

Тест-дизайн и тест-аналитика
Урок 1

Работа с требованиями





Оглавление

На этом уроке	4
Введение	4
Что такое требования?	4
Классификация требований	6
Функциональные требования	6
Нефункциональные требования	7
Атрибуты требований	8
Атомарность	9
Полнота	8
Однозначность	8
Непротиворечивость	8
Необходимость	10
Осуществимость	10
Тестируемость	10
Дефекты в требованиях	11
Форматы требований	12
Текстовое описание	12
Схематичное описание	13
User story — пользовательские истории	13
Use case — пользовательский сценарий	14
Системы управления требованиями	17
Задачи RMS	17
Confluence	17
Декомпозиция требований. Mind Map	18
Декомпозиция и создание Mind Map	19



Выделяем крупные блоки или компоненты	19
Определяем страницы сайта или экраны мобильного приложения для блока	20
Содержание экранов	20
Глоссарий	23
Контрольные вопросы	24
Практическое задание	24
Дополнительные материалы	24
Используемые источники	24



На этом уроке

1. Познакомимся с курсом.
2. Узнаем, что такое требования и их особенности.
3. Познакомимся с декомпозицией требований.
4. Научимся создавать Mind Map.

Введение

На этом курсе мы научимся применять разные техники и приёмы для проектирования тестов — тест-кейсов и чек-листов. Как результат, составление тестовой документации и сам процесс тестирования будут интуитивными, а тестовое покрытие функциональности — оптимальным (без лишних проверок и без пропуска важных).

Также научимся работать с требованиями — тестировать и анализировать их, составлять тестовую документацию по требованиям и тестировать без требований. Узнаем, как составлять отчёты для разных видов тестирования и разного уровня отчётности на проекте.

Что такое требования?

Требование — описание функций и условий, которые выполняет приложение в процессе решения пользовательской задачи (например, оформления заказа в интернет-магазине). Это отправная точка процесса разработки.

Раздел «Поиск»

В главном меню есть кнопка с иконкой «Поиск» —



По клику на кнопку открывается модальное окно.

FoodBuzz
GOOD FOOD FOR GOOD HELTS

[Главная](#)

[Меню](#)

[Забронировать](#)

[Блог](#)

[Take away](#)

[Контакты](#)



В модальном окне отображается строка поиска с подсказкой в поле ввода текста (плейсхолдер от англ. placeholder), говорящей пользователю, что поисковый запрос нужно ввести в это поле. Значение плейсхолдера: «Введите ваш запрос...».

При вводе поискового запроса исчезает плейшолдер. При клике на крестик закрывается модальное окно.

Поисковая строка на вход принимает кириллицу и латиницу, цифры. В противном случае, отображается ошибка (поле подсвечивается красным и под полем отображается текст: «Неверный формат данных»).

Максимальная допустимая длина — 256 символов.

При введённом поисковом запросе и клике на лупу выполняется поисковой запрос. Пользователь перенаправляется на страницу «Поиск», на которой отображаются результаты запроса.

Также на модальном окне есть крестик в правом верхнем углу, по клику на который модальное окно закрывается и пользователь возвращается на ту же страницу, с которой перешел в поиск.

При вводе поискового запроса пользователю отображаются сдджесты (подсказки). Например, если пользователь вводит «д», отображаются подсказки: «десерт», «детское меню», «диетическое меню». Если пользователь вводит «с», отображаются подсказки: «супы», «салаты», «сулугуни» и так далее.

Если отсутствует подключение к интернету, отображается информация с ошибкой «Отсутствует соединения к интернету» и кнопка «Обновите страницу».



Произошла ошибка.

 Обновите страницу

При клике на кнопку «Обновите страницу» отображается строка «Поиск».



Требования создаёт заказчик, аналитик или технические специалисты на этапе планирования.

Задача требования — зафиксировать ожидания заказчика о том, какой продукт планируется разработать.

Как только появляются требования, может начинаться процесс тестирования. Тестировщик участвует в проверке и анализе требований, находит неясности и противоречия, предоставляет отзыв о функциях и об удобстве использования будущего приложения.

Классификация требований

По характеру требования делятся на функциональные и нефункциональные.

Функциональные требования:

- бизнес-требования;
- пользовательские требования;
- системные требования.

Нефункциональные требования:

- бизнес-правила;
- внешние интерфейсы;
- атрибуты качества.

Функциональные требования

Отвечают на вопрос: «Что должна делать программа?». Например, показывать прогноз погоды или воспроизводить видео.

Бизнес-требования — это обобщённое описание функций продукта без технической детализации (например, без требований к аппаратному обеспечению). Они отвечают на вопрос: «Какую потребность пользователя закрывает ПО?».

Пример: для привлечения новых клиентов надо разработать сайт с описанием преимуществ компании «ОПТИМА КОНСАЛТИНГ ПЛЮС».

