовательского интерфейса (GUI/UI-тестирование).**

Удобство пользователя при взаимодействии с разными параметрами интерфейса (кнопки, цвета, выравнивание и т. д.).

Проверка соответствия интерфейса и требований, насколько удобно пользователям работать с программным продуктом. Проверяют, ведёт ли себя программное обеспечение в соответствии со спецификацией, когда пользователь взаимодействует с ним с помощью клавиатуры и мыши (когда тестируется десктопное приложение), или с помощью сенсорного экрана, жестов или движений устройства (когда тестируется мобильное приложение).

1. **Тестирование UX (юзабилити)** – правильность логики использования программного продукта.

Любое приложение создается для того, чтобы им воспользовались. Удобство использования - важный качественный показатель программы. IT индустрия знает множество примеров, когда проекты взлетали после удачного исправления удобства использования. Чем шире аудитория, тем важнее фактор юзабилити. Тестирование юзабилити включает в себя детальный анализ поведения пользователей. Для оценки эргономики важно иметь данные не только о скорости выполнения бизнес-задачи, но и об эмоциях пользователя, мимике лица, тембра голоса. Результат грамотного UX-тестирования – перечень рекомендаций: что и как изменить, чтобы повысить количество конверсий и превратить посетителей сайта в постоянных и преданных пользователей

1. **Тестирование локализации**

Оценка правильности версии программного продукта (в т.ч. языковой и культурный аспекты).

1. **Тестирование безопасности**

Определение уровня безопасности ПО: защищено ли оно от атак хакеров, несанкционированного доступа к данным и т. д.

Основные понятия, которые охватывает тестирование: конфиденциальность, целостность и сохранность данных, аутентификация, авторизация и невозможность отказа от авторства (атрибуты качества). Проводится для тех объектов, в работе которых обеспечение защищённости — одна из важнейших задач.

1. **Тестирование надежности**

Тестирование работы программы при длительной средней ожидаемой нагрузке.

Неважно, как долго идёт это тестирование, основная задача — наблюдая за потреблением ресурсов в течение определённого времени, выявить утечки памяти и проследить, чтобы скорость обработки данных или время отклика в начале теста и с течением времени не уменьшалась. Иначе вероятны сбои в работе продукта и перезагрузки системы.

1. **Конфигурационное тестирование.** Включает в себя кроссбраузерное и кроссплатформенное тестирование. Такое тестирование дает уверенность, что приложение заработает на разных платформах (на разных операционных системах, либо на разных видах устройств: десктоп, мобильный, планшет) и в разных браузерах, а значит у максимального числа пользователей.

[**Глава 2. Составление тестовой документации для ручному тестированию веб-приложения**](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-3-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D1%82%D0%B5%D1%81%D1%82%D0%BE%D0%B2%D0%BE%D0%B9-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D0%B0%D1%86%D0%B8%D0%B8-%D0%B4%D0%BB%D1%8F-%D1%80%D1%83%D1%87%D)**.**

**Глава 2.1. Составление чек-листов**

**Чек-листы** тестировщика – это список задач, которые нужно выполнить в процессе тестирования. В чек-листе могут быть перечислены тест-кейсы, условия тестирования, требования к продукту и многое другое. Цель чек-листа – не пропустить ни одной важной детали в процессе тестирования.

Использование чек-листов помогает тестировщикам упростить и структурировать свою работу, а также не забыть важные детали. Кроме того, чек-листы могут быть использованы в качестве инструмента для автоматизации тестирования.

**Составление чек-листов** – это процесс, который может значительно упростить и ускорить тестирование программного обеспечения. Ниже представлены **основные шаги**, которые помогут вам составить эффективный чек-лист.

1. Определите цели и задачи. Перед тем, как начать составление чек-листа, необходимо понять, какие задачи вы хотите решить с помощью него. Какие функциональные возможности продукта вы хотите проверить? Какие аспекты безопасности требуют особого внимания? Что важно для пользователей? Определите цели и задачи, которые вы хотите достичь с помощью чек-листа, и исходя из этого составьте его структуру.

2. Разбейте на категории. Разбейте продукт на категории и подкатегории, чтобы вам было легче ориентироваться. Например, если вы тестируете веб-приложение, вы можете разбить его на следующие категории: регистрация, авторизация, работа с профилем пользователя, работа с контентом, оплата и т.д. Каждая категория должна содержать соответствующие тест-кейсы.

3. Проверьте чек-лист. Проверьте ваш чек-лист на полноту и актуальность. Убедитесь, что вы не упустили ничего важного и что все соответствуют целям и задачам, которые вы определили в начале.

Составление чек-листа – это творческий процесс, который требует от вас внимательности, аккуратности и профессионализма. Чем более структурированным и подробным будет ваш чек-лист, тем более эффективнее и быстрее будет проходить тестирование.

Некоторые **советы**, которые помогут вам составить более эффективный чек-лист

● Используйте чек-листы в качестве дополнительного инструмента. Чек-листы не могут заменить полноценное тестирование продукта, но могут помочь выявить проблемы и ошибки быстрее и более эффективно.

● Составляйте чек-листы в соответствии с конкретными требованиями. Каждый продукт имеет свои уникальные особенности, поэтому чек-листы должны быть адаптированы под конкретный продукт и его требования.

● Обновляйте чек-листы регулярно. Продукты постоянно меняются и обновляются, поэтому чек-листы должны быть пересматриваемы и обновляемы регулярно.

● Работайте в команде. Чек-листы могут быть использованы как инструмент для совместной работы тестировщиков и разработчиков. Разделите ответственность между членами команды и используйте чек-листы для координации работы.

Чем более детально и конкретно вы составите чек-лист, тем более эффективным будет тестирование продукта.

Составление чек-листов - это важный инструмент для эффективного тестирования продуктов. Чек-листы помогают выявлять проблемы и ошибки быстрее и более эффективно, а также упрощают совместную работу тестировщиков и разработчиков. Следуйте советам, описанным выше, и создавайте детальные и адаптированные под конкретный продукт чек-листы, чтобы обеспечить более качественное тестирование продукта.

**Правила составления.**

1. Один пункт — одна операция.

Пункты чек-листа — это однозначные атомарные (выполняющее одно действие) и полные (логически завершённые) операции. Например, авторизация в профиль пользователя и редактирование данных пользователя — две разные задачи. В списке проверок их оформляют отдельными пунктами.

2. Пункты всегда начинаются с существительного или глагола неопределённой формы. При составлении пунктов важно придерживаться унифицированных форм: начинать их с существительного («Проверка», «Добавление», «Отправка») или неопределённой формы глагола («Проверить», «Добавить», «Отправить»). Так пункты будут понятными и однозначными.

3. Соблюдайте структуру. Чтобы выполнять чек-лист было удобно, лучше составлять тесты так, чтобы они исходили из логики использования функционала. Например:

● в рамках раздела «Регистрация и Личный профиль» — регистрация на сайте, редактирование профиля;

● в рамках раздела «Форма обратной связи» — валидация полей, отправка письма, доставка письма.

4. Ничего не придумывайте. Чек-лист важно составлять, опираясь на требования. Не придумывайте то, чего нет в требовании.

5. Общепринятые названия. Хорошо, когда в команде все разговаривают на одном языке, используют общую терминологию.

Если один тестировщик в команде называет линию, разделяющую элементы на странице, дивайдером, а остальные — сепаратором, может возникнуть недопонимание.

**Правила выполнения чек-листа.**

1. Не пропускать проверки чек-листа. Если какие-то пункты нельзя проверить из-за багов или недоступности среды, надо отметить их и вернуться к ним позже.

2. Если указываются тестовые данные, нужно использовать именно их.

3. Указывать результат и комментарий.

**Резолюция.**

Когда исследуемый объект протестирован, нужно выставить кейсу резолюцию, то есть вердикт – прошёл он проверку или нет.

Рассмотрим распространённые статусы и разберёмся, какой из них когда используется:

● passed – успешно пройден

● failed – кейс не прошел проверку

● skipped – пропуск проверки

● blocked – проверка заблокирована

● untested – кейс еще не брали в работу

passed – ставится, когда исследуемый объект проходит проверку в соответствии с ТЗ.

failed – исследуемый объект не соответствует ТЗ. Например, функционал поломан, не работает или его нет (разработчик потерял кнопку).

skipped – пропуск проверки кейса. Например, из-за нехватки времени или если в кейсе есть неточности: он устарел, из документации вы понимаете, что его уже изменили в проекте.

blocked – проверка заблокирована поломанным функционалом, от которого прямо или косвенно зависит тестируемый. Например, есть функционал «авторизация пользователя» и функционал «редактирование данных пользователя». Разработчик допустил ошибку в коде и функционал авторизации поломан — ему мы выносим резолюцию failed. Проверить редактирование данных пользователя не можем — выставляем статус blocked. В комментарии указываем, почему не можем протестировать этот функционал.

В данном проекте Чек-лист применялся при Исследовательском тестировании, как основная тестовая документация. Чек-лист выполнялся в Qase.io (Приложение 2. Test runs. Исследовательское тестирование.).

**Глава 2.2. Составление чек-кейсов.**

**Тест-кейс** – это такое описание проверки работы системы, которое может выполнить любой человек из команды, будь то тестировщик, разработчик, аналитик или даже бизнес-заказчик.

Набор тест-кейсов называется тестовым набором (test suite).

**Стандартные атрибуты тест-кейса на этапе их проектирования:**

1. **Номер** - уникальный идентификатор тест-кейса. Его удобно использовать для одинакового понимания, о какой проверке идет речь (например, дать ссылку в баге).

2. **Название** – краткое описание сути проверки. Должно помещаться в твиттер и быть понятным! Кратко, но емко.

3. **Предварительные шаги** – описание действий, которые необходимо выполнить, но прямого отношения к проверке они не имеют (например, зарегистрироваться в системе для проверки создания элемента). Если предварительных шагов нет, то секция не заполняется.

Шаги – описание действий, необходимых для проверки (например, создание элемента).

4. **Ожидаемый результат** (ОР) – сама проверка: что мы ожидаем получить после выполнения шагов ("Элемент создан").

5. **Приоритет** тест-кейса зависит от приоритета функций и сценариев, которые он проверяет. Определяет очерёдность выполнения теста во время тестирования. Например, при высоком приоритете тест-кейс будет в начале очереди на выполнение, а при низком — в конце. Если во втором случае что-то сломано, для нас это не так критично, как в первом.

6. **Тестовые данные** – данные, которые используются для проверки. Могут указываться в шагах тест-кейса или в отдельном файле с указанием ссылки на них. Это, например, данные для авторизации пользователя или данные тестовой карты для оплаты заказа.

7. **Предусловия** – действия, которые нужно выполнить, прежде чем приступать к тест-кейсу, а также настройки приложения и тестовой среды. Например, если мы проверяем отправку сообщения в чате с другом в социальной сети, мы точно знаем, что пользователь должен быть авторизован. Но сама авторизация как набор действий с вводом логина/пароля не имеет отношения к отправке сообщения другу. Таким образом, правильно будет указать в предусловии этого кейса, что пользователь должен быть авторизован, а страница чата — открыта. В самих шагах мы будем описывать действия, имеющие непосредственное отношение к отправке сообщения другу.

8. **Постусловия** – выполнение тест-кейса, как правило, переводит систему из одного состояния в другое: меняются настройки, производятся расчёты. Состояние, в которое нужно привести систему после прохождения тест-кейса, указывается в постусловии. Допустим, мы проводили тестирование кейса с редактированием данных пользователя на боевом сервере (на окружении, которым пользуются обычные пользователи) и в процессе тестирования изменили часть данных на ненастоящие: вместо имени указали странную последовательность букв, в описании интересов написали несвязный текст, вместо фотографии человека прикрепили небо. Но, так как мы тестируем на продакшене, наш тестовый профиль может попасть в выдачу обычным пользователям, и это будет выглядеть странно. Чтобы этого избежать, мы можем в постусловиях указать, что все изменённые значения надо вернуть в исходное состояние и указать, какие данные надо ввести в каждое поле.

**Атрибуты на этапе исполнения тест-кейсов:**

1. **Фактический результат** – то, что мы получаем после выполнения всего тест-кейса или его конкретного шага. Это необязательный атрибут, его можно не указывать. Но если он указывается, в нём может быть, как краткое описание того, что мы получили по факту прохождения шага/шагов, так и статус для каждого фактического результата. Если фактический результат совпадает с ожидаемым, в графе с фактическим ставится статус passed – тест-кейс успешно пройден, ошибок не обнаружено. Если результат отличается от ожидаемого – статус failed – проверка провалена, полученный результат не соответствует ожидаемому. Во втором случае тестировщик создаёт отчёт о дефекте. К тест-кейсу обязательно добавляется ссылка на дефект, обнаруженный при исполнении.

