GeekBrains

Основы тест-аналитики

Анализ требований. Декомпозиция

На этом уроке

1. Узнаем о требованиях и их атрибутах.
2. Научимся находить баги в требованиях.
3. Декомпозируем требования и создадим майнд-карту.

Оглавление

Что такое требования?

- Классификация требований
  - Функциональные требования
  - Нефункциональные требования
- Атрибуты требований
  - Полнота
  - Однозначность
  - Непротиворечивость
  - Необходимость
  - Осуществимость
  - Тестируемость
- Дефекты в требованиях
- User story — пользовательские истории
  - Преимущества пользовательских историй
  - Недостатки пользовательских историй
- Use case — пользовательский сценарий
  - Сценарий — таблица
  - Сценарий — диаграмма
- Системы управления требованиями
  - Задачи RMS
  - Критерии выбора системы, чтобы управлять требованиями
  - Confluence

Декомпозиция требований. Mind map

- Уровни декомпозиции
  - 1 уровень. Крупные блоки или компоненты

2 уровень. Страницы сайта или экраны мобильного приложения

3 уровень. Содержание экранов

Глоссарий

Контрольные вопросы

Практическое задание

Дополнительные материалы

Используемые источники

Что такое требования?

Требование — описание функций и условий, которые выполняет приложение в процессе решения задачи. Это отправная точка процесса разработки.

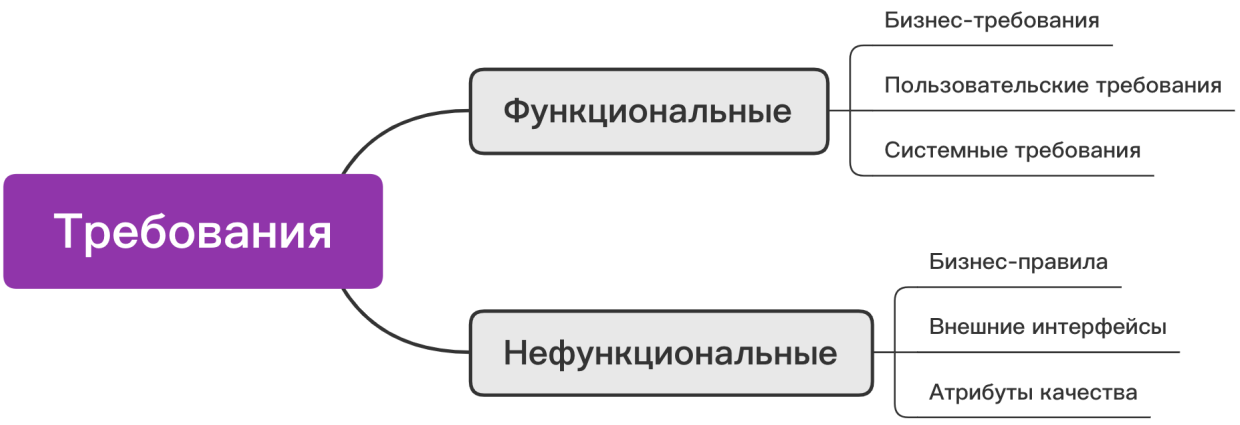
Требования создаёт заказчик, аналитик или технические специалисты на этапе планирования.

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

Тестировщик участвует в проверке и анализе требований, находит неясности и противоречия, предоставляет отзыв о функциях и удобстве использования будущего приложения.

Классификация требований

По характеру требования делятся на функциональные и нефункциональные.



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. Они отвечают на вопрос: «Какую потребность пользователя закроет 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, предоставляемых компанией «ОПТИМА КОНСАЛТИНГ ПЛЮС».
2. Пользователь может скачать прейскурант услуг компании.
3. Пользователь может оставить заявку на обратный звонок:
    a. Заполнить Ф. И. О.
    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 создаётся 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?». Нефункциональные требования связаны с нефункциональными видами тестирования, т. е. относятся к быстродействию, внешнему виду, удобству использования, локализации и т. д. Примеры нефункциональных требований:

- время обработки пользовательского запроса не превышает 100 миллисекунд;
- размер загружаемых файлов не превышает 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 прочие причины.

Пример

Табачные компании требуют постоянного доказательства, что промосайтами пользуются люди, достигшие определённого возраста. 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. К таким характеристикам относятся:

- 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:

- полными;
- однозначными;
- непротиворечивыми;
- необходимыми;

- осуществимыми;
- тестируемыми.