Пользовательские требования — описание задач, которые пользователь сможет решить, используя приложение. Эти требования более детализированные, чем бизнес-требования, но не содержат технических подробностей.

Пример:

1. Пользователь заходит на сайт и видит список услуг, которые предоставляет компания «ОПТИМА КОНСАЛТИНГ ПЛЮС».
2. Пользователь может скачать прейскурант услуг.



3. Пользователь может оставить заявку на обратный звонок:

- a. Заполнить ФИО.
- b. Заполнить номер телефона.
- c. Согласиться с обработкой персональных данных.

Системные требования — описание технической реализации программы, а также требуемое программное и аппаратное окружение.

Пример:

1. Сайт разрабатывается на платформе WordPress.
2. Для хранения пользовательских данных используется БД MySQL.
3. Сайт должен открываться в браузерах Chrome, Firefox, Safari.

Нефункциональные требования

Отвечают на вопрос: «Как должна работать программа?». Связаны с нефункциональными видами тестирования, то есть относятся к быстродействию, внешнему виду, удобству использования, локализации и другим подобным критериям.

Примеры нефункциональных требований:

- время обработки пользовательского запроса не выше 100 миллисекунд;
- размер загружаемых файлов не выше 10 Мб;
- размер шрифта на странице — 14 pt.

К нефункциональным требованиям относятся бизнес-правила и внешние интерфейсы.

Бизнес-правила определяют, почему система должна работать именно так. Это ссылки на законодательство, внутренние правила заказчика и другие причины.

Пример: табачные компании требуют постоянного доказательства, что промосайтами пользуются люди, достигшие 18 лет. Это бизнес-правило возникает по требованию этических комитетов заказчика, хотя и несколько противоречит маркетинговым целям.

Внешние интерфейсы — интерфейсы пользователя (макеты) и протоколы взаимодействия с другими системами.

Пример: сайт должен взаимодействовать с CRM по HTTP.



Атрибуты качества:

- лёгкость и простота использования;
- производительность;
- удобство эксплуатации и технического обслуживания;
- надёжность и устойчивость к сбоям;
- взаимодействия системы с внешним миром;
- расширяемость;
- требования к пользовательским и программным интерфейсам.

Атрибуты требований

Чтобы требования считались корректными и давали разработчику и тестировщику однозначные указания по работе приложения, они должны быть:

- атомарными
- полными;
- однозначными;
- непротиворечивыми;
- необходимыми;
- осуществимыми;
- тестируемыми.

Атомарность

Требование нельзя разбить на отдельные части без потери деталей.

Неправильно:

Если пользователь оформляет заказ и редактирует в нём товары или меняет дату доставки, то выполняется запрос на создание или редактирование заказа.

Запрос выполняется при комбинации всех условий? Какой именно запрос в каком случае должен выполняться? Из-за объединения нескольких частей в одно требование его трудно понять.

Правильно:

Если пользователь оформляет заказ, выполняется запрос на создание заказа.

Если пользователь редактирует товары в заказе, выполняется запрос на редактирование заказа.

Если пользователь меняет дату доставки заказа, выполняется запрос на редактирование заказа.



Полнота

Требования должны учитывать все возможные пользовательские действия, входные параметры, сообщения об ошибках. Если требование неполное, в этом месте возникают дефекты.

Лучший способ проверить требования на полноту — начать писать тесты. При планировании проверок «белые пятна» станут очевидны.

Неправильно:

1. При регистрации пользователь указывает дату рождения.
2. Если дата рождения указана, то возраст указывается в анкете.

А если дата не указана? Поле «Возраст» пустое? Или не отображается вообще? Обязательное ли поле даты рождения?

Правильно:

1. При регистрации пользователь указывает дату рождения, поле необязательное.
2. Если дата рождения указана, то возраст указывается в анкете.
3. Если пользователь не указал дату рождения, то поле возраста скрыто из анкеты

Однозначность

Важно, чтобы требования не допускали двусмысленных формулировок. Все требования, касающиеся субъективных параметров, например, скорости работы, эстетики интерфейса и удобства использования, описываются в абсолютных числах.

Неправильно:

Требуется, чтобы страница загружалась быстро.

Что значит «быстро»? Пять секунд — это быстро или медленно? А если у пользователя нестабильное интернет-соединение?

Правильно:

Страница должна загружаться не более 5 секунд при стабильном интернет-соединении.



Непротиворечивость

Когда требований много, они противоречат сами себе. Например, поведение одного и того же компонента описывается различными аналитиками в разных разделах требований. И это поведение будет различаться.

Пример

1. Требуется, чтобы отчёт о продажах формировался за 5 секунд.
2. Формирование отчёта о продажах занимает около 15 минут.

Какое требование считать верным?

Необходимость

Принцип составления документации: «Кратко, но ёмко». Важно, чтобы в ней было всё необходимое, но без лишней детализации.