2. Статус тест-кейса обозначает результат исполнения этого кейса или причину, по которой он не может исполняться. Основные статусы:

● **passed** – успешно пройден

● **failed** – кейс не прошел проверку, исследуемый объект не соответствует ТЗ

● **skipped** – пропуск проверки

● **blocked** – проверка заблокирована

● **untested** – кейс еще не брали в работу

Кроме того, для некоторых тест-кейсов могут потребоваться дополнительные атрибуты:

● **Требования к среде** – специальное оборудование, программное обеспечение и т. п. вещи, необходимые для выполнения тест-кейса и не перечисленные в соответствующей спецификации проекта тестирования.

● **Специальные процедурные требования** – особые процедуры настройки, выполнения или очистки, уникальные для этого тест-кейса.

● **Межкейсовые зависимости** – тест-кейсы, которые нужно выполнить перед этим тест-кейсом.

**Виды тест-кейсов.**

Тест-кейсы делят на несколько групп в зависимости от входных данных, действий и предполагаемого поведения системы.

1. **Позитивные тест-кейсы**. Доказывают, что программное обеспечение отвечает всем требованиям: если были введены верные данные, а пользователь следовал указаниям, система реагирует адекватно.

2. **Негативные тест-кейсы**. Их результаты позволяют убедиться в способности программного обеспечения правильно реагировать на ошибочные вводные или некорректные действия. Это может быть, например, появление всплывающего окна с подсказкой.

3. **Деструктивные тест-кейсы**. Служат для проверки способности системы выдерживать большие нагрузки и внешние воздействия без утери данных пользователя. Должно соблюдаться условие о запрете разрушения аппаратной части.

**Правила написания тест-кейсов.**

1. **Заголовок** -

● должен быть чётким, кратким, понятным и однозначно характеризующим суть тест-кейса;

● не может содержать выполняемые шаги и ожидаемый результат.

2. **Предусловие** -

● может содержать полную информацию о состоянии системы или объекта, необходимом для начала выполнения шагов тест-кейса;

● может содержать ссылки на информационные источники, которые необходимо изучить перед прохождением тест-кейса (инструкции, описание систем…);

● не может содержать ссылки на тестируемый ресурс, если у информационной системы более одной среды (прод, тест, препрод…), данная информация должна быть вынесена в инструкцию, и ссылка приложена в предусловии;

● не может содержать данные для авторизации, данная информация должна быть вынесена в инструкцию, и ссылка приложена в предусловии;

● не может содержать выполняемые шаги и ожидаемый результат, если нам нужно, чтобы до выполнения шагов проверки у нас была открыта главная страница, то мы в предусловии указываем «открыта главная страница сайта»;

● не может содержать ожидаемый результат.

3. **Шаги проверки** -

● должны быть чёткими, понятными и последовательными;

● следует избегать излишней детализации шагов. Правильно: «ввести в поле число 12». Неправильно: «нажать на клавиатуре на цифру ‘1’, следующим шагом нажать на клавиатуре на цифру ‘2’»;

● должны использоваться безличные глаголы. Правильно: нажать, ввести, перейти. Неправильно: нажмите, введите, идите;

● не должно быть комментариев и пояснений, если есть необходимость привести мини-инструкцию, то оформляем инструкции в базе-знаний и в предусловии ссылаемся на неё;

● не должно быть жёстко прописанных статических данных (логины, пароли, имена файлов) и примеров, для исключения эффекта пестицида.

4. **Ожидаемый результат** -

● должен быть у каждого шага проверки;

● должно быть кратко и понятно описано состояние системы или объекта, наступающее после выполнения соответствующего шага;

● не должно быть избыточного описания.

**Общие требования к тест-кейсам:**

1. язык описания тест-кейсов должен быть понятен широкому кругу пользователей, а не узкой группе лиц;

2. тест-кейс должен быть максимально независим от других тест-кейсов и не ссылаться на другие тест-кейсы (лучшая практика, когда зависимостей нет вообще);

3. тест-кейсы группируются в функциональные блоки по их назначению;

4. в тест-кейсах, проверяющих работу функционала скриншотов быть не должно, иначе вы будете посвящать сотни часов на изменение всех скриншотов в тысячах тест-кейсах при изменении интерфейса тестируемой программы. Скриншоты могут быть добавлены только в тест-кейсы, проверяющие отображение страниц и форм.

**Плюсы и минусы тест-кейсов.**

● Главное достоинство тест-кейса состоит в том, что его может провести практически любой сотрудник компании, не имеющий отношения к работе над проектом. Если к созданию тест-кейса подошли ответственно, исполнитель справится с ним без труда.

● Минусы такого типа тестирования тесно взаимосвязаны. Заполнение требует долгой монотонной работы. Например, при тестировании корректного ввода ФИО надо выполнять простейшие одинаковые шаги: «ввести только символы, «ввести только числа» и т.д. Это особенно заметно, если посмотреть несколько тест-кейсов на один и тот же функционал. Затратное по времени редактирование. Малейшее изменение содержания сайта требует коррекции сотен сценариев тест-кейса. Не самое увлекательное и довольно выматывающее занятие.

**Ошибки в тест-кейсах**

1. Заголовка нет или он сформулирован некорректно. Например, из заголовка «Кнопка “Сохранить”» или «Чат с поддержкой» непонятно, что мы проверяем в кейсе. То, что они есть, или их функциональность? Заголовок должен кратко, но чётко описывать суть проверки.

2. Ссылки ведут на разные или недействительные требования. В тест-кейсах можно ссылаться на требования, но, если мы используем несколько ссылок, стоит убедиться, что их содержимое актуально и не противоречит друг другу.

3. Используются личные формы глаголов: «нажми», «перейдите», «укажи». В тест-кейсах, как и в чек-листах, следует использовать обезличенные формы глаголов: «нажать», «перейти», «указать».

4. Пунктуационные, орфографические, синтаксические ошибки. Текст с ошибками и опечатками читать сложнее, чем текст, не нарушающий правила языка.

5. «Выдумывание» особенностей поведения приложения без отсылки к требованиям. Тест-кейс должен описывать сценарий и ожидаемые результаты чётко в соответствии с требованиями. Тестировщик ничего не выдумывает.

6. Нет описания приготовления к выполнению тест-кейса, если требуются предусловия. Например, для проверки фишек верифицированного пользователя в соцсети нам нужен такой верифицированный пользователь.

При тестировании у нас есть возможность создать его через админку (не так, как это происходит у обычного пользователя). Нам подойдёт этот вариант, так как мы проверяем не верификацию, а функцию, доступную такому пользователю. Соответственно, верификация пользователя — наше предусловие. В нём обязательно нужно либо сказать, что надо верифицировать пользователя по такой-то ссылке, либо дать конкретные данные для авторизации пользователем.

В итоге, хороший тест-кейс: понятен любому члену команды; аккуратно и точно написан; соответствует требованиям; воспроизводим; пригоден для многократного использования.

Суть тест-кейсов заключается в том, что они делают процесс тестирования программного обеспечения структурированным и доступным для неспециалистов. В чек-листах прописываются объекты проверки, а в тест-кейсах — пошаговый алгоритм.

Применение данного формата тестирования систем позволяет значительно экономить время на проверках. Гораздо рациональнее один раз потратить время на основательную подготовку набора тест-кейсов, чем каждый раз разрабатывать новое тестирование продукта.

**Глава 2.3. Анализ тестирования и тест-дизайн.**

**Анализ тестирования.**

Представим, что мы тестируем интернет-магазин. Исходя из требований (базиса тестирования) мы понимаем, что именно нам нужно протестировать. Например, нам надо проверить, что пользователь может зарегистрироваться, войти в приложение, найти там товар, добавить его в корзину, после чего оплатить и получить.

Дальше, в рамках анализа тестирования, нам нужно определить следующие вещи:

1. Уровни тестирования – на сколько глубоко нам нужно протестировать каждое требование (Компонентное, интеграционное и системное тестирование);

2. Уровень детализации и качества наших требований;

3. Связи между требованиями;

4. Сложность нашего приложения;

5. Продуктовые и проектные риски;

6. Жизненный цикл разработки ПО и его длительность;

7. Инструменты по управлению тестированием (Test management system);

8. Опыт и постоянность команды;

9. Доступность для консультации других участников проекта.

Исходя из вышеописанных моментов, мы можем принять решение о том, на сколько глубоко нам надо тестировать конкретное требование и какой вид документации лучше всего применить.

Например, если проект представляет собой сложную систему, с высокими рисками и нестабильной командой – то необходимо будет выбрать наиболее подробный вид документации, скажем тест-кейс. Из-за высоких рисков и сложности тесты необходимо будет проектировать на всех уровнях и максимально детально. Благодаря максимально проработанным тестам новым членам команды будет намного проще войти в проект нежели при использовании менее детальной документации.

Главный принцип для выбора документации – это окупаемость этой самой документации.

Я думаю логика понята. Чем сложнее, рискованней, дольше и стабильней наш проект, тем глубже и детальнее нужно прорабатывать тесты.

Для простого проекта, с невысокими рисками и продолжительность, с не совсем стабильными требованиями и стабильной командой можно использовать высокоуровневую тестовую документацию, например, чек-листы. В противном случае мы рискуем потратить большую часть времени на тест дизайн и поддержку документации, а не на выполнение тестов.

Иногда для проверки разных требований может применяться тестовая документация разных уровней.

Например, для сложного и рискового функционала – детальные тест кейсы, а для простого и не рискового – чек-лист.

Зачем делать анализ тестирования?

Можно выделить 2 больших достоинства анализа тестирования:

1. Создание ясности и гибкости в других активностях тестирования благодаря определению точных целей тестов. Мы знаем, что именно и на сколько глубоко нужно тестировать.

2. Предотвращение дефектов: именно на этапе анализа тестирования проводится верификация и статическое тестирование.

После определения того, что мы будем делать, можно приступить к этапу создания тестов.

**Проектирование тестов и Тест-дизайн.**

Разработка тест-кейсов – важный этап в жизненном цикле тестирования. От того, насколько правильно написаны тест-кейсы, может зависеть весь процесс тестирования. Создавать эффективные тест-кейсы позволяют техники тест-дизайна.

Тест-дизайн – этап тестирования ПО, на котором проектируются и создаются тестовые случаи (тест-кейсы). Они соответствуют определённым ранее критериям качества и целям тестирования.

Важно, что критерии качества и цели тестирования должны быть определены до начала написания тест-кейсов. От этих критериев и целей зависит, какими будут тест-кейсы, для каких модулей они будут описаны в первую очередь, проверка каких функций будет приоритетной. Когда мы пишем тест-кейсы, одна из основных задач – создать оптимальное тестовое покрытие функциональности, то есть не допустить «слепых зон» в системе, которые не покрываются проверками.

Задачи тест-дизайна на проекте:

● максимально покрыть функциональность тестами;

● обнаружить серьёзные баги;

● сократить количество тестов, исключив непродуктивные;

● не пропустить важные тесты.

**Техник тест-дизайна** довольно много. Мы сосредоточимся на самых популярных:

1. **Классы эквивалентности**.

2. **Анализ граничных значений**.

3. **Переход состояний**.

4. **Попарное тестирование**.

5. **Таблицы принятия решений**.

Эквивалентное разделение и анализ граничных значений направлены на сокращение количества необходимых тестовых сценариев. В связи с этим при разработке тестов для тестирования методом черного ящика эти техники применяются чаще всего.

**Классы эквивалентности.**

Класс эквивалентности – набор данных, которые обрабатываются одинаковым образом и приводят к одному результату.

Эквивалентное разделение подразумевает разбиение тестовых данных на классы по какому-то признаку. Этот метод имеет смысл только в том случае, если компоненты чем-то похожи и могут войти в общую группу. Если мы выбираем в качестве техники тест-дизайна эквивалентное разделение, это означает, что мы будем тестировать только несколько значений из каждого класса элементов. Помните, что это не гарантирует отсутствия ошибок в остальных значениях, не охваченных тестами. Мы лишь предполагаем, что использование нескольких элементов из каждой группы будет достаточно показательным.

Эквивалентное разделение – хорошее решение для случаев, когда вы имеете дело с большим объемом входящих данных или множеством одинаковых вариантов ввода. В противном случае, возможно, имеет смысл более тщательно охватить продукт тестами.

Пример эквивалентного разделения. Допустим, есть интернет-магазин, который предлагает разные тарифы на доставку в зависимости от стоимости корзины.

1. Стоимость доставки для заказов на сумму менее 1000 руб составляет 200 руб.

2. Стоимость доставки для заказов на сумму более 1000 руб составляет 100 руб.

3. При заказе от 3000 руб долларов доставка бесплатна.

У нас есть следующие ценовые категории для работы:

1. от 1 руб до 1000 руб.

2. от 1000 руб до 3000 руб.

3. 3000 руб и выше.