Полнота

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

Лучший способ проверить требования на полноту — начать писать тесты. В процессе планирования проверок «белые пятна» станут очевидны.

Пример

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?

Однозначность

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

Пример

Требуется, чтобы страница загружалась быстро.

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

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 содержалось 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, т. к. 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 так, 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 сохраняются в багтрекинговых системах, как и дефекты на функциональность.

Рассмотрим такое требование:

1. Важно, чтобы группам выдавались проектные роли в настройках проекта или в панели администрирования.

Оно не считается однозначным, т. к. непонятно где именно должны выдаваться проектные роли: в настройках проекта, в панели администрирования или в обоих местах?

Дефект на требование может выглядеть следующим образом:

1. **Название.** Требование №1 — неоднозначное.
2. **Описание.** В требовании непонятно, как именно выдаются проектные роли: в настройках проекта, в панели администрирования, или в обоих местах?

В багах на требования не нужны шаги воспроизведения, ожидаемый и фактический результат. Достаточно описания, 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 и ускорит процесс создания тестовой документации, и позволит найти требования как можно быстрее.

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

User story — пользовательские истории

Пользовательские истории — способ описания требований к системе в виде одного или нескольких предложений. 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 впоследствии тратить ресурсы на их поддержание. Цель пользовательских историй — быть в состоянии 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.

Преимущества пользовательских историй

1. Истории представляют маленькие кусочки бизнес-ценности, которые реализуются в период от нескольких дней до нескольких недель.
2. Позволяют разработчикам и клиентам обсуждать требования на протяжении всей «жизни» проекта.
3. Нуждаются в небольшом обслуживании.
4. Рассматриваются только в момент использования.
5. Поддерживают близкий контакт с клиентом.
6. Позволяют разбить проект на небольшие этапы.
7. Подходят для проектов, где требования изменчивы или плохо поняты.
8. Облегчают оценку заданий.

Недостатки пользовательских историй

1. Без конкретных приёмочных испытаний они открыты для различных интерпретаций, что усложняет их использование как основу для соглашения. Таким образом, нарушается принцип однозначности.
2. Требуют близкого контакта с клиентом на протяжении всего проекта, что может быть сложно или приводить к накладным затратам.
3. Плохо масштабируются на больших проектах — историй становится слишком много, сложно выделить самые приоритетные.
4. Истории могут быть сложными для понимания и требовать специфических знаний, или требования изменились со времени написания.
5. К каждой пользовательской истории в какой-то момент прикрепляется одно или более приёмочных тестирований. Это позволяет разработчику узнать, когда пользовательская

история готова, и как клиенту проверить это. Без точных формулировок требований в момент поставки продукта часто возникают длительные неконструктивные разногласия.

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.

Пользовательские сценарии 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 1. Администратор выбирает учётную запись пользователя и нажимает кнопку «Активировать» 2. Система переключает учётную запись в статус «Активный» 3. Система отправляет сообщение пользователю на email.	
Результат	Учётная запись пользователя перешла в статус «Активный»