В техническом задании описывается инструментарий, основной сценарий и альтернативы, типы ошибок.

В пользовательской документации описывается, как пользоваться системой, не доходя до крайностей и обучения включению компьютера.

Осуществимость

Важно, чтобы требования осуществлялись с учётом внутреннего устройства программы и технологий, которые используются разработчиками.

Пример: требуется, чтобы отчёт о продажах за квартал агрегировал данные из пяти таблиц и отображался на экране за одну секунду.

Возможно, это требование невыполнимо, так как на выполнение запроса, агрегацию данных и отрисовку на экране физически потребуется больше времени.

Тестируемость

Важно, чтобы у тестировщика была возможность проверить функциональность, которую создал разработчик. Особенно это касается автотестов: может случиться, что действующие библиотеки автоматизации не покроют новые функции, и их придётся дорабатывать.



Пример: для регистрации пользователя требуется указать уникальный email.

1. Есть ли у тестировщика доступ к базе данных, в которой хранятся уже зарегистрированные адреса?
2. Могут ли автотесты создавать уникальные адреса или проверять их уникальность? Если нет, то требование становится не тестируемым.

Дефекты в требованиях

Если требования не соответствуют атрибутам, значит, в них есть дефект. Дефекты на требования заводятся при тестировании требований и сохраняются в багтрекинг-системах, как и дефекты на функциональность.

Рассмотрим требование: важно, чтобы группам выдавались проектные роли в настройках проекта или в панели администрирования.

Требование неоднозначное — непонятно, где именно должны выдаваться проектные роли: в настройках проекта, в панели администрирования или и там, и там?

Дефект на требование может выглядеть так:

1. **Название.** Требование №1 — неоднозначное.
2. **Описание.** Непонятно, как именно выдаются проектные роли: в настройках проекта, в панели администрирования или и там, и там?

В багах на требования не нужны шаги воспроизведения, ожидаемый и фактический результат. Достаточно описать, какой атрибут не соблюдается.

Баги на требования сложно найти, просто прочитав документацию. Обычно они обнаруживаются в процессе написания чек-листов или тест-кейсов. Поэтому лучше начинать писать списки проверок сразу после получения документации: это ускорит процесс создания тестовой документации и позволит найти требования как можно быстрее.

Когда обнаружен баг в требовании (несоответствие атрибуту) и составлен баг-репорт на это несоответствие, автор требований их правит, после чего тестировщик перепроверяет требования и закрывает заведённый баг, если всё исправлено. Или снова возвращает требование на доработку, если дефект остался.

Один из популярных форматов оформления требований — пользовательские истории и пользовательские сценарии.



Форматы требований

Требования на проекте могут быть в разном виде — текст, схематическое описание, user story или use case.

Вид требований на проекте определяет команда: с какими удобнее работать, те и выбирают. Видов может быть несколько, чтобы они дополняли друг друга.

Давайте рассмотрим, как выглядят требования каждого из форматов:

Текстовое описание

В тексте требований должна быть описана логика, внешний вид, допустимые и недопустимые значения и так далее.


Формат удобный, но довольно сложный в составлении — легко упустить важные детали, в требованиях часто встречаются баги.

Если функциональность меняется, могут быть сложности с актуализацией текста.


Пример текстового требования на раздел «Корзина»:

Раздел «Корзина»

Раздел «Корзина» находится в блоке «Меню». Раздел отображается в виде иконки корзины



На иконке отображается каунтер добавленных позиций (например, если пользователь добавил 2 позиции куриных крылышек и 1 позицию лимонада, то на каунтере отображается значение 3). Минимальное значение каунтера — 1, а максимальное отображается как 99+ (если в корзине будет 99 позиций, на счётчике отображается значение 99, если 100 и более — 99+).



Пример требования на весь проект — [требования на foodbuzz](#).



Схематичное описание

Описание функциональности представлено в виде схематичного изображения требований из макетов с пояснениями и визуализацией переходов. Могут использоваться таблицы, блок-схемы и так далее.

Если нужно максимально полно описать функциональность, схематическое описание может дополняться текстовым.

Пример: [карта функциональности «Поиск»](#)

User story — пользовательские истории

Пользовательские истории — способ описания требований к системе в одном или нескольких предложениях. Для заказчиков (пользователей) пользовательские истории — основной инструмент влияния на разработку программного обеспечения.

Пользовательские истории — быстрый способ документировать требования клиента без необходимости разрабатывать обширные формализованные документы и тратить ресурсы на их поддержание. Цель пользовательских историй — оперативно и без накладных затрат реагировать на быстро изменяющиеся требования реального мира.

Пользовательская история описывает:

- человека, использующего систему (заказчик);
- то, что должно содержаться в этой системе (примечание);
- то, для чего она требуется пользователю (цель).

Пример:

1. Я как администратор хочу, чтобы в настройках проекта пользователям выдавались проектные роли.
2. Я как тестировщик хочу присваивать автотестам лейблы.