При использовании техники эквивалентного разделения мы получаем три набора данных для тестирования:

1. От 1 до 1000:

* допустимые значения: любая цена в диапазоне от 1 до 999,99
* недопустимые значения: любая цена ниже 1 или выше 999,99

2. От 1000 до 3000:

* допустимые значения: любая цена в диапазоне от 1000 до 2999,99
* недопустимые значения: любая цена ниже 1000 или выше 2999,99

3. 3000 и выше:

* допустимые значения: любая цена выше 2999,99;
* недопустимые значения: любая цена ниже 3000.

Таким образом, мы можем выбрать несколько чисел из каждого диапазона цен и предположить, что остальные числа из этих диапазонов будут давать такие же результаты.

Аналогичные действия проводятся и с другими данными, которые используются системой, например, со временем.

В примере выше мы рассмотрели данные, которые можно расположить на числовой прямой – классы эквивалентности этих данных будут линейными. Их можно разбить на диапазоны с точными границами начала и конца.

Нелинейные классы эквивалентности – это набор неупорядоченных данных. У них нет границ, они являются частью множества данных. Пример — расширения файлов, операционные системы, группы пользователей с различными правами (пользователь, модератор, администратор) и так далее. В этом случае можно выделить только два класса эквивалентности:

● валидный – соответствует требованиям,

● невалидный – не соответствует требованиям или обрабатывается системой отличным от валидного класса образом.

Рассмотрим другой пример. Тестируется система таможенного контроля, которая обрабатывает паспортные данные выезжающего за границу и определяет, есть ли он в базе лиц, для которых выезд ограничен (например, из-за финансового долга).

Здесь есть два класса эквивалентности:

1. Валидный — данных пересекающего границу нет в базе лиц, которым выезд запрещён. Выезд разрешён.

2. Невалидный — данные пересекающего границу внесены в базу. Выезд запрещён.

Кроме чисел, на классы эквивалентности можно разбить:

● символы — они могут быть валидными (@ в адресе электронной почты) и невалидными (?, %,\*);

● длину строки — например, валидный класс от 1 до 30 знаков, невалидный — всё остальное (меньше 1 и больше 30);

● объём памяти, который необходим приложению для стабильной работы;

● разрешение экрана — всё, что меньше или больше заявленных требований к разрешению экрана, будет относиться к невалидным классам;

● версии операционных систем, библиотек — также определяются согласно требованиям. Например, приложение должно работать на ОС Windows 7, но поддержка Windows Nt уже не требуется.

● объём передаваемых данных — по требованиям. Например, если мощности сервера не позволяют обработать объём данных больше определённого значения.

Классы эквивалентности – одна из основных техник тест-дизайна. Именно с ней тестировщики и тест-дизайнеры работают чаще всего. Она сокращает число тестов (можно выбрать только несколько значений из класса эквивалентности), но к использованию нужно подходить внимательно: если неверно выделить класс эквивалентности, можно получить некорректные результаты тестирования и пропустить ошибку.

**Анализ граничных значений.**

Анализ граничных значений в чем-то похож на эквивалентное разделение. Можно даже сказать, что оно лежит в основе анализа граничных значений. Но есть некоторые отличия.

При анализе граничных значений мы тоже группируем данные по линейным эквивалентным классам, но проверяем не значения из определенного класса, а граничные значения – те, которые находятся на «границах» классов.

Граничные значения обязательно используются при написании тестов, так как именно на границе классов эквивалентности чаще всего и обнаруживаются ошибки. Например, если в требованиях указано, что пользователь сайта должен быть старше 16 лет, тестировщику следует уточнить у аналитика, входит ли значение «16 лет» в валидный класс эквивалентности. А затем – проверить, действительно ли это реализовано в приложении. Может оказаться, что разработчик понял требования иначе и указал в коде, что сайтом могут пользоваться лица с 17 лет (>16 вместо >=16).

Алгоритм определения граничных значений:

1. Выделить классы эквивалентности.

2. Определить граничные значения этих классов.

3. Определить, к какому классу будет относиться каждая граница.

4. Для каждой границы провести тесты: проверить значения до границы, на ней и сразу после неё.

**Переход состояний.**

Диаграмма перехода состояний визуализирует состояния программы в разные периоды времени и на разных этапах использования. Визуальную информацию воспринимать проще, чем текст. Таким образом, техника перехода состояний позволяет быстрее получить максимальное тестовое покрытие.

Этот метод эффективен при создании наборов тестов для систем со множеством вариаций состояний. Он вам пригодится для тестирования последовательности событий с конечным числом входных параметров.

Прежде чем рассматривать эту технику, познакомимся с основными понятиями, которые используются при составлении диаграмм переходов и состояний.

**Точка входа** – старт работы системы или приложения.

**Переход** – переход системы из одного состояния в другое. Происходит в результате действий пользователя или при определённых условиях.

**Событие** – действие пользователя, которые он выполнил для перевода системы в другое состояние. Или действия самой системы, меняющие её состояние.

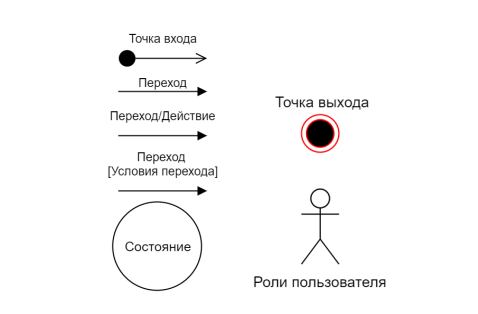
**Действие** – реакция приложения на действия пользователя или самой системы (на событие).

**Условия перехода** – условия, которые необходимы для перехода системы в другое состояние. Например, изменение даты для начисления процентов на вклад.

**Состояние** – состояние системы до или после перехода в результате действий пользователя или при определённых условиях.

**Точка выхода** – успешное окончание полного цикла работы приложения, то есть выполнение всех переходов и состояний.

**Роли пользователей** – пользователи, которые могут по-разному влиять на систему в зависимости от уровня прав доступа (зарегистрированный пользователь, администратор).



Классический пример – бронирование авиабилетов. Начнём с позитивного сценария: пользователь успешно проходит весь цикл бронирования, включая оплату и использование билета. Всегда стоит начинать с позитивных проверок, чтобы убедиться, что система работоспособна и выполняет ключевые функции.

Если это не так, дальнейшее тестирование не имеет смысла до устранения дефектов.



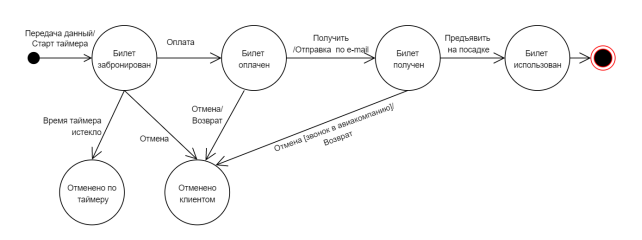
1. Точкой старта будет вход в систему бронирования и выбор нужного билета. Затем пользователь передаёт информацию, нужную для бронирования (имя и фамилию, паспортные данные), и нажимает кнопку «Забронировать». Нажатие можно считать событием. После него стартует таймер до окончания срока оплаты. Система перешла в первое состояние — «Билет забронирован».

2. Дальнейшее событие — «Оплата билета». Переводит систему в следующее состояние — «Билет оплачен».

3. Затем по событию «Получить билет» система должна выполнить действие «Отправка билета по email» и перейти в другое состояние — «Билет получен».

4. Последним звеном в этой цепочке будет событие «Предъявление билета при посадке» (в примере часть событий пропущена — в реальных проектах их, конечно, может быть намного больше). Состояние «Билет использован» — цикл бронирования успешно завершён, система попадает в точку выхода.

Рассмотренный сценарий — позитивный, он не предполагает дополнительных действий пользователя. Пользователь может не оплатить билет или оплатить, но потом отменить и прочее. Эти состояния также нужно отразить на диаграмме.



Так как пользователь может забронировать билет, но не оплатить его, бронь отменится, когда время на оплату истечёт. В таком случае нужно добавить в диаграмму состояние «Отменено по таймеру». А также «Отменено клиентом», в которое система может перейти из трёх предыдущих: «Билет забронирован», «Билет оплачен», «Билет получен». При переходе из «Билет забронирован» в «Отменено клиентом» пользователю достаточно просто произвести событие «Отмена». Если билет уже был оплачен, действием системы должен стать возврат денежных средств — «Возврат». Предположим, что для перехода системы из состояния «Билет получен» в «Отменено клиентом» помимо отмены билета пользователь должен выполнить условие перехода — позвонить в авиакомпанию. Это тоже нужно отразить на диаграмме. Условие перехода указывается над стрелкой в квадратных скобках.

Такая визуализация актуальна для сложных проектов с множеством состояний, переходов и условий для них. Она позволяет не пропустить важные звенья системы и наиболее полно описать тестовые сценарии для проверки.

Например, первым сценарием может стать проверка полного цикла работы системы от входа в неё до точки выхода. Затем можно выполнять тестирование более детально, добавляя новые сценарии на основании диаграммы переходов и состояний.

При построении диаграмм состояний и переходов важно:

● не допускать пересечения линий переходов — это усложняет визуальное восприятие диаграммы и может привести к ошибочному переходу;

● сложные процессы лучше представлять в виде нескольких диаграмм — если охватить всё одной схемой, она будет слишком трудной для понимания;

● главную последовательность состояний следует размещать на одной горизонтальной линии, чтобы прослеживался позитивный сценарий работы системы. Дополнительные состояния можно представить в виде ответвлений и разместить по бокам от основной последовательности.

**Попарное тестирование.**

Попарное тестирование считается самым сложным и запутанным из отобранных пяти техник тест-дизайна. Попарное тестирование основано на математических алгоритмах, а именно на комбинаторике. Оно позволяет создавать уникальные пары и тестировать огромное количество поступающих данных в разных сочетаниях, но расчеты могут быть сложными.

Чтобы охватить тестовыми сценариями максимум фич и при этом потратить минимальное время на тестирование, нужно правильно сопоставлять данные, комбинируя пары определенным образом на основе расчетов.

Пример попарного тестирования. Допустим, есть сеть пекарен, продающих яблочные пироги и чизкейки онлайн. Каждый товар доступен в трех размерах – маленьком, среднем и большом. Пекарня предлагает доставку, как на ближайшее время, так и к определенному времени, а также возможность самовывоза. Пекарня работает в трех городах – Красноярск, Сосновоборск и Дивногорск. Также пользователь может заказать до трех товаров одновременно.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Заказ** | **Размер** | **Город** | **Количество** | **Доставка** | **Время** |
| Яблочный пирог | Маленький | Красноярск | 1 | Адресная | Ближайшее |
| Чизкейк | Средний | Сосновоборск | 2 | Самовывоз | К указанному времени |
|  | Большой | Дивногорск | 3 |  |  |

Если вы захотите протестировать все возможные варианты инпута, у вас будет 2x3x3x3x2x2=216 комбинаций. Но проверять каждую нет смысла. Вместо этого вы можете выстроить переменные таким образом, чтобы охватить максимум сценариев.

Для этого вам нужно сгруппировать переменные или использовать какой-нибудь инструмент, который сделает это за вас. Например, воспользовавшись Pairwise Tool, мы получили 17 сценариев, способных охватить все 216 комбинаций.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Заказ** | **Размер** | **Город** | **Коли-чество** | **Доставка** | **Время** |
| 1 | Яблочный пирог | Большой | Дивногорск | 3 | Адресная | Ближайшее |
| 2 | Чизкейк | Большой | Красноярск | 2 | Адресная | К указанному времени |
| 3 | Чизкейк | Маленький | Сосновоборск | 1 | Самовывоз | Ближайшее |
| 4 | Чизкейк | Средний | Красноярск | 2 | Адресная | К указанному времени |
| 5 | Чизкейк | Маленький | Сосновоборск | 3 | Самовывоз | Ближайшее |
| 6 | Яблочный пирог | Большой | Сосновоборск | 2 | Самовывоз | Ближайшее |
| 7 | Яблочный пирог | Маленький | Красноярск | 3 | Адресная | К указанному времени |
| 8 | Яблочный пирог | Маленький | Дивногорск | 2 | Самовывоз | К указанному времени |
| 9 | Яблочный пирог | Средний | Красноярск | 1 | Адресная | Ближайшее |
| 10 | Чизкейк | Средний | Дивногорск | 1 | Самовывоз | К указанному времени |
| 11 | Чизкейк | Средний | Сосновоборск | 3 | Адресная | К указанному времени |
| 12 | Чизкейк | Большой | Красноярск | 1 | Самовывоз | Ближайшее |
| 13 | Яблочный пирог | Средний | Красноярск | 3 | Самовывоз | Ближайшее |
| 14 | Яблочный пирог | Маленький | Сосновоборск | 1 | Адресная | К указанному времени |
| 15 | Яблочный пирог | Средний | Красноярск | 2 | Самовывоз | Ближайшее |
| 16 | Яблочный пирог | Большой | Сосновоборск | 1 | Адресная | К указанному времени |
| 17 | Яблочный пирог | Маленький | Дивногорск | 2 | Адресная | Ближайшее |

**Таблицы принятия решений.**

Часто аналитики создают требования в виде сплошного текста с множеством условий вида «если …, то ...». Например, «если пользователь старше 16 лет, то доступ на сайт разрешён», «если пользователь авторизован в системе, то его личные данные в форме заказа должны быть заполнены автоматически».