Сценарий — диаграмма

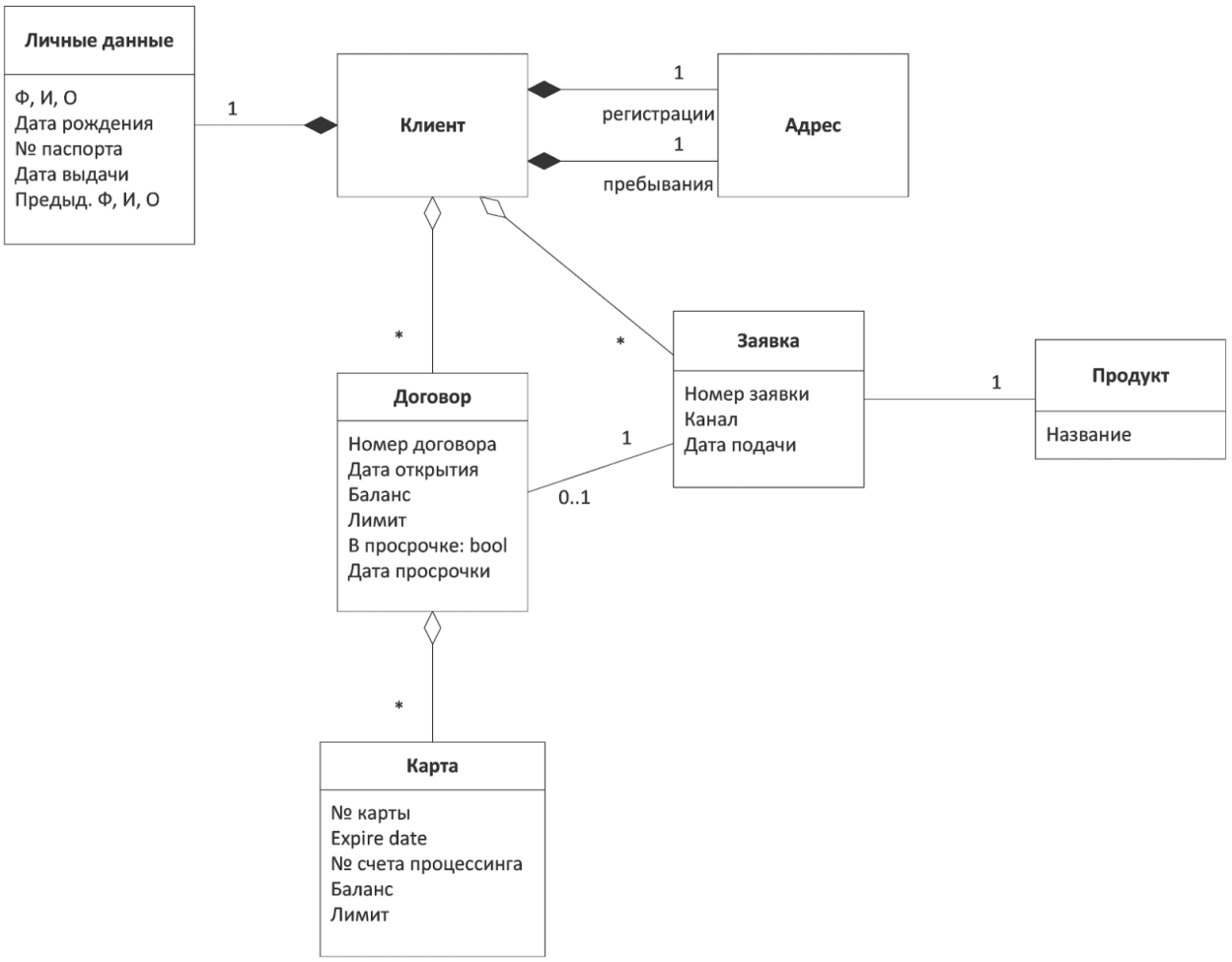
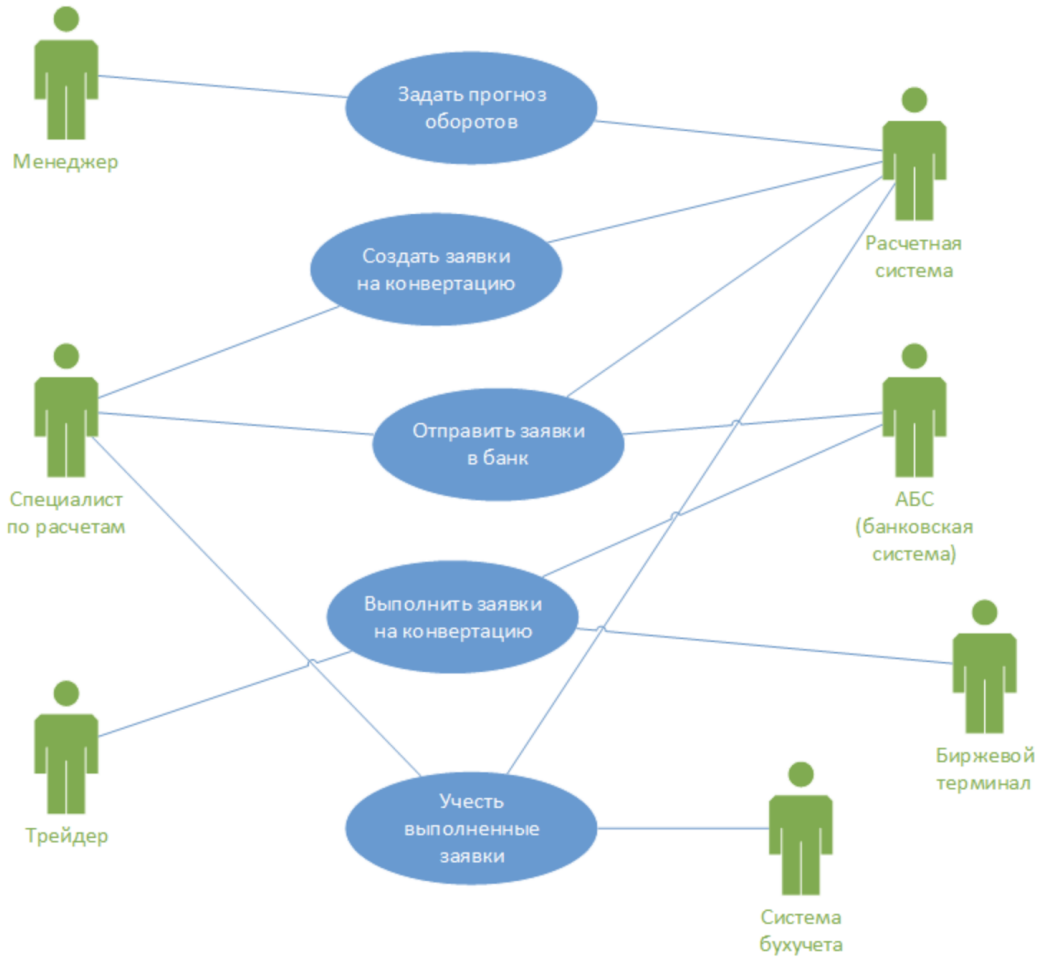