Пользовательские истории — результат планирования. Они:

- определяют, что должно реализоваться в программе;
- приоритезируются клиентом по важности для системы;
- разбиваются на серию задач и оцениваются разработчиками.



Use case — пользовательский сценарий

Пользовательский сценарий описывает взаимодействия участников, как правило, пользователя и системы. Количество участников — от двух и больше. Пользователь — человек или другая система. Оформляются в виде таблиц или диаграмм.

Сценарий — таблица:

Действующие лица	Пользователь, система
Цель	Изменить статус учётной записи пользователя на «Активный»
Предусловие	Учётная запись пользователя неактивна
Успешный сценарий	
<ol style="list-style-type: none">Администратор выбирает учётную запись пользователя и нажимает кнопку «Активировать»Система переключает учётную запись в статус «Активный»Система отправляет сообщение пользователю на email.	
Результат	Учётная запись пользователя перешла в статус «Активный»

Сценарий — диаграмма:

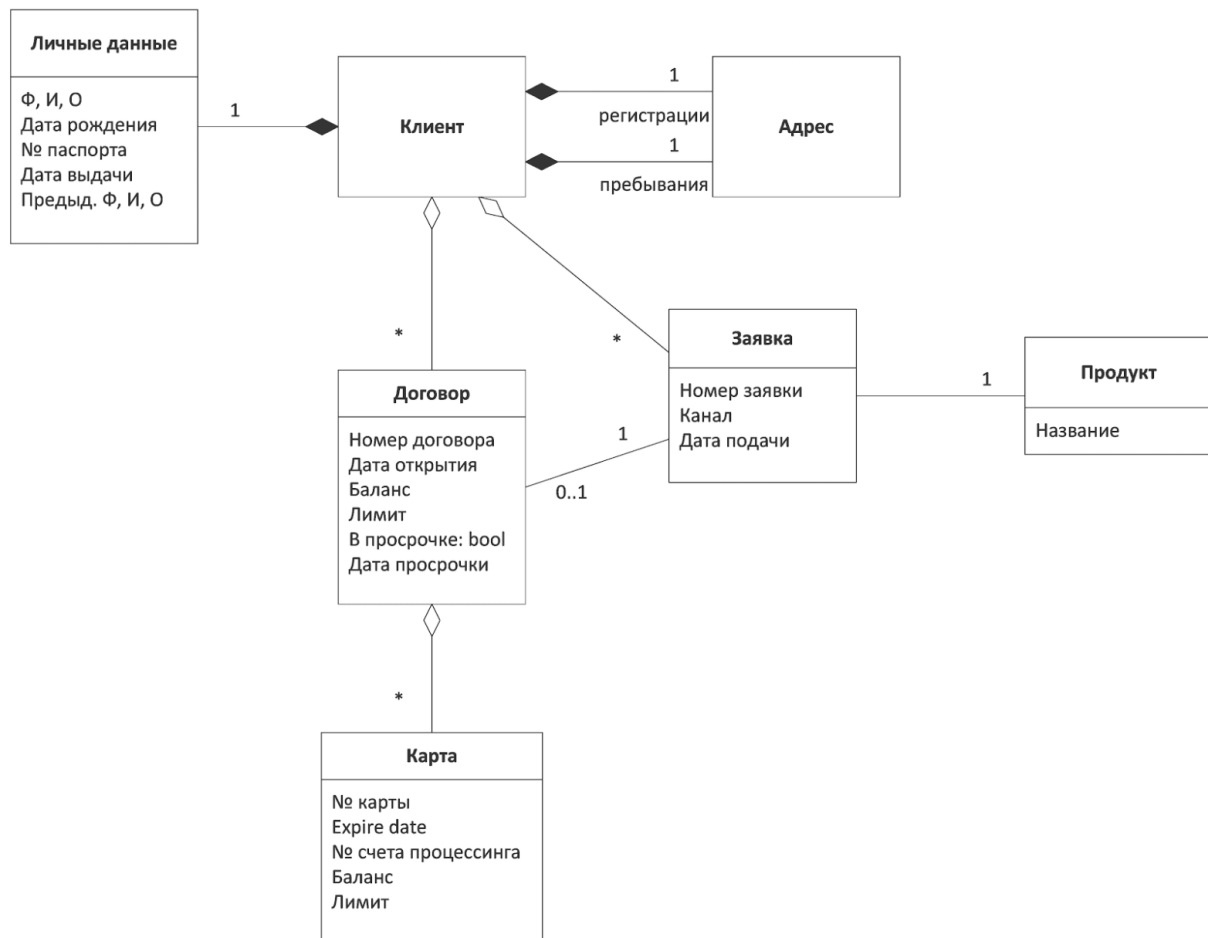


Диаграмма написана с использованием нотации UML. Подробнее — [UML-диаграммы классов](#)