Тестирование таких требований и создание на их основе тест-кейсов трудоёмкое, нужно повышенное внимание. Для таких случаев можно использовать технику тест-дизайна «Таблицы принятия решений».

Таблицы принятия решений (таблицы решений) — способ компактно представить модели со сложной логикой. А ещё это техника тестирования чёрного ящика, которая применяется для систем со сложной логикой. Таблицы принятия решений используют, чтобы упорядочить и задокументировать сложную логику приложения, а также протестировать все комбинации условий и состояний.

Рассмотрим сущности, из которых состоят таблицы.

**Условия –** короткое описание входных условий (данных), сформулированное в виде вопроса. Ответ — либо «да/нет», либо ограниченный набор значений.

**Действия –** чёткое описание ожидаемого результата, действия системы. Формулировка действия — утвердительное предложение. Одно предложение обязательно описывает только одно действие.

**Значения –** значения, допустимые для входных данных, указанных в условии.

**Правила** – комбинации входных данных, которые отражены в таблице.

Начнём составлять таблицу по плану:

1. Разбить требование на условия.

2. Посчитать количество возможных правил (комбинаций).

3. Составить таблицу принятия решений.

4. Исключить лишние комбинации, если они есть.

5. Создать тесты.

**Примеры таблиц принятия решений:**

Допустим, тестировщик проверяет работу формы выдачи кредита на сайте банка. Из требований он знает, что кредит выдаётся со следующими условиями:

* На момент рассмотрения заявки человеку больше 18 лет.
* На момент рассмотрения заявки человеку меньше 55 лет.
* Если ежемесячные выплаты меньше трети ежемесячного дохода, то стандартный процент, иначе – +1%.
* Кредит не выдаётся безработным.

У тестировщика есть четыре условия и два правила — «да» и «нет». Это значит, что количество столбцов будет рассчитываться через степень двойки: 24 = 16.

Первые два условия можно оптимизировать, так как они оба касаются возраста. Тогда получится три условия, а количество столбцов сократится вдвое.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Условия** | **Значе**  **ния** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **Возраст от 18 до 55 лет** | да, нет | да | да | да | да | нет | нет | нет | нет |
| **Ежемесячная выплата кредита меньше 1/3 ежемесячного дохода** | да, нет | да | да | нет | нет | да | да | нет | нет |
| **Человек трудоустроен** | да, нет | да | нет | да | нет | да | нет | да | нет |
| **Действия** |  | **Кредит с обычной ставкой** | **Кредит не выдаем** | **Выдаем кредит с увеличенной на 1 % ставкой** | **Кредит не выдаем** | **Кредит не выдаем** | **Кредит не выдаем** | **Кредит не выдаем** | **Кредит не выдаем** |

**Преимущества и недостатки метода**

Таблица принятия решений в тестировании не единственный инструмент. Она не универсальна, но нельзя сказать, что это недостаток. С помощью таблицы не получится проверить корректность ввода логина или пароля — потому, что условий может быть слишком много. Придётся вносить в таблицу миллионы комбинаций из букв, цифр и специальных символов, и всё это с учётом регистра. Это только усложнит работу тестировщика, поэтому для проверки полей ввода обычно используют другие инструменты, например, классы эквивалентности и граничные значения.

Таблица не подойдёт для тестирования линейных процессов — например, экранов приложения, где пользователь может нажать только одну кнопку вроде «Согласен» или «Принять».

Несмотря на ограничения, такая техника, как таблица принятия решений при правильном использовании имеет больше преимуществ, чем недостатков.

**Глава 2.4. Что такое баг-репорт и как его составить.**

Во время тестирования тестировщик записывает свои наблюдения, найденные баги и другую полезную информацию, полезную разработчикам и менеджменту. Вся эта информация — баг-репорт.

**Баг-репорт** - это документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

**Баг** – это некорректная работа ПО. Причиной является ошибка в коде, которую допустил разработчик в процессе его написания.

**Структура баг-репорта.**

Детальный баг-репорт — важный итог, полученный в результате тестирования. Он содержит следующую информацию(атрибуты):

1. **Заголовок**. Краткий заголовок, описывающий проблему в общем. Не должен быть слишком длинным, но должен быть понятным и точным.
2. **Описание**. Как правило, повторяет заголовок баг-репорта, так что это поле иногда пропускают.
3. **Версия**. Production-версия продукта, в которой был найден баг. Можно потратить много времени, пытаясь воспроизвести ошибку на других версиях (в которых, возможно, ее просто нет)
4. **Приоритет**. Степень важности устранения дефекта.
5. **Серьезность.** Степень влияни дефекта на работу системы.
6. **Статус.** В каждый момент времени баг может быть в одном из состояний: «Active» (активный), «Ready for Testing» (готов к тестированию), Closed (закрыт). Баг становится «активным», когда он найден; готов к тестированию — когда разработчик его исправляет. Тестировщик может перевести его в состояние «Closed» после исправления и повторного тестирования.
7. **Окружение.** Информация об окружении, на котором был найден баг: операционная система, сервис пак, для WEB тестирования - имя и версия браузера и т.д.
8. **Шаги воспроизведения**. Перечисление действий, которые необходимо выполнить, чтобы воспроизвести найденный баг.
9. **Фактический результат.** Результат, полученный после прохождения шагов к воспроизведению.
10. **Ожидаемый результат**. Ожидаемый правильный результат.
11. **Вложения.** Файл с логами, скриншот или любой другой документ, который может помочь прояснить причину ошибки или указать на способ решения проблемы.
12. **Дополнительная информация.** В этом поле могут быть указаны пояснения от тестировщика (что он еще делал для воспроизведения бага, и каковы были результаты).

Баг-репорт представляет собой технический документ, следовательно, его необходимо формировать в техническом стиле. Не должно быть никаких художественных элементов и размытых формулировок. Чтобы сделать все максимально правильно, нужно пользоваться шаблонами, которые приняты в конкретной компании. Несколько важных нюансов, которые необходимо учесть при составлении данного документа:

* **Размер заголовка**. Эта часть баг-репорта помогает разработчику быстро понять суть проблемы. Заголовок не должен быть слишком длинным или коротким.
* **Локализация**. Тестировщик должен не просто найти баг, но еще и правильно его описать. В противном случае разработчики будут решать проблему, которая не имеет никакого отношения к их обязанностям.
* **Вложения**. Если баг визуальный или UX (к примеру, нашлась некорректная вёрстка или не функционирует кнопка), то для наглядности необходимо сделать скриншоты. Так разработчик поймет, что видит пользователь при возникновении ошибки.
* **Шаги воспроизведения**. Не стоит начинать с этапа «Включить компьютер». При этом шаги должны быть описаны максимально точно, без размытых формулировок.
* **Взгляд на проблему**. Специалист должен поставить себя на место заказчика. К примеру, если текст не помещается в поле, то такая ошибка мешает бизнесу? Это необходимо, чтобы правильно оценить серьезность и приоритет дефекта.
* **Фактический и ожидаемый результаты**. Если тестировщик опишет проблему непонятно, то разработчику придется задать ему уточняющие вопросы. К примеру, фактически результат— кнопка не работает, а ожидаемый результат — кнопка работает. Такое описание не дает разработчику никакого понимания сути проблемы.

Обычно после того как баг был обнаружен, он попадает на стадию «Активный». Когда все работы по исправлению будут завершены, он получит статус «Закрыт». Однако между этими стадиями есть еще четыре промежуточных **этапа**, к которым может быть отнесен баг:

* **Отклонен**. Эта стадия нужна для багов, которые не получилось воспроизвести из-за неточностей в «Шагах воспроизведения» или повторном попадании дефекта в категорию «Новый».
* **Открыт**. Если баг требуется исправить в кратчайшие сроки.
* **Исправлен**. Сюда относят уже исправленные ошибки.
* **Переоткрыт**. Если ранее баг был отсрочен или отклонен, но потом решение поменялось.

**Степени серьезности и приоритетов в баг-репортах.**

**Степень серьезности в баг-репорте** (**Severity**) — это атрибут, который характеризует уровень критичности (влияния) ошибки для ПО. Иными словами, данный параметр указывает на масштабность изменений в программе, произошедшими из-за ошибки в коде.

Есть пять базовых степеней серьезности:

1. **Блокирующий(Blocker)**. Вся программа не сможет функционировать, если ошибку не исправить.
2. **Критический(Critical)**. Большинство составляющих программы работает некорректно.
3. **Значительный(Major)**. Ошибка мешает функционированию одной из основных логических цепочек ПО. Программа работает, но при этом задача не решается с помощью предназначенного для этого способа.
4. **Незначительный(Minor)**. Основные логические цепочки программы работают нормально. ПО можно использовать с незначительными потерями качества.
5. **Тривиальный(Trivial).**Вся программа будет нормально работать, даже если ошибку не исправлять.

**Приоритет (Priority)** - это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.

1. **Высокий (High)** Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.
2. **Средний (Medium)** Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.
3. **Низкий (Low)** Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

Порядок исправления ошибок по их приоритетам: **High -> Medium -> Low**

**Основные ошибки при написании багов репортов.**

Тестировщик должен иметь определенные знания и навыки, чтобы правильно формировать баг-репорты.

* **Неправильно составлен заголовок.** Он должен отвечать на вопросы «Что? Где? Когда?». В загаловке не должны быть лишние сведения (версии, окружения, учетные данные пользователей ).
* **Недостаточность предоставленных данных.** Не всегда одна и та же проблема проявляется при всех вводимых значениях и под любым вошедшим в систему пользователем, поэтому настоятельно рекомендуется вносить все необходимые данные в баг репорт
* **Определение серьезности.** Очень часто происходит либо завышение, либо занижение серьезности дефекта, что может привести к неправильной очередности при решении проблемы.
* **Язык описания.** Часто при описании проблемы используются неправильная терминология или сложные речевые обороты, использование «жаргона», которые могут ввести в заблуждение человека, ответственного за решение проблемы.
* **Отсутствие ожидаемого результата.** В случаях, если вы не указали, что же должно быть требуемым поведением системы, вы тратите время разработчика, на поиск данной информации, тем самым замедляете исправления дефекта.
* **Нет скриншота / видеозаписи для UI/UX багов.** Либо на скриншоте не выделен дефект.

Если начинающий тестировщик будет заранее знать об этих часто встречаемых ошибках, то он сможет написать более качественный баг-репорт. При составлении баг-репорта многие новички слишком сосредоточены на деталях. Нужно стараться идти от частного к целому. Это позволяет более объективно рассмотреть проблему. Необходимо понять, какое влияние баг оказывает на процессы, функциональность и удобство пользователя.

**Глава 3. Проведение исследовательского тестирования веб-приложения и составление Технического Задания программного продукта, с учетом ожиданий заказчика (владельца сайта)**

**Глава 3.1. Особенности Исследовательского тестирования**

**Исследовательское тестирование** – это одновременное изучение программы, проектирование и выполнение тестов. Этот подход – противоположность сценарного, когда список проверок составлен до проведения тестирования. Исследовательские тесты не определены заранее и не выполняются в точном соответствии с планом. Исследовательское тестирование – это не методика тестирования. Это подход или образ мыслей, который можно применить к любой методике. Исследовательское тестирование не должно выполняться небрежно, в спешке. Такой подход требует тщательной подготовки, а знания и умения тестировщика – важная форма этой подготовки. Исследовательское тестирование может проводиться как вручную, так и со вспомогательными инструментами.

**Когда применять исследовательское тестирование?**

● Нужна быстрая обратная связь о новом продукте.

● Нужно быстро изучить продукт.

● Сценарное тестирование не находит баги, требует разнообразия.

● Нужно принять решение о необходимости покрытия области сценарными тестами.

● Требований нет, они неполные или устарели.

● Продукт маленький, разработка тестовых сценариев займёт больше времени, чем сам процесс тестирования.

**Преимущества исследовательского тестирования:**

1. Нестандартные ходы выявляют нестандартные дефекты.

2. Не тратится время на описание всех сценариев.

3. Не нужна поддержка тестовых сценариев.

4. Не наступает «эффект пестицида».

5. Можно тестировать без требований.

6. Тесты могут стать интереснее и креативнее.

Чистое исследовательское тестирование хорошо работает на небольших краткосрочных проектах или на начальных этапах жизни продукта. В первом случае объёмная тестовая документация может не оправдать времени, затраченного на её написание. Во втором – требования могут быть очень размытыми, так как нет определённости, в каком направлении будет двигаться разработка. Функциональность будет меняться, так что потребуется постоянная актуализация тест-кейсов.

