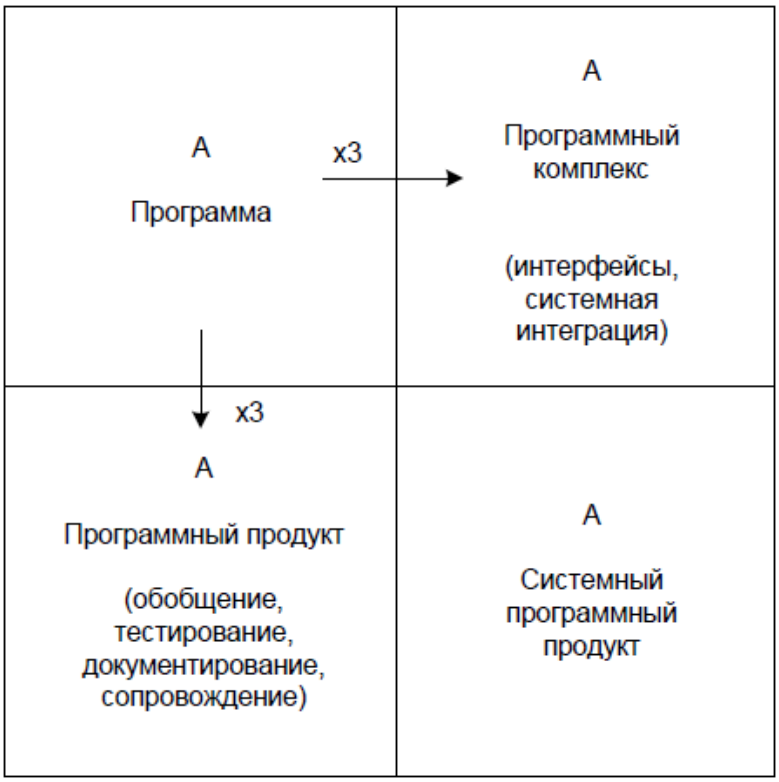
Раздаточный материал № 1



Раздаточный материал № 2 (самостоятельно)

Вспомогательные процессы

Процесс документирования. В процессе разработки и далее исполнитель пишет документацию и руководства пользователя к разрабатываемому программному продукту. Данные документы помогут разработчикам [вспомнить/разобраться] структуру и код ПО (ибо со временем всё забывается, особенно в больших проектах), а пользователям освоить работу с программой.

Процесс управления конфигурацией. Данный процесс включается в себя работы по управлению наборами разрабатываемых компонентов ПО и по управлению версиями ПП.

Процесс обеспечения качества. Он отвечает за то, чтобы разрабатываемый программный продукт соответствовал предварительным требованиям к разработке, а также стандартам организаций исполнителя и заказчика.

Процесс верификации. Нужен для того, чтобы выявить ошибки, внесённые в ПО во время конструирования, а также выявить несоответствия разрабатываемого ПО выработанной архитектуре.

Процесс аттестации. Процесс направлен на подтверждение соответствия получаемых величин эталонным. То есть, выходные данные должны иметь погрешность, удовлетворяющую требованиям и установленным стандартам.

Процесс совместной оценки. Нужен для контроля и проверки состояния персонала и разрабатываемого программного продукта. Выполняется обеими сторонами (заказчиком и исполнителем) на протяжении времени всех работ по проекту.

Процесс аудита. Аудит направлен на независимую оценку текущих положений, состояния проекта, документации и отчетов. При аудите выполняется сравнение с договором и документами, определяющими стандарты. Может выполняться также обеими сторонами.

Процесс разрешения проблем. Реализует устранение недочётов, выявленных во время всех процессов, связанных с контролем и оценкой.

Организационные процессы жизненного цикла программного продукта

Существует и проводится ряд мер, направленных на повышение организации и качества разработки программного обеспечения. Они называются организационными процессами жизненного цикла.

Процесс управления, который направлен на грамотное и эффективное управлением персонал компании-исполнителя. За это отвечают люди, находящиеся на руководящих постах, а также специальный отдел в фирме.