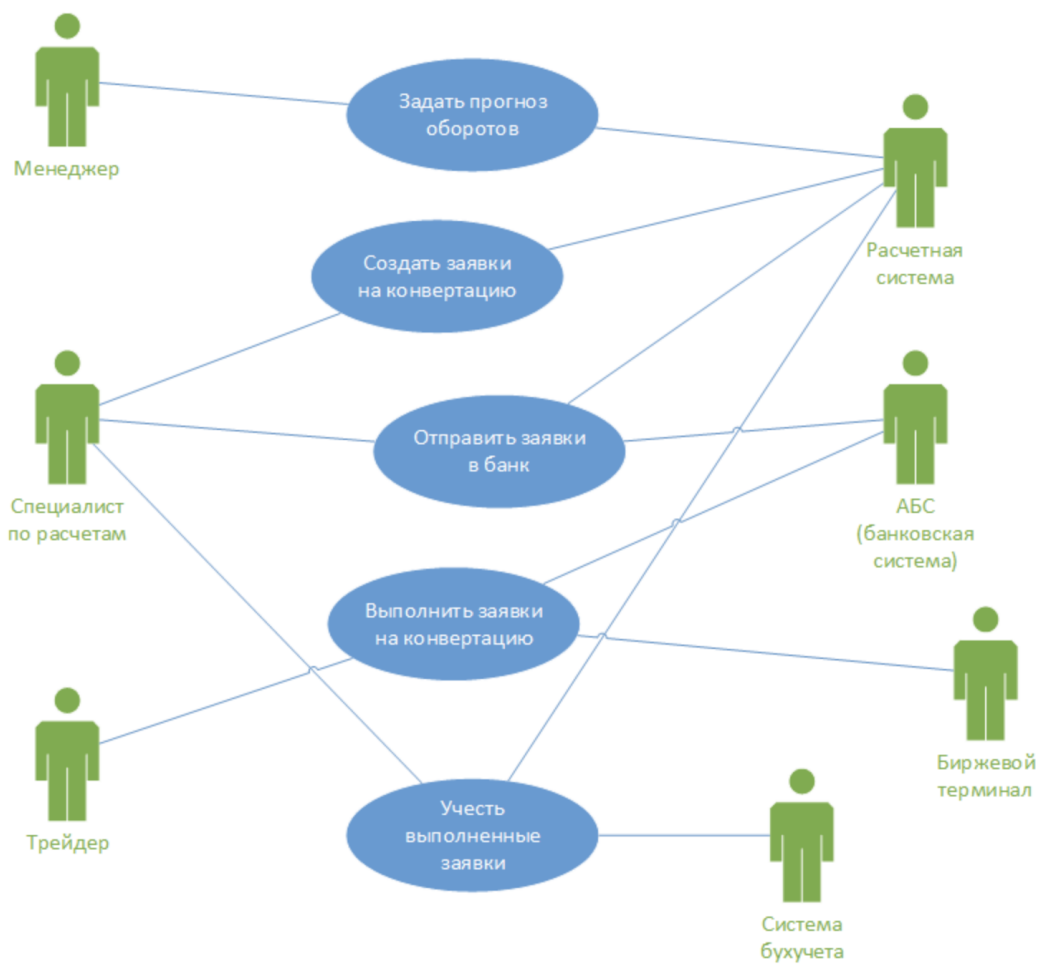


Диаграмма пользовательских сценариев. На ней обобщённо показано, какие участники вступают во взаимодействия

Пользовательские сценарии решают несколько задач:

1. **Оценка трудоёмкость проекта** тем точнее, чем детальнее список предстоящих работ. Сценарии использования — первый этап разбивки системы на отдельные элементы. Каждый сценарий декомпозируется на основной или альтернативные потоки либо задачи, которые требуется выполнить программисту для реализации отдельного сценария.
2. **Планировка графика работ.** Когда определена трудоёмкость каждого сценария использования, менеджер устанавливает сроки работы над каждым из них. Таким образом, определяются вехи начала и окончания разработки и тестирования каждого сценария использования, процесс разработки становится прогнозируемым.
3. **Выявление пропущенных требований.** Когда заказчик озвучивает требования к будущей системе, он концентрируется на полезной функциональности и не упоминает «второстепенные вещи»: настройки системы или особенности управления учётными записями пользователей,



хотя без них система не будет полноценно работать. Однако «второстепенные» функции могут потребовать больше времени, что серьезно повлияет на планирование.

Системы управления требованиями

Система управления требованиями (RMS, requirements management systems) — средство поддержки и автоматизации процесса работы с требованиями на протяжении всего жизненного цикла разработки программного продукта.

Есть разные системы управления требованиями. Мы не будем рассматривать их все, с некоторыми можно ознакомиться в статье «[10 Best Requirements Management Tools & Software Of 2022](#)».