**Жизненный цикл исследовательского тестирования**

В процессе исследовательского тестирования большое внимание уделяется свободе тестировщика, но при этом соблюдается определенная структура.

**Этап 1: Обучение.** Тестировщики начинают с глубокого понимания программного обеспечения и его функциональности – критического анализа, чтобы определить, как оно сочетается друг с другом. Это позволяет тестировщику выяснить, какие обычные вводы может сделать пользователь, хотя он, возможно, уже знаком с приложением и его функционированием.

На этапе обучения может даже потребоваться учебник по работе с программным обеспечением. Это этап исследования, на котором тестировщик получает всю необходимую информацию для разработки широкого спектра полезных тестов.

**Этап 2: Разработка тестов.** Разработка исследовательских тестов действительно включает в себя различные правила и параметры, но все же предлагает значительно больше свободы по сравнению со сценарным тестированием, специфика которого уже известна до начала тестирования. Тестировщик может разработать проверки, которые, по его мнению, более точно соответствуют приложению, и потенциально может обнаружить ценные данные для команды разработчиков, включая заметные ошибки, которые они должны исправить.

Команды тестирования используют этот этап, чтобы определить, какого подхода придерживаться и как распределить работу между различными тестировщиками таким образом, чтобы использовать их сильные стороны.

**Этап 3: Исполнение.** После разработки используемых проверок, тестировщики теперь могут проверять приложение теми способами, которые они считают наиболее эффективными – они могут проводить их сразу после разработки конкретной проверки. Это этап, на котором тестировщики активно ищут проблемы и то, как обнаруженные ими проблемы могут быть использованы в других функциях и возможностях. Хотя при проведении исследовательских тестов в определенной степени присутствует интуиция, они все же следуют установленным процессам и целям, что позволяет проводить плавное тестирование, которое легко может соответствовать конкретным целям тестирования.

**Виды исследовательского тестирования.**

1. **Вольное исследовательское тестирование.** Это специальный подход к проверке приложения.

При этом учитывается мало правил, поэтому его эффективность может варьироваться; некоторые программы и компоненты требуют более надежной методики. Эти проверки могут принести много пользы, помогая тестировщикам ознакомиться с данным приложением и подтвердить работу предыдущего тестировщика. Даже при отсутствии строгих правил опытные и квалифицированные тестировщики могут легко использовать это в своих интересах.

2. **Исследовательское тестирование на основе сценариев**

Тестирование на основе сценариев использует реалистичные ситуации в качестве основы для каждого теста, например, проверяя вводы, которые пользователи, скорее всего, сделают во время типичной работы данного программного обеспечения. Тестировщики усердно работают над тем, чтобы каждый разработанный ими сценарий соответствовал тому, как пользователь взаимодействует с приложением. Время может стать ограничением, поскольку цель команды – протестировать как можно больше сценариев; в зависимости от предстоящих сроков, скорее всего, не удастся охватить все возможности.

3. **Исследовательское тестирование на основе стратегии.**

Тестирование на основе стратегии включает в себя широкий спектр специальных методов, в том числе тестирование граничной стоимости, методы эквивалентности, методы, основанные на риске, и многое другое. В этом случае предпочтение обычно отдается тестировщикам, которые уже знакомы с приложением, поскольку они могут разработать индивидуальные стратегии, включающие эти отдельные методы. Подход, основанный на стратегии, фокусируется в основном на функциональности (и внутренней работе) программного обеспечения, не рассматривая возможные сценарии, которые могут привести пользователя к возникновению проблем. Это может привести к более широкому анализу приложения и его различных особенностей, потенциально более глубокому, чем при различных других подходах.

Для управления исследовательским тестированием может использоваться:

● Тестовые сессии – ограниченные промежутки времени, в рамках которых происходит тестирование‎. При этом каждая сессия имеет тему.

● Отчеты по результатам сессий, оформленные в пригодной для парсинга и сбора статистики форме.

● Обсуждение лидом и тестировщиком результатов проведенной сессии.

**Идея туров в исследовательском тестировании**

Чтобы систематизировать исследовательское тестирование можно использовать идею туров Джеймса Виттакера. Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

**Глава 3.1.1. Идея туров в исследовательском тестировании. Туры Джеймса Виттакера.**

Джеймс Виттакер, хоть и не придумал саму идею туров, но предложил свой подход к исследовательскому тестированию с использованием туров и в своей книге “Exploratory Software Testing” в доступной форме озвучил идею туров и описал сами туры.

Тур – это своего рода план тестирования, он отражает основные цели и задачи, на которых будет сконцентрировано внимание тестировщика во время сессии исследовательского тестирования. При этом Виттакер использует метафору, что тестировщик – это турист, а тестируемое приложение – это город. Обычно у туриста (тестировщика) мало времени, поэтому он выполняет конкретную задачу в рамках выбранного тура, ни на что другое не отвлекаясь. Город (ПО) разбит на районы: бизнес-районы, исторический район, район развлечений, туристический район, район отелей, захудалый район.

**Туры.**

1. **Туры по бизнес-центру**. Это места, где «делается бизнес». Они начинаются с запуска кода и до завершения его работы. В них расположены функции, ради которых пользователи и используют данное приложение. Это «задняя стенка коробки», которая является главной при демонстрации коммерческой ценности продукта, а также код, который это поддерживает.

* **Тур по путеводителю**. Путеводители часто содержат много информации о местах, которые можно посмотреть. Они рассказывают о лучших отелях, торговых комплексах и привлекательных местах без подробных деталей. Эксперты, посетившие эти места, любезно скажут туристам, как именно нужно наслаждаться этими местами. Здесь должно быть красиво, чисто, безопасно, чтобы туристы могли спокойно тратить свои деньги. Это очень важные места, и они должны быть включены в тестовую стратегию исследовательского тестирования. Мы должны заботиться о том, чтобы пользователи получали пользу и удовольствие от работы здесь.

Вариантом такого тура является строгое следование руководству пользователя, предоставленное (если предоставлено) производителем, в котором даны описания ключевых функций. Цель в том, чтобы пройти все сценарии, которые описаны в руководстве пользователя, так полно, как это возможно. Зачастую здесь можно найти различные советы по входным данным и навигации по интерфейсу в процессе использования функций. Помимо самой функциональности этот тур позволяет проверить и то, насколько точно и аккуратно составлено руководство пользователя.

Разновидностями тура также являются тур блогера, который содержит советы третьей стороны, а также тур эксперта, который рассматривает отзывы неудовлетворенных пользователей. Этому могут помочь различные форумы, новости, сообщества или даже книги (в случае, если приложение большое). Еще один полезный тур – тур конкурента, который позволяет рассмотреть приложение со стороны конкурирующей системы.

Вышеуказанные варианты тура по путеводителю позволяют оценить приложение в разрезе его разрекламированных функций. Это незамысловатые тесты, и любое отклонение от руководства должны воспринимать как ошибка. Указанный тур позволяет нам использовать функции именно в такой последовательности и взаимосвязи, в какой это будут делать сами пользователи.

Типичные баги: несоответствия описанных шагов в справке работе приложения; несоответствия скриншотов в справке и самого приложения; ошибки (от орфографических до логических) в справке; ошибки удобства использования справки: длинная или короткая, непонятная или примитивная, не отвечающая на те вопросы, которые предполагают прочитать пользователи; ошибки удобства использования приложения.

* **Денежный тур**. Каждое приложение имеет функциональность, которая, в глазах пользователя (заказчика), принесет им деньги. «Это самое привлекательно для меня» — скажет в этом случае пользователь. «Денежная» функциональность несет с собой «денежную силу». Сюда входят различные нюансы использования продукта, поскольку именно они позволяют зарабатывать деньги. Специалисты по продажам могут направлять демонстрацию продукта в сторону указанных сценариев, затрагивая то, что не было четко зафиксировано в документации к продукту. Такое «продажное видение» – отличная основа для денежного тура.

В указанном туре важно услышать то, как специалисты по продажам преподносят продукт, и пройтись именно по тому, на чем ставится акцент. Во время подобного тестирования вы не только находите ошибки, но и сохраняете видение «денежной жилы» приложения.

Разновидность денежного тура – тур скептически настроенного заказчика. Проходясь по туру, вы внезапно останавливаетесь и спрашиваете: «А что, если я сделаю вот так?» или «А как мне сделать это?». «А что, если я сейчас дополню проход такими сценариями, которые выходят за рамки изначально задуманных? Возьму да включу сюда еще парочку фич!».

Типичные баги: напрочь отсутствие в приложении некоторых описанных в промо-материалах возможностей; несоответствие информации в рекламных материалах и работы самого приложения; функциональные ошибки в работе приложения, влияющие на демонстрацию; устаревшие скриншоты приложений; лингвистические ошибки (от орфографических до логических).

* **Тур по отметкам**. Мы выбираем какие-либо ориентиры и осуществляем скачок к одному из них через все приложение. Например, мы выбираем ключевую функцию в качестве отметки, которую мы определили в туре по путеводителю или в денежном туре. Далее мы составляем набор из таких отметок, определяем их порядок и исследуем приложение от отметки к отметке, пока не достигнем каждой из них. Далее можно создать карту покрытия отметками. Разный порядок, разные отметки – вот поле для вариативности подобного тура.

Типичные баги: блокирующие, критические проблемы в работе приложения.

* **Интеллектуальный тур**. Задавайте приложению сложные, интеллектуальные вопросы. Как можно заставить работать приложение так тяжело, как это возможно? Какие фичи приблизят его к собственному лимиту? Какой ввод и входные данные заставят его напрягаться? Какие данные могут обмануть процедуры проверки ошибок? Очевидно, это напрямую зависит от особенностей конкретного приложения. Задавай глупые вопросы, старайся насолить, привлекай к себе внимание. Придумывайте препятствия и посмотрите, как с ними справится приложение. Имеет ли это смысл? Возможно и нет. Но ведь мы можем так сделать.

Интеллектуальный тур и его варианты могут помочь найти как действительно сложные и приоритетные ошибки, так и глупые ошибки. Нужно уметь различать их. Для этого необходимо создавать реальные случаи, чтобы в будущем было легче аргументировать то, что это важно и это нужно исправлять.

Типичные баги: креши приложения при выходе за граничные значения; автоматические прерывания текущих процессов, изменение состояние приложения; недоработки в логике; низкая производительность, медленная работа приложения; потеря данных; примитивные ошибки в удобстве использования (например, в файловом менеджере мало места отведено под название документа и нет возможности его просмотреть другим способом).

* **Тур службы доставки**. Думайте о данных, которые должны проходить через приложение. Данные начинают свой путь с ввода, далее хранятся во внутренних хранилищах и переменных, где часто происходит их обработка и изменение, а далее они могут подвергаться вычислениям. В конце большинство из таких данных предоставляются пользователю или куда-либо еще. Нужно концентрироваться на данных. Нужно определить сохраненные данные и «проследить» их путь через приложение. Вот ввели вы, к примеру, адрес своей почты. А где он отображается? В каких местах? Попробуйте найти все области, с которыми соприкасаются данные. Определите весь их жизненный цикл.

Типичные баги: совершенно разные – от критических функциональных до багов в удобстве использования, интерфейсе, локализации (поддержка форматов разных стран).

* **Тур «после работы», внеурочный тур**. После денежной активности приложения, оно все еще продолжает работу. Это поддержка задач, архивация данных и файлов. Это может быть автоматическим процессом, но его также можно выполнять принудительно. Данный тур напоминает нам делать это. То, что пользователь перестает использовать приложение, не обозначает, что приложение останавливает работу. Часто приложение бэкапит данные, архивирует, получает обновления и т.д. Тестировщик должен проследить за этими операциями.

Типичные баги: проблемы неэффективного использования ресурсов (нерациональное использование трафика, места на диске, процессора, утечки памяти и т.п.), функциональные ошибки иногда приводящие к остановке работы приложения.

* **Тур уборщика**. Для тестировщика этот тур похож на выборочную проверку, в процессе которой он по кратчайшему пути должен методично переходить с экрана на экран, вызывать диалог за диалогом и проверять какую-то простую примитивную вещь, не углубляясь особо в детали. Например, наличие и правильность заголовков на всех окнах, зеленый цвет кнопки сохранения везде, где она есть, наличие метки с количеством комментариев к статье везде, где эта статья как-то упоминается и т.д.

Указанный тур включает выбор цели (все меню, ошибки, диалоговые окна, к примеру) и посещение каждой из них наиболее коротким путем.

2. **Туры по историческому району.**

Это излюбленные туристами исторические места, места, окруженные историей. Сюда включены места, связанные с предыдущей версий, со старым функционалом или кодом, которые исторически содержали большое количество ошибок. Его использование и изменение подразумевает большое количество предположений. Именно на этом компоненте и сосредотачивается данный тур.