Диаграмма написана с использованием нотации UML. Подробнее о ней — [здесь]



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

Пользовательские сценарии решают различные задачи:

Оценивать трудоёмкость проекта

Оценка трудоёмкости 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.

Планировать график работ

После того как 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.

Выявлять пропущенные требования

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 серьёзно повлияет на планирование.

Системы управления требованиями

Система управления требованиями — requirements management systems, RMS — 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 этой статье.

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.

Критерии выбора системы, чтобы управлять требованиями

1. Возможность импорта существующих требований в систему.
2. Гибкая конфигурация атрибутов для различных типов требований.
3. Возможность создания версий и меток.
4. Возможность связи требований между собой.
5. Интеграция с другими системами, возможность связи требований с объектами внутри этих систем.
6. Управление доступом.
7. Экспорт и подготовка отчётной документации.

8. Система запросов на изменение.
9. Возможность включать объекты (файлы, графику, диаграммы) в текст требований.
10. Возможность обсуждения требований.
11. Автоматические нотификации.
12. Веб-интерфейс.
13. Удобство использования.
14. Примеры проектов, методические материалы, документация, доступность обучающих курсов.
15. Поддержка со стороны компании-разработчика.

Confluence

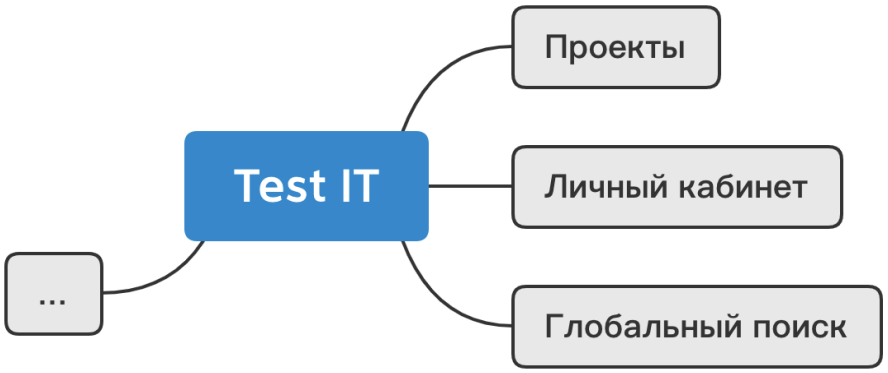
Confluence — внутренняя вики-система для организаций, разработанная, чтобы создать единую базу знаний. Продукт написан на Java и разрабатывается компанией Atlassian. Распространяется под проприетарной лицензией, бесплатно для некоммерческих организаций и открытых проектов.

Confluence позволяет решать следующие 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.
9. Прочее.

Создать аккаунт и начать работать в Confluence

Декомпозиция требований. Mind map



Если документация объёмная и представлена в текстовом формате, в ней легко запутаться. Это ведёт к рискам пропустить дефекты. Чтобы снизить этот риск, требования надо декомпозировать и представлять в наглядной форме, например, в виде майнд-карты (mind map). Рассмотрим пример декомпозиции и составим майнд-карту на основе требований к TMS Test IT.