Задачи RMS

1. Хранение требований в одном месте.
2. Единое управление требованиями.
3. Повышение производительности труда благодаря контролю над изменениями в требованиях и управлению ими.
4. Минимизация расходов и рисков благодаря оценке влияния происходящих изменений.
5. Демонстрация соответствия требований благодаря полному отслеживанию требований.
6. Сокращение объёма доработок и ускорение выхода на рынок благодаря совместной работе с заинтересованными лицами.

Давайте рассмотрим одну из самых популярных RMS систем — Confluence.

Confluence

Confluence — внутренняя вики-система для организаций. Поможет создать единую базу знаний. Написана на Java, разрабатывается компанией Atlassian. Распространяется под проприетарной лицензией, бесплатна для некоммерческих организаций и открытых проектов.

Позволяет решать задачи:

1. Создание и хранение проектной и технической документации.
2. Создание и управление требованиями в более узком виде, чем RMS.
3. Экспорт и импорт документации (документов).



4. Создание связи (ссылок и меток) между документами.
5. Возможность комментирования и обсуждения требований и документации.
6. Возможность отслеживания версионности и внесения изменений, а также сравнения разных версий документа.
7. Автоматические уведомления о внесении изменений в документах и страницах, на которые подписаны.
8. Управление доступом к проектам.

[Создать аккаунт и начать работать в Confluence](#)

Декомпозиция требований. Mind Map

Когда нам в жизни нужно решить какую-то большую задачу (например, переехать), мы разбиваем её на небольшие части: найти новое жилье, собрать вещи, сделать уборку, перевезти вещи, разложить всё по местам.

Когда разработчик получает задачу на реализацию новой функциональности, он также делит её на небольшие задачи: например, сверстать экран, запрограммировать логику, поработать с API.

Так и при работе с требованиями — мы можем поделить большое описание функциональности на мелкие части. Степень детализации будет зависеть от нужд команды и предназначения такой декомпозиции. В результате тестировщик часто составляет Mind Map (интеллект-карту) функциональности или всей системы.

Mind Map для переезда:





Mind Map (диаграмма связей, интеллект-карта) — это способ представления и хранения информации в виде некоторой схемы.

Преимущества:

- структурированная информация;
- снижение риска пропустить дефекты за счёт наглядности;
- возможность проведения тестирования по интеллект-карте без тестовой документации.

Рассмотрим пример декомпозиции и составим майнд-карту на основе [требований к сервису Foodbuzz](#).

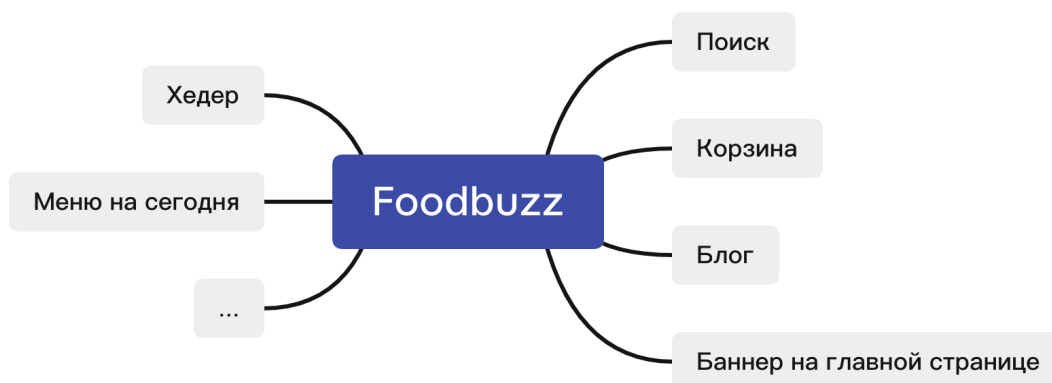
Декомпозиция и создание Mind Map

Выделяем крупные блоки или компоненты

На этом этапе выделяем крупные модули или подсистемы программы, которые могут существовать практически независимо друг от друга. На примере Foodbuzz:

- поиск;
- корзина;
- блог;
- баннер на главной странице;
- меню на сегодня
- и так далее.

Начнём создавать майнд-карту. В центр поместим главный объект — Foodbuzz. От него пойдут ответвления — объекты первого уровня.