* **Тур по нерекомендуемым местам**. В приложении – это фичи или места в коде с наибольшим скоплением багов. Про эти места вы можете знать интуитивно, а можете воспользоваться багтрекером. Как только вы нашли один баг, знайте, что, скорее всего, рядом есть и другой; он может находиться в том же функционале, а может и в соседнем. Задача тестировщика состоит в том, чтобы пройти по местам. Связано это с тем, что баги имеют свойство скапливаться в одном месте. Тур также применим и после исправления багов. Пройдитесь взаимосвязанным областям.

Типичные баги: функциональные.

* **Музейный тур**. Код тоже может быть “античным”. Это тот код, который давно не изменялся. Такой код, попадая в новую среду, может вообще не работать, или, подвергаясь внешнее ревизии, может оказаться непригодным. Тестировщики могут найти такой код по метке даты изменения в репозитории. Например, если создают для iOS новый, но схожий с другим проект, то могут скопировать лишний код или ресурсные файлы, которые при проверке Apple ревьюверами может привести к отклонению приложения.

Типичные баги: креши, функциональные ошибки, несоответствие стандартам или гайдлайнам, увеличение размера приложения.

* **Тур предыдущей версии**. После обновления приложения пользователь должен легко и быстро адаптироваться к новому функционалу, или новым путям использования приложения. Особенно это важно проверять, когда вырезается какой-то функционал, изменяется интерфейс или даже исправляется очень старый баг, который некоторые пользователи могли использовать как фичу.

Типичные баги: ошибки юзабилити, функциональные ошибки, связанные с утратой функциональности, потерей данных, ошибками в логике.

3. **Туры по туристическому району**.

Во многих городах есть места, которые интересны только туристам и пользователям-новичкам. Местные ребята избегают таких мест, поскольку им там тесно. То есть (прим. для ПО) пользователи, которые уже имеют опыт взаимодействия с приложением, подобные функции почти не используют. Быстрые проверки, с целью просто пробежаться по функциям. Мол, “я здесь был”.

* **Тур коллекционера**. Собирайте всевозможные артефакты вашего приложения – то, что пользователь может сохранить себе, оставить “на память”; это всё, что приложение “отдает” пользователю. Например, в браузере – это история, закладки, посещенные страницы, в файловом менеджере – файлы разных форматов, папки всевозможные и т.д.

Типичные баги: в основном, некритичные функциональные ошибки.

* **Тур одинокого бизнесмена**. Доберитесь до фичи, которая находится глубже всего, дальше всего от старта приложения. Или же выберите какую-то фичу и доберитесь до неё самым длинным путем, какой сможете придумать.

Типичные баги: разные – от функциональных (чаще некритических), до багов в удобстве использования, интерфейсе, локализации, производительности.

* **Тур супермодели**. Этот тур о внешнем интерфейсе приложения: красив ли, привлекателен ли, правильно ли используются цвета, нет ли лишних артефактов в интерфейсе, интересная и быстрая ли анимация, соответствует ли интерфейс принятым стандартам, ожиданиям пользователя. Приложение может содержать много багов, но и как супермодель на подиуме, оно будет смотреться великолепно!

Типичные баги: проблемы интерфейса и удобства использования.

* **Тур шопоголика**. Необходимо создать ситуацию, в которой несколько копий одного и того же приложения используют один и тот же объект, одну и ту же фичу в один момент времени. Например, запустить несколько приложений и обращаться к одному файлу или разделу виртуальной памяти; залогиниться несколькими пользователями в разные приложения; удалять и редактировать одновременно одну и ту же статью; по одной сети отправлять на сервер конфликтующие данные в один момент времени и т.д.

Типичные баги: недоработки в логике, приводящие как к некритичным, так и к критичным багам; непонятные для пользователя сообщения об ошибках (формата “error 784”); проблемы производительности.

* **Тур по шотландским пабам**. На форумах, в блогах, при общении с пользователями, да и просто блуждая по приложению можно найти неожиданные для себя, как тестировщика, фичи, но те, которые полюбились настоящему ценителю-пользователю. Этот тур наиболее хорош для очень больших приложений. Его задача состоит не только в том, чтобы проверить, но и в том, чтобы глубже познакомиться с приложением (в идеале, посредством общения с пользователем).

Типичные баги: совершенно разные.

4. **Туры по району развлечений**

В большинстве приложений есть места, где можно отвлечься от основной бизнес-задачи (сценария использования). Например, настроить приложение под свои особенные нужды, или навести “красоту” изменяя шрифт и цвет текста и т.п.

* **Тур актера второго плана**. Выполняя этот тур, используйте, изучайте, проверяйте не ключевой функционал, а тот, что его окружает – не главный, не столь заметный, но такой, который найдёт своего пользователя.

Типичные баги: разные, но больше интерфейс, лингвистика или некритичные функциональные ошибки.

* **Тур по темным переулкам**. Это путешествие по фичам, которые маловероятно, что будут использоваться, по фичам, наименее привлекательным для пользователей. Если у тестировщика есть статистика по использованию приложения, то в этом туре он может следовать по фичам из самых нижних позиций рейтинга.

Типичные баги: некритичные функциональные ошибки; определение мест и фич, назначение которых пользователь вообще не понимает без подсказки или справки; ошибки юзабилити: примеры странного, нелогичного поведения; ошибки лингвистики; ошибки интерфейса.

* **Тур любителя ночной жизни.** Сколько сможет выдержать ваше приложение без перезагрузки? Никогда не перезапускайте приложение! Используйте одну и ту же фичу постоянно, повторяйте одно и то же действие раз за разом. Примеры: открыть файл и никогда его не закрывать, соединиться с сервером и никогда не разрывать соединение, запустить приложение и никогда его не перезапускать.

Типичные баги: проблемы производительности и эффективности такие как утечки памяти, медленная скорость работы; потеря данных; функциональные ошибки.

5. **Туры по району отелей.**

В каждом туристическом городе должны быть места, где можно отдохнуть, восстановить силы, переждать непогоду. Но вот ПО в моменты отдыха пользователя совсем не отдыхает. И хорошо бы проверить что и как оно делает. Место, где можно отвлечься от основной функциональности и популярных фич, и проверить что-то второстепенное.

* **Тур, отмененный из-за дождя.** Отменяйте любой начавшийся процесс; отменяйте любым способом – кнопкой “Cancel”, закрыть приложение (через Alt+F4, диспетчер задач), закрыть вкладку или окно Ctrl+F4, нажать ESC, в браузере остановить загрузку или перейти на страницу другую страницу или нажать кнопки “Вперед”, “Назад”, “Обновить”, “Остановить”.

Задача тестировщика:

1. убедиться, что отмена в целом возможна;
2. что отмена действия не вызывает проблем в работе приложения;
3. что данные не теряются и не повреждаются (здесь лучше смотреть на приложение не только “снаружи”, через UI, но и изнутри – через структуру приложения, через БД);
4. что действие, которое мы отменяем, можно вызвать повторно и успешно его выполнить.

Типичные баги: критичные и некритичные функциональные ошибки; проблемы целостности данных; юзабилити ошибки о невозможности отменить процесс.

* **Тур лежебоки**. Идите по “дефолтному” пути: не изменяйте значений, установленных по умолчанию, не заполняйте поля, если в них нет строгой необходимости, не переходите по рекламе, пытайтесь перейти между экранами, не нажимая кнопок и не вводя данных, не беспокойтесь о части ELSE в конструкции IF-THEN-ELSE, не совершайте комплексных действий – идите по пути “наименьшего сопротивления”.

Типичные баги: проблемы юзабилити; функциональные ошибки.

6. **Туры по неблагополучному району.**

Это небезопасные район, которые, однако, для кое-кого представляют некоторый интерес. Там творятся плохие вещи, и, видимо, будет лучше не лезть туда. Всем, кроме тест-инженеров. Эти места должны быть протестированы, поскольку они могут содержать уязвимости продукта. Места приложения, подверженные атакам недобропорядочных пользователей.

* **Тур саботажника, диверсанта.** Задача тестировщика – попробовать подорвать (подстрекнуть) работу приложения любым возможным способом.

И для этого можно выполнить 4 главных шага:

1. Начните выполнять какое-то действие.

2. Определите какие ресурсы нужны системе, чтобы выполнить это действие.

3. Удалите или ограничьте системе доступ к этим ресурсам.

4. Повторите действие.

Примерами условий для тестов могут быть: отсутствие интернета, недостаточное количество оперативной памяти, удаленный файл, который нужно прочитать, отсутствие необходимых для выполнения операции прав, поиск или запрос несуществующего объекта, запуск и работа приложения на изначально проблемном окружении, подкладывание в систему битых файлов или данных, параллельная работа с другим приложением за конфликтующие ресурсы.

Типичные баги: креши; непонятные для пользователя по формулировке сообщения об ошибках; функциональные ошибки средней критичности.

* **Антисоциальный тур.** Работать так, как нормальные пользователи делать не будут, выполнять всё, что противоречит логике приложения, вводить данные, которые должно быть запрещено вводить.

Антисоциальный тур подразделяется на 3:

1. **Противоположный тур**. Предполагает вводить или выбирать те значения, которые пользователь маловероятно, что выберет. Например, в плейлист добавить 10763 песен, или отправить на печать 12 копий. В этом туре речь идет именно о тестировании на позитивных, но маловероятных данных.

2. **Криминальный тур**. Предполагает вводить недопустимые (нелегальные) значения – это и sql инъекции, и ввод текста в цифровые поля, и подмена данных.

3. **Тур с неправильными поворотами**. Предполагает выполнять действия в неправильной, нестандартной для пользователя последовательности, менять действия местами. Например, сначала указать адрес доставки, а потом выбирать товар; сначала в пустом плеере включить shuffle песен, а затем запускать сами песни и т.д.

Типичные баги: креши и другие функциональные ошибки; уязвимости в защищенности; непонятные для пользователя по формулировке сообщения об ошибках; проблемы в юзабилити.

* **Обсессивно-компульсивный тур или тур невротика.** Повторяйте одно и то же – вводите один и тот же текст раз за разом, выполняйте многократно какое-то действие или последовательность действий.

Например, осуществите поиск в интернет-магазине; откройте информацию по любому товару; положите товар в корзину; повторите шаги 1-3 очень много раз.

А можно просто осуществлять постоянно поиск. Это тоже относится к этому туру.

Типичные баги: некритичные функциональные ошибки; проблемы производительности; утечки памяти.

Туры привносят структуру в тестирование. Они помогают тест-инженерам обнаружить более актуальные и интересные сценарии, нежели при freestyle тестировании. Предоставляя тест-инженерам цели, туры подталкивают их на определенный путь, которые являются более сложными нежели классическое тестирование, основанное на декомпозиции функциональностей. Туры – инструмент для изучения фич в их комбинации. К тому же туры – это то, что является фиксируемым, их можно повторять и исключает многие проблемы пересечения тест-инженеров. Какие-то туры будут более удачные, какие-то менее. Опытным путем это можно определить. А далее их можно ранжировать различными способами (по найденным ошибкам, по времени выполнения, по покрытию кода/фич/интерфейса). Турами можно управлять, и это важно. На них можно учиться. Их можно распределять в команде. Можно определять их эффективность. Они помогают придать структуру тестированию и знаниям.

**Глава 3.2. Проведение Исследовательского тестирования.**

Предмет исследования - интернет сайт "Фотографии на памятники от производителя" (https://oval24.ru.)

Цель исследования – проверить в каком состоянии находится продукт с точки зрения функционала и пользовательских характеристик, чтобы на основе проведенного исследования будет составлено Техническое задания веб-продукта. Определить требования и ожидания к продукту непосредственно с его владельцем.

В качестве метода Исследовательского тестирование выбрано тестирование на основе сеансов. Это метод, основанный на времени, который направлен на количественную оценку процесса тестирования путем разделения его на «сессии» с учетом миссии.

Миссия – это цель и продолжительность конкретного сеанса, обеспечивающая исследователю четкую направленность.

Определяются рамки каждой сессии и детализирует любые конкретные цели, которые тестировщик намерен достичь. Это позволяет повысить уровень подотчетности (и документирования) за счет разделения этих проверок на более управляемые компоненты. Тесты на основе сеансов также повышают производительность и предоставляют тестировщику четкие метрики и информацию об устранении неполадок.

Тестирование будет проводится на основе стратерии, в качестве которой выступят Туры Джеймса Виттакера.

Приступая к проведению Исследовательского тестирования, для этого в первую очередь нужно определиться с тем, какие туры буду использовать при тестировании.

Я выделила список из 4х туров, которые подходят для тестирования моего веб-приложения:

1. Тур по путеводителю;

2. Интеллектуальный тур;

3. Тур уборщика;

4. Тур супермодели;

Для более эффективного планирования тестирования составлена матрица тестирования (Приложение 1.Матрица туров и типов ошибок.xlsx)

Матрица тестирования – это основа для совершенствования. На вертикальной оси отображены районы и туры, которые используются в качестве плана тестирования. На горизонтальной оси - типы возможных ошибок, найденных при тестировании. Т.о. матрица позволяет убедится, что в результате проведенных проверок будут охвачены все типы ошибок, необходимые на данном этапе проверки продукта.