Уровни декомпозиции

1 уровень. Крупные блоки или компоненты

Название уровня говорит само за себя. На этом уровне выделяются относительно крупные модули или подсистемы программы, которые могут существовать практически независимо друг от друга. На примере Test IT выделим:

1. Проекты.
2. Личный кабинет.
3. Глобальный поиск.
4. Прочее.

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

2 уровень. Страницы сайта или экраны мобильного приложения

Рассмотрим сущность «Проект». Согласно документации, при работе с проектом пользователь может попасть на страницы:

1. Общие настройки.
2. Редактор атрибутов.

3. Настройки доступа.
4. Команда проекта.

Добавим их в нашу майнд-карту.

3 уровень. Содержание экранов

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

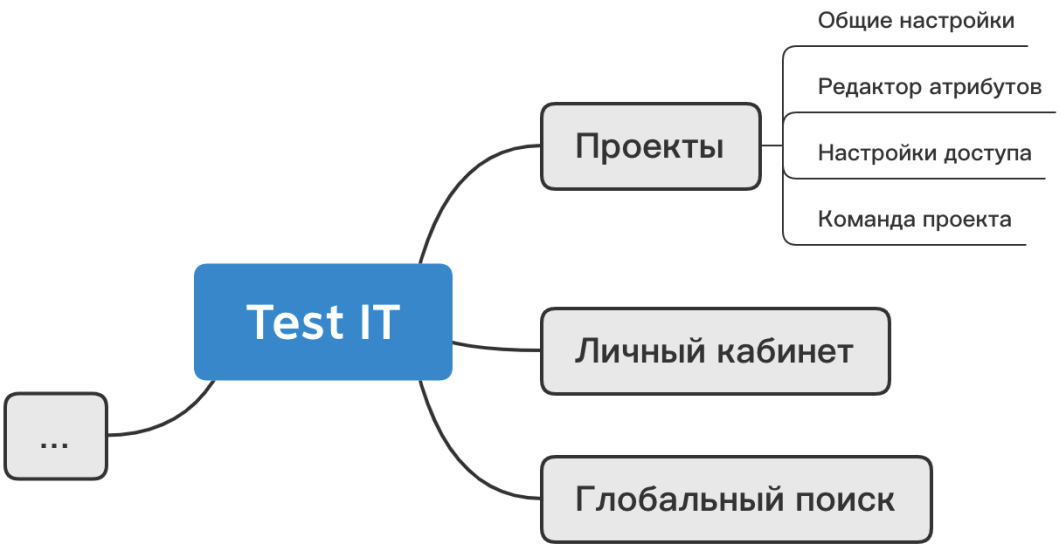
На третьем уровне декомпозиции важно отметить:

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

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

Рассмотрим экран «Общие настройки». На нём — два поля:

1. Название — обязательное.
2. Описание — необязательное, до 256 символов.



Эти поля похожи, т. к. в них вводится текстовое значение. Подумаем, что это может быть, и попытаемся немного сломать систему.

1. Название.

a. 0 символов (оставить пустым).

b. 30 символов (типичное значение).

c. 1 тыс. символов (очень длинное).

d. На русском языке.

e. На английском языке.

f. На китайском языке.

g. Буквы, цифры, спецсимволы.

h. Несколько пробелов.

i. Ручной ввод.

j. Копипаст.