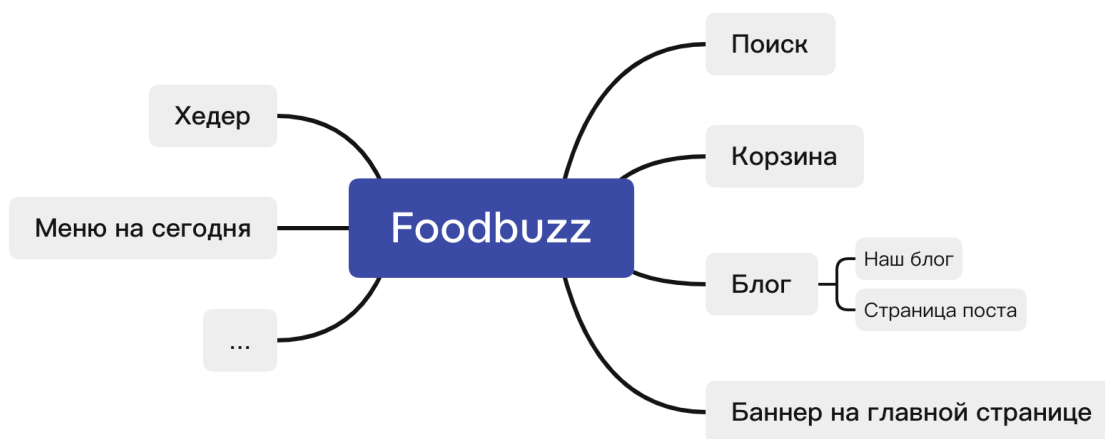


Определяем страницы сайта или экраны мобильного приложения для блока

Рассмотрим раздел «Блог». Согласно [документации](#), в этом разделе две страницы:

1. Наш блог
2. Страница опубликованного поста

Добавим их на майнд-карту.



Иногда может быть так, что выделенный раздел в первом шаге слишком маленький, но мы не можем объединить его с другим — в таком случае перейдём сразу к третьему шагу и рассмотрим продолжение декомпозиции раздела «Блог» и декомпозируем раздел «Поиск».

Содержание экранов

На экранах иногда появляются отдельные блоки с информацией, поля ввода, кнопки, списки — всё, с чем взаимодействует или просто просматривает пользователь.

На этом этапе декомпозиции важно отметить:

- элементы, расположенные на экране;
- действия, которые может совершить пользователь;
- параметры действий пользователя.

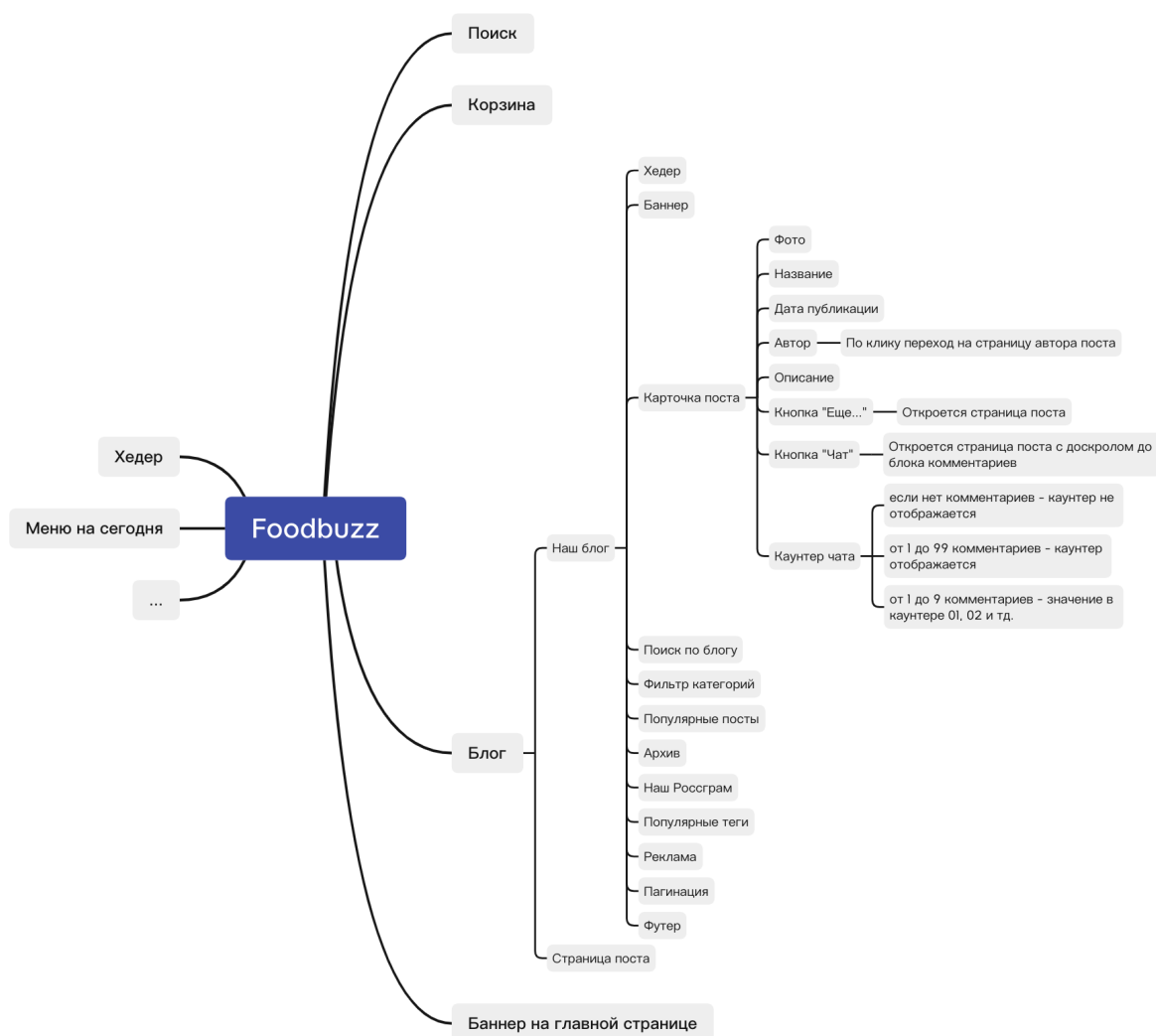
Добавляется также ожидаемое поведение системы. Главное, чтобы это не перегружало карту. Делать это надо только в крайнем случае.