На данный момент (перед составлением Технического Задания) запланировано провести тестирование на выявление следующих типов ошибок:

* функциональные;
* лингвистические;
* юзабилити;
* интерфейсные.

План проведения исследовательского тестирования:

1. Выбрать тур из списка.

2. Изучить его цели.

3. Поставить таймер на 2 часа.

4. Провести исследование системы строго по целям тура. Ни на что не отвлекаясь, только “миссия” тура. В качестве документирования сессий использовать чек-лист с фиксированием результатов проверок.

5. При необходимости повторить.

6. Проанализировать результаты и подвести этого

**Результаты исследовательского тестирования.**

В качестве тестовой документации при фиксировании результатов исследований выступили Чек-листы в TMS Qase.io (Приложение 2. Test runs):

https://app.qase.io/public/report/458e07293631237b072fd6cdd40a155503612dc5

https://app.qase.io/public/report/5a93b4a429c59e5bec91bf9f2cd7891cfcbe7072

https://app.qase.io/public/report/ffb3c53f83887bf097bfb0b641614ca11a1be1c6

https://app.qase.io/public/report/0ce194bfb60d26b39b55b28be8b9e23b29a7bf2c

https://app.qase.io/public/report/2cafacd4733882c255c6bccc0e1be0c640d3c48a

В процессе тестирования было проведено 5 сессий, 140 тестов, из них 27% провальные и 0.01% заблокированы для проверки (1 тест), было обнаружено 27 багов (Приложение3. Баги. Исследовательское тестирование).

По результату проверки обнаружено:

* отсутствует футер;
* у большинства полей ввода отсутствует проверка на валидацию вводимых данных;
* проиcходит обработка запроса в блоке "Задать вопрос" с пустыми полями (даже без контактов для обратной связи);
* поля ввода на страницах "Заказать онлайн" и "Вопросы" выходят за границы экрана при загрузке на мобильном;
* адрес электронной почти не в формате ссылки на нее;
* требуется уточнения по адресу офиса и условий оплаты

**Так же есть список замечаний после проведенной проверки:**

1. Кнопка «Выберите файл» в разделе "Заказать онлайн" при наведении на нее курсора изменяет цвет мало заметно, немного изменив оттенок серого, тоже самое после загрузки изображения.

2. Кнопки "Отравить" в разделах "Заказать онлайн" и "Вопросы" при наведении курсора или заполнении всех полей меняет цвет мало заметно, немного изменив оттенок серого.

3. Кнопка меню, которой соответствует страница, где находится пользователь недостаточно визуально выделена. Текст названия кнопки отличается от остальных полужирным стилем шрифта.

4. Номер телефона в хедере не отдельный элемент, для автоматического набора номера, а просто картинка.

5. «Краткая схема работы» на "Главной" странице - мелко, трудно читаемо. Надо увеличить размер на странице.

6. Информация на "Главной" странице разрознена, схема работы и инструкция «Как заказать» разорвана другими блоками («О нас» и др.). Требуется реструктуризация страницы, блоки расположить, учитывая логику оформления заказа, а остальную информацию вынести отдельно.

7. У части полей ввода фон серого цвета, выглядят задизейбленными.

Далее на основе полученных данных будет составлено Техническое задание к сайту. Баги, заведенные на данный момент, требуются только для определения требований к продукту по функционалу и удобству использования, а также согласования их с владельцем сайта. На данном этапе будут устраняться только лингвистические баги и те что заказчик почитает нужно срочно устранить, остальные будут аннулированы. После будут составлены тест-кейсы для повторной проверки на основе требований к веб-продукту.

**Глава 3.3. Что такое Техническое задание.**

ТЗ или техническое задание на разработку сайта – это специальный документ, в котором подробно описаны технические, функциональные и контентные составляющие будущего сайта. И чем подробнее будет документ, тем качественнее будет выполнен сайт: заказчик получит сайт, который он хотел, а подрядчик сделает ровно то, что от него требуется.

Сайты делаются не один день, иногда на это уходит несколько месяцев. За это время может случиться все что угодно, например:

1. У вас может смениться менеджер проекта. Новый человек будет не в курсе всех деталей проекта без ТЗ.
2. Вам может потребоваться перерыв в разработке сайта: будь то финансовые трудности или фокусирование внимания на другом направлении бизнеса.

Составление технического задания – это безопасность и прозрачность выполнения всех этапов разработки для обеих сторон. Также преимущество составления ТЗ – при учете всех этапов и мелочей вы избегаете потерянного времени и лишних денежных затрат, например, на правки или доработки. Вы получите тот результат, о котором договорились с исполнителем.

**Польза от технического задания очевидна:**

1. Заказчик:

● Понимает, за что он будет платить деньги и какой ему сделают сайт. Структура сайта видна сразу, и, если что-то не устраивает, изменения можно внести еще до начала разработки.

● Оценивает компетентность исполнителя. Грамотно составленное и понятное техзадание повышает доверие к разработчику.

● Защищает себя от недобросовестности исполнителя. Готовый сайт можно проверить на соответствие техническому заданию. Есть неточности? Разработчик их исправит. При наличии официального договора его можно принудить сделать это через суд.

● Упрощает замену исполнителей. Бывает, что заказчик и исполнитель ссорятся и не могут продолжать совместную работу. В такой ситуации с созданием сайта возникают проблемы. Однако при наличии подробного техзадания их можно легко решить: заказчик просто передает ТЗ новой команде, и она сразу же включается в работу.

● Узнает стоимость разработки продукта. Понять, когда будет готов сложный сайт и узнать окончательную стоимость разработки сразу нельзя. Сначала нужно разобраться с функционалом веб-ресурса. Именно для этого нужно составить техническое задание.

2. Исполнитель:

● Понимает желания заказчика. Для этого ему придется задать клиенту десятки вопросов, показать примеры, предложить решения. Потом записать все в соответствующий документ и согласовать с заказчиком. Он одобрил? Значит, исполнитель понял его правильно.

● Застраховывается от внезапных «хотелок» заказчика. Случается, что в ходе создания сайта заказчик вдруг решает поменять задачу. Если разработчик согласовал и подписал ТЗ, он может быть спокоен: даже суд встанет на его сторону.

● Показывает свою компетентность. Четко и понятно подготовленное ТЗ говорит о профессионализме разработчика.

● Зарабатывает деньги. Иногда составление технического задания оценивается как отдельная услуга.

● Облегчает и ускоряет работу. Благодаря качественному техническому заданию становится понятна структура сайта и функционал каждой страницы: можно переходить к написанию кода и разработке дизайна.

Техническое задание составляет разработчик. Грамотное ТЗ может составить только исполнитель. Проект-менеджер или разработчик понимают в создании сайтов больше владельцев кафе и стоматологических клиник. Тем не менее заказчик должен принимать в процессе самое непосредственное участие.

Заказчик:

● знакомит исполнителя с компанией, товарами или услугами, целевой аудиторией;

● объясняет цель создания сайта;

● рассказывает о своих желаниях и делится идеями;

● показывает примеры хороших (как ему кажется) сайтов.

● отвечает на вопросы исполнителя.

Заказчик может предложить свой вариант технического задания. В некоторых случаях это ускоряет процесс создания конечного ТЗ.

Если не составлять никакого ТЗ или сделать его некачественно, то есть не описать все необходимые требования, то образуется «серая зона». Например, у заказчика есть представление о том, как должна выглядеть страница бонусной программы, но он не описал ее исполнителю. Тогда исполнитель разрабатывает эту страничку исходя из своего предыдущего опыта и своих суждений. По итогу получится работающая страница бонусной программы с хорошим оформлением, однако она совсем не соответствует ожиданию заказчика. Обе стороны остаются недовольны работой друг друга.

**Глава 3.4. Составление Технического Задания.**

**Пишите однозначно и точно**! Главная цель техзадания – понимание между заказчиком и разработчиком. В документе не должно быть качественных прилагательных: красивый, удобный, современный. Такие слова можно оценить неоднозначно: каждый по-своему понимает красоту и современность.

То же самое относится и к невнятным формулировкам. Например:

● Сайт должен понравиться заказчику. А если не сможет?

● Сайт должен быть удобным. Для чего и для кого?

● Сайт должен выдерживать большие нагрузки. Сколько конкретно посетителей?

● Качественный экспертный контент. Ну, это понятно.

Обязательно проверьте текст: в нем не должно быть неоднозначных формулировок. В противном случае ТЗ придется переписать. Все мысли следует сформулировать четко и точно. Например:

● не «загрузка сайта должна быть быстрой», а «у каждой страницы должно быть более 80 баллов в Google PageSpeed Insights»;

● не «большая нагрузка», а «50 тысяч пользователей одновременно;

● не «на главной странице размещен список статей», а «на главной странице выведен список последних шести опубликованных статей»;

● не «разработка минималистичного удобного интерфейса подписки», а «поле «Оставьте e-mail» с кнопкой «Подписаться»».

**Разъясните сложные термины**. Техническое задание должны понимать все, для кого оно предназначено. Если вы планируете пользоваться терминами, которые непонятны вашей клиентке - владелице магазина сувениров - необходимо пояснить их.

**Опишите инструменты и требования к хостингу**. Допустим, вы в течение двух месяцев разрабатывали сайт. Каждый этап был согласован с заказчиком. И вот работа сделана. Во время показа админки заказчик возмущается: «Это «Модэкс»?! Я рассчитывал, что сайт будет на «Вордпрессе»!». Исключите такие ситуации. Для этого вам нужно четко описать инструменты, движки и библиотеки, а также указать требования к хостингу. Вдруг вы сделаете на PHP, а у заказчика сервер на .NET.

**Составьте список требований к работе сайта**. Готовый сайт должен работать в любом браузере и на всех устройствах. Это нужно обязательно прописать в ТЗ.

Также можно указать требования к следующим параметрам: скорость загрузки сайта; устойчивость к нагрузкам; защита от хакерских атак и т.д.

**Создайте структуру сайта.** До того, как вы начнете отрисовывать дизайн и верстать, согласуйте с заказчиком структуру сайта. Сначала нужно выяснить, что он хочет. Затем собрать сотрудников (разработчики, SEO-специалисты, маркетологи, главный редактор) и решить, какие именно страницы нужны на сайте и как их связать между собой.

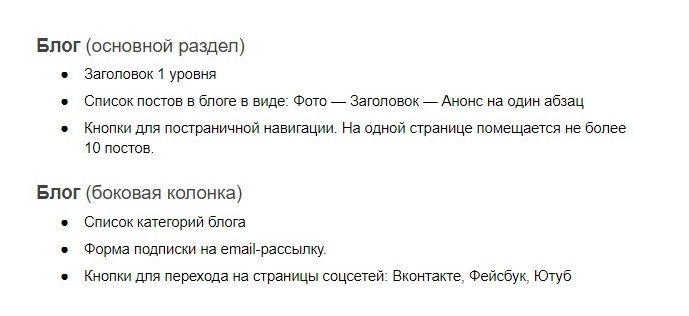
Структуру можно показать списком или нарисовать в виде блок-схемы.

Структура – фундамент сайта. Ее создание – самый важный этап работы. Если она получится неудачной, сайт будет «кривым».

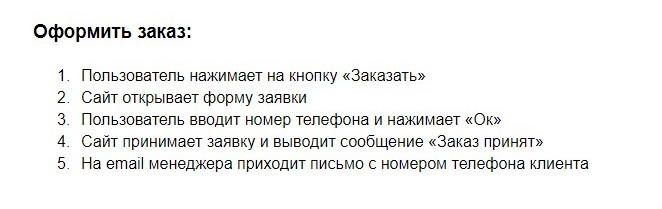
**Объясните содержание страниц.** Заказчику нужно понимать назначение каждой страницы и ее элементов. Для демонстрации есть два способа.

1. Прототип. Самый наглядный и однозначный способ. Исполнитель рисует эскизы каждой страницы и прикладывает их к ТЗ. Заказчик увидит, как будет выглядеть интерфейс сайта, и сможет сказать, что ему понравилось, а что лучше изменить.

2. Перечисление элементов – ленивая альтернатива прототипу. Если вы выбираете этот вариант, нужно лишь составить список блоков, которые предполагается разместить на странице.

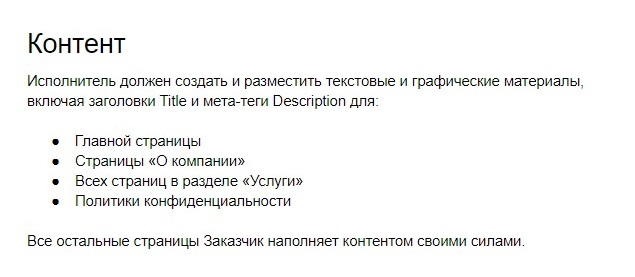


**Распишите варианты использования сайта.** Если интерфейс, который вы разрабатываете, будет нестандартным, простым показом структуры и эскизов страниц обойтись не получится. И ваши коллеги, и заказчик должны четко понимать, как именно посетители будут использовать сайт. Для наглядности нужно составить простую схему сценария: действие пользователя - ответное действие сайта - результат.