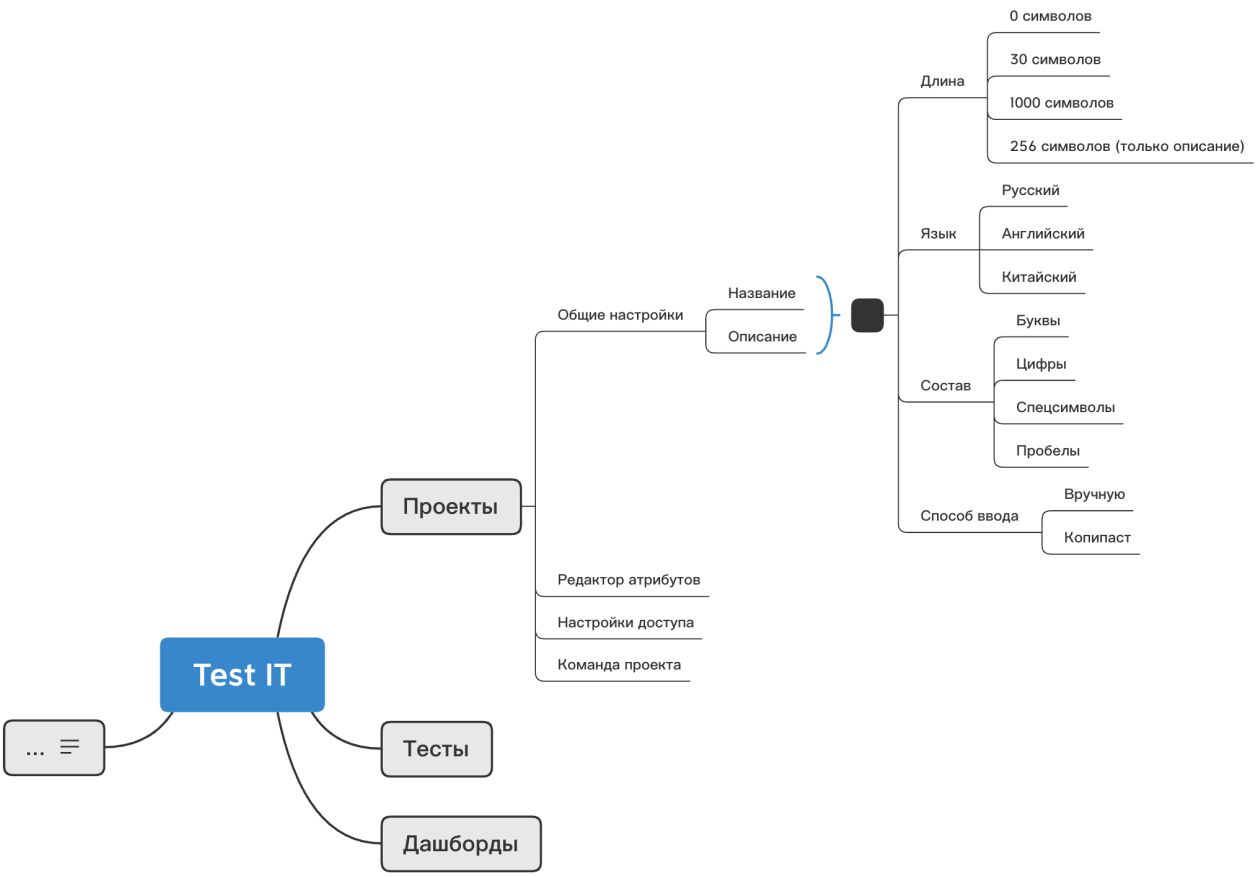
2. Описание.

    a. Всё то же самое, что и для названия.

    b. Ровно 256 символов. Почему это важно, узнаем на следующих занятиях.

Добавим всё это на нашу майнд-карту.

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, т. к. майнд-карта станет основой для дальнейшего составления чек-листов и тест-кейсов.



Важно! *Элементы интерфейса имеют очень интересные названия.*

Например, разные кнопки могут называться:

1. Кнопка.
2. Radio button.
3. Burger menu.
4. Тоггл.

Типы списков:

- чек-лист;
- аккордеон;
- select;
- dropdown.

Кроме бургеров и аккордеонов, на странице иногда появляются хлебные крошки, карусели, маски и другое. Подробнее — по ссылке 1 и ссылке 2.

Глоссарий

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

Бизнес-требования — это обобщённое описание функций продукта без технической детализации. Они 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 в виде одного или нескольких предложений. 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 — это описание взаимодействия участников, как правило, пользователя и системы. Количество участников — от 2 и больше. Пользователь — человек или другая система.

Система управления требованиями — requirements management systems, RMS — 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?

Практическое задание

1. Изучите требования к TMS Test It.
2. Скачайте заготовку майнд-карты по этой ссылке.
3. Добавьте два объекта первого уровня.
4. Для любого из них сделайте детализацию второго уровня.
5. Одному объекту второго уровня сделайте детализацию третьего уровня (объект, действия, параметры).

Дополнительные материалы

1. Виды RMS.
2. Сайт Atlassian.
3. Документация Test IT.
4. Названия элементов интерфейса.
5. Ещё о названиях элементов интерфейса.
6. Скачать xmind можно тут.

Используемые источники

1. Статья «Требования к ПО на пальцах».
2. Статья «Как правильно писать User Stories: руководство для разработчиков».
3. Статья «Работаем с User Stories».
4. Статья «Декомпозиция».