Рассмотрим экран «Наш блог». На нём есть:

- хедер,
- баннер,
- карточки постов,
- поиск по блогу,
- фильтр категорий,
- популярные посты,
- архив,
- наш Россграм,
- популярные теги,
- реклама,
- пагинация(нумерация страниц),
- футер.

Добавим всё это на нашу майнд-карту.





Проведем аналогичную последовательность действий для раздела «Поиск». Важно учесть, что у раздела есть два разных состояния — работа с интернет-соединением и работа без интернет-соединения. В итоге получаем такой вариант декомпозиции:



Таким образом, майнд-карта заполняется до тех пор, пока не будут охвачены все объекты. Третий уровень детализации — самый подробный, на нём важно ничего не пропустить, так как майнд-карта станет основой для дальнейшего составления чек-листов и тест-кейсов.

Мы рассмотрели один из возможных вариантов декомпозиции требований и составления майнд-карты. Можно выбрать другой подход и структуру майнд-карты: при составлении ориентироваться на цель составления и тех, кто с ней будет работать. Карта должна быть понятной и информативной для всех заинтересованных лиц.

Важно! У элементов интерфейса интересные названия. Например, кнопки могут называться:

- кнопка;
- radio button;
- burger menu;
- тоггл.

Типы списков:

- чек-лист;
- аккордеон;
- select;



- dropdown.

Кроме бургеров и аккордеонов, на странице иногда появляются хлебные крошки, карусели, маски и другое. Подробнее:

- [32 элемента User interface для UI-дизайнеров](#);
- [Элементы интерфейса сайта](#).

Глоссарий

Бизнес-правила — принципы, которые определяют, почему система должна работать именно так: ссылки на законодательство, внутренние правила заказчика и прочие причины.

Бизнес-требования — обобщённое описание функций продукта без технической детализации. Отвечают на вопрос: «Какую потребность пользователя закроет ПО?».

Внешние интерфейсы — интерфейсы пользователя (макеты) и протоколы взаимодействия с другими системами.

Требование — описание функций и условий, выполняемых приложением в процессе решения задачи. Это отправная точка для процесса разработки.

Пользовательские истории — способ описания требований к системе одним или несколькими предложениями. Пользовательская история описывает:

- человека, использующего систему (заказчик);
- то, что содержится в этой системе (примечание);
- то, для чего она требуется пользователю (цель).

Пользовательские требования — описание задач, которые сможет решить пользователь, используя приложение. Эти требования более детализированные, чем бизнес-требования, но не содержат технических подробностей.

Пользовательский сценарий — описание взаимодействия участников, как правило, пользователя и системы. Количество участников — от 2 и больше. Пользователь — человек или другая система.

Система управления требованиями (RMS, requirements management systems) — средства поддержки и автоматизации процесса работы с требованиями на протяжении всего жизненного цикла разработки программного продукта.



Системные требования — описание технической реализации программы, а также требуемое программное и аппаратное окружение.

Контрольные вопросы

1. Какие виды требований существуют?
2. Какие атрибуты требований существуют? Дайте краткое описание каждому атрибуту.
3. Какие задачи решают user story и use case?
4. Для чего требуются RMS?

Дополнительные материалы

1. [10 Best Requirements Management Tools & Software Of 2022](#) — виды RMS.
2. [Confluence](#) — сайт Atlassian.
3. [32 элемента User interface для UI-дизайнеров](#) — названия элементов интерфейса.
4. [Элементы интерфейса сайта](#) — ещё о названиях элементов.
5. [Mind Map в помощь тестировщику](#).
6. [Как нарисовать карту приложения \(Mind Map\)](#).
7. [Mind Map вместо тест-кейса, или как визуализация позволяет тестировать приложение быстрее](#).
8. [Xmind](#) — ссылка для скачивания.

Используемые источники

1. [Требования к ПО на пальцах](#).
2. [Как правильно писать User Stories: руководство для разработчиков](#).
3. [Работаем с User Stories](#).
4. [Декомпозиция](#).