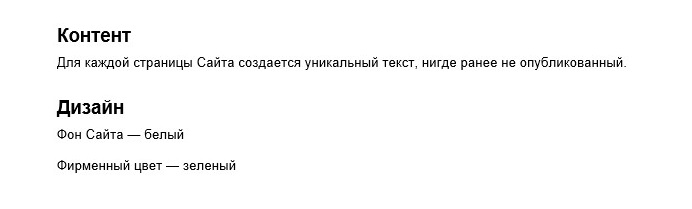


При создании стандартной визитки или лендинга вам не нужно писать сценарий. Но если вы работаете над размещением интерактивных сервисов на сайте, сделать это необходимо.

**Определитесь с контентом**. Некоторые исполнители разрабатывают сайты сразу с контентом. Другие делают рыбу. Кто-то может написать тексты, но не бесплатно. Обговорите с заказчиком, какой именно контент вы будете готовить и зафиксируйте это в техническом задании.



**Опишите дизайн.** Объективных критериев оценки дизайна сайта нет. Если заказчик хочет определенную цветовую гамму, пропишите это в ТЗ. Если он имеет брендбук с конкретными шрифтами, напишите и это. А вот слова «красивый» и «современный» употреблять не нужно.



**Вывод: структура ТЗ.** Одинаковых технических заданий не бывает: для каждой задачи пишется отдельное ТЗ. Грамотное техническое задание должно содержать:

1. Информацию о компании и целевой аудитории, целях и задачах сайта;

2. Глоссарий терминов, непонятных заказчику;

3. Требования к верстке и работе сайта;

4. Описание применяемых технологий и список требований к хостингу;

5. Подробную структуру сайта;

6. Прототипы страниц и описания содержащихся на сайте элементов;

7. Сценарии использования интерфейса, если он нестандартный;

8. Список контента;

9. Требования к дизайну (в общих чертах).

На основе полученных данных после проведения исследовательского тестирования и обсуждения выявленных багов и замечаний к работе веб-приложения с владельцем сайта (заказчиком), а также полученных знаний из данной главы было составлено Техническое задание к сайту [Фотомастерской “Кадр”](https://oval24.ru/)(<https://oval24.ru>) (Приложение 4. Техническое задание)

[**Глава 4. Проведение тестирования веб-приложения и составление рекомендаций по улучшению продукта.**](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B3%D0%BB%D0%B0%D0%B2%D0%B0-4-%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%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%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%)

**Глава** [**4.1. Проведение тестирования на основе составленной тестовой документации**](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#41-%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%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%BD%D0%B0-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B5-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9-%)**.**

И так уже составлено Техническое задание (Приложение 4. Техническое задание) к нашему веб-продукту, изучили в Главе 2.2. что такое текст-кейсы, как их составлять, и техники тест-дизайна. Теперь пора применить все на практике и на основе полученных знаний составить тест-кейсы для тестирования нашего сайта.

Всего было составлено 124 тест-кейса по 9 разделам сайта:

1. Хедер сайта.
2. Футер сайта.
3. Раздел "Главная".
4. Раздел "Варианты оформления".
5. Раздел "Цены".
6. Раздел "Оплата / Доставка".
7. Раздел "Заказать онлайн".
8. Раздел "Вопросы".
9. Раздел "Контакты".

В «Приложение 5. Test runs. Тестирование на основе тест-кейсов.» можно посмотреть подробно какие тест-кейсы были составлены и результаты их тестирования.

При составлении тест-кейсов использовались техники Эквивалентного разделения и Граничных значений, учитывались функциональные требования из Технического задания.

Так же было определено окружение для проведения тестирования - Windows 7, Google Chrome Версия 109.0.5414.120 (Официальная сборка), (64 бит)

[**4.2. Подведение итогов. Разработка предложений по улучшению работы программного продукта**](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#42--%D0%BF%D0%BE%D0%B4%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B8%D1%82%D0%BE%D0%B3%D0%BE%D0%B2-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0-%D0%BF%D1%80%D0%B5%D0%B4%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9-%D0%BF%D0%BE-%D1%83%D0%BB)**.**

Предмет исследования – сайт Фотомастерской «Кадр» (<https://oval24.ru>).

Цель тестирования – проверить в каком состоянии находится функциональность продукта, удобство использования интерфейса и логику и составить рекомендации для дальнейшего его развития.

Тестирование проводилось на основе тестовой документации:

1. Чек-листы (применялись при Исследовательском тестировании, Приложение 2. Test runs. Исследовательское тестирование).
2. Техническое задание (Приложение 4. Техническое задание).
3. Тест-кейсы (для проверки на соответствие требованиям к продукту, Приложение 5. Test runs. Тестирование на основе тест-кейсов)

В качестве инструментов документирования процесса тестирования выступил TMS [Qase.io](https://github.com/AnnaCherleniak/Diploma_project/blob/main/CherlenyakAL_3247.md#%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-kanban-qaseio-jira-devtools-git-obs-studio-2321-excel), а результатов (баг-репорты) – Jira.

В процессе первого этапа исследования, Исследовательского тестирования, было проведено 4 тура и 140 тестов. После составления Технического задание был проведен второй этап исследования, Тестирование на основе тест-кейсов. На данном этапе были проверены все разделы сайта и проведено 124 теста, из них 105 (84,7%) пройдены успешно, 16 (12,9%) – провалены и 3 (2,4%) – заблокированы для проверки.

После первого этапа проверки обнаруженные баги обсуждались с владельцем сайта на предмет их актуальности, важности и соответствуют ли ожидаемым требованиям к продукту. В результате этого обсуждения были составлены в Jira баг-репорты (Приложение 3. Jira\_Баг-репорт. Исследовательское Тестирование) и отправлены на исполнение.

В результате второго этапа исследования на основе тест-кейсов было обнаружено 16 багов и составлены баг-репорты (Приложение 6. Jira\_Баг-репорт. Тестирование на основе тест-кейсов.docx). Продукт имеет баги с высокой степенью критичности:

* 1. [**PSK-1**](https://ankakras.atlassian.net/browse/PSK-1) – Кнопки меню не соответствуют ТЗ, расположены НЕ в одну линию.
  2. [**PSK-3**](https://ankakras.atlassian.net/browse/PSK-3) – На сайте отсутствует Футер – на сайте отсутствует важный элемент интерфейса.
  3. [**PSK-6**](https://ankakras.atlassian.net/browse/PSK-6) – Условия оплаты в разделе "Оплата / Доставка" не соответствуют заявленным в ТЗ – информация перемешана с условиями доставки, есть лишние данные, а тех что должны быть – нет.
  4. [**PSK-**](https://ankakras.atlassian.net/browse/PSK-6)**14** – Отсутствуют сообщения об ошибке при отправке запроса со всеми пустыми полями в блоке "Задать вопрос" – отсутствует проверка на заполненность обязательных полей.
  5. [**PSK-**](https://ankakras.atlassian.net/browse/PSK-6)**15** – Отсутствуют сообщения об ошибке в разделе "Вопросы", после клика на кнопку «Отправить» , при заполнении поля "номер телефона" двумя пробелами – та же недоработка функционала, что и с [**PSK-**](https://ankakras.atlassian.net/browse/PSK-6)**14.**

Учитывая найденные баги на всех этапах исследования и список рекомендаций по улучшению интерфейса и удобства использования сайта (подробно в Главе 3.2.) требуется провести еще большой объем работы, однако следует отметить, что эти баги и рекомендации не оказывают глобального влияния на общий функционал сайта и его основную бизнес-цель.

Необходимость проведения дополнительной работы связана с тем, что улучшение интерфейса и удобства использования сайта является важным аспектом для привлечения и удержания пользователей.

Таким образом, несмотря на обнаруженные баги, их уровень серьезности, и рекомендации, которые требуют дополнительных улучшений, сайт остается полноценным веб-приложением, способным выполнять свою основную бизнес-цель. Работа над улучшением интерфейса и удобства использования сайта остается перспективной задачей на будущие этапы его развития.

**Заключение.**

В данной дипломной работе были исследованы вопросы организации процесса ручного тестирования веб-приложений на основе исследовательского подхода в условиях отсутствия технического задания. Целью работы было разработать методический подход к организации тестирования на примере интернет-сайта, учитывая особенности и требования исследовательского тестирования.

В ходе исследования были проанализированы особенности исследовательского тестирования и его применимость в рамках ручного тестирования веб-приложений. Были изучены технологические и методические основы организации процесса тестирования, а также рассмотрены различные подходы и методы к созданию тест-кейсов и баг-репортов.

Для практической проверки разработанного подхода было проведено исследовательское тестирование интернет-сайта с использованием Туров Джеймса Виттакера. В ходе тестирования были выявлены скрытые ошибки и уязвимости, что подтверждает эффективность исследовательского подхода в условиях отсутствия четкого технического задания.

На основе проведенного тестирования было составлено Техническое задание, учетом пожеланий и требований владельца сайта (заказчика), которое послужило основой для подробного тестирования интернет-сайта на основе созданных тест-кейсов. Это позволило углубить проверку функциональности и надежности веб-приложения, а также улучшить его общую производительность и удобство использования.

В результате данной работы был разработан и применен методический подход к организации процесса ручного тестирования веб-приложений в условиях отсутствия технического задания. Полученные результаты подтверждают его эффективность и практическую применимость. Разработанный подход позволяет выявить скрытые ошибки и уязвимости, повысить качество и надежность веб-приложений, а также улучшить пользовательский опыт.

Дальнейшая работа в данной области может быть направлена на изучение других методов и подходов к тестированию веб-приложений. Это поможет улучшить процесс разработки программного обеспечения и повысить его конкурентоспособность на рынке.

**Список использованной литературы и ресурсов.**

1. Роман Савин. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. (https://itexts.net/avtor-roman-savin/157563-testirovanie-dot-com-roman-savin/read/page-1.html)
2. Чек-лист тестирования WEB приложений - <https://habr.com/ru/articles/542422/>
3. Фундаментальная теория тестирования - <https://habr.com/ru/post/549054/>
4. Семь главных принципов тестирования - <https://testengineer.ru/sem-glavnyh-principov-testirovaniya/>
5. Структура, содержание и процесс написания проверок -<https://habr.com/ru/companies/surfstudio/articles/575360/>
6. Гид по ручному тестированию приложений: преимущества, этапы и методологии - <https://habr.com/ru/companies/skillbox/articles/418889/>
7. Тестирование программного продукта - <https://www.tadviser.ru/index.php/Статья:Тестирование_программного_продукта>
8. Исследовательское тестирование и исследовательские туры Виттакера - <https://www.software-testing.by/blog/exploratory-testing-exploratory-tours/>
9. Классификация видов тестирования - <https://test-planet.by/news/klassifikaciya-vidov-testirovaniya/>
10. Что такое smoke-тестирование? <https://testengineer.ru/chto-takoe-smok-testirovanie/>
11. ЧТО ТАКОЕ ВЕБ-ПРИЛОЖЕНИЕ: ВИДЫ И АЛГОРИТМ РАЗРАБОТКИ - <https://ru.ecomlife.pro/blog/chto-takoe-web-prilozhenie#submenu:about>
12. Виды веб-приложений: выбираем подходящий вариант для вашего бизнеса -<https://leantech.ai/vebprilozhenie-osnovnye-vidy-i-ih-osobennosti>
13. Исследовательское тестирование — глубокое погружение в типы, процессы, подходы, инструменты, фреймворки и многое другое! - <https://www.zaptest.com/ru/исследовательское-тестирование-глу>
14. Исследовательское тестирование - <https://qarocks.ru/issledovatelskoe-testirovanie/#stages>
15. Как составить грамотное ТЗ на разработку сайта - <https://habr.com/ru/articles/593661/>
16. Как составить чек-листы для эффективного тестирования продуктов: простые шаги и примеры - <https://habr.com/ru/articles/723948/>
17. Техники тест-дизайна: теория и примеры - https://testengineer.ru/tekhniki-test-dizajna-teoriya-i-primery/
18. ПРОЦЕСС ТЕСТИРОВАНИЯ. ЧАСТЬ 2: АНАЛИЗ ТЕСТИРОВАНИЯ И ТЕСТ ДИЗАЙН - <https://crashtest.by/test-analysis-and-test-design/>
19. Написание баг репорта - <http://www.protesting.ru/testing/bugwriting.html>
20. Что такое баг-репорт и как его составить - <https://gb.ru/blog/chto-takoe-bag-report/#:~:text=Баг-репорт%20–%20отчет%20об%20ошибках,коллег%2C%20которые%20ничего%20не%20исправят>.
21. Как составить грамотное ТЗ на разработку сайта - <https://habr.com/ru/articles/593661/>