Процесс создания инфраструктуры. Разработка программных продуктов требует наличия огромного количества инфраструктурных компонентов: компьютеров, серверов, специальных программ для разработки и т.д. Кроме того, готовый продукт требует наличия определённых единиц для его работы. Данный процесс необходим для подготовки оборудования и ПО для разработчиков, а также для успешного функционирования готового ПП у заказчика.

Процесс усовершенствования. Направлен на усовершенствование всех остальных процессов жизненного цикла программного обеспечения. Усовершенствование может повысить производительность разработчиков и добиться большей выгоды от выполнения заказа на производство программы.

Процесс обучения. Постоянное обучение сотрудников и повышение их квалификации — это залог производства качественных продуктов и программ. Процесс обучения направлен на организацию мероприятий для повышения уровня и получения новых навыков сотрудниками компании-разработчика.

Раздаточный материал № 3

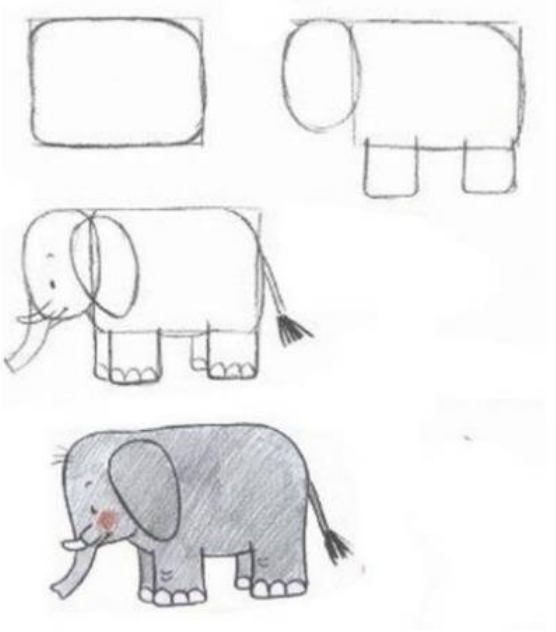
Software Development Life Cycle (SDLC) – жизненный цикл программного обеспечения

Раздаточный материал № 4

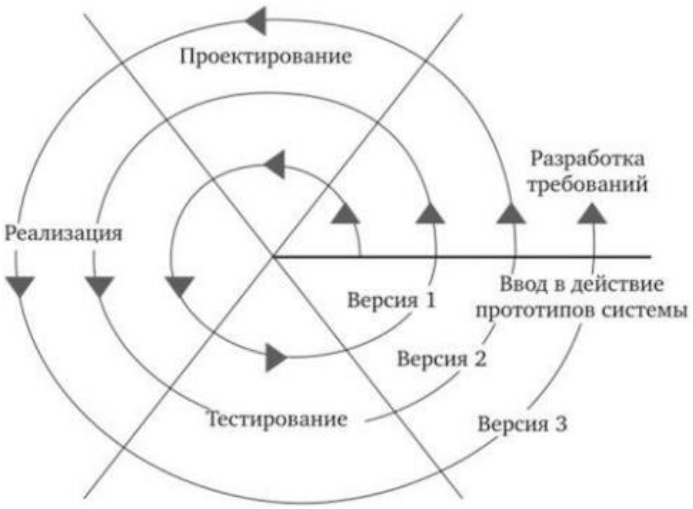
Раздаточный материал № 5

Раздаточный материал № 6

Раздаточный материал № 7



Раздаточный материал № 8



Раздаточный материал № 9 (самостоятельно)

Основополагающие принципы Agile Manifesto:

наивысшим приоритетом признается удовлетворение заказчика за счёт ранней и бесперебойной поставки ценного программного обеспечения;

изменение требований приветствуется даже в конце разработки (это может повысить конкурентоспособность полученного продукта);

частая поставка работающего программного обеспечения (каждые пару недель или пару месяцев с предпочтением меньшего периода);

общение представителей бизнеса с разработчиками должно быть ежедневным на протяжении всего проекта;

проекты следует строить вокруг заинтересованных людей, которых следует обеспечить нужными условиями работы, поддержкой и доверием;

самый эффективный метод обмена информацией в команде — личная встреча;

работающее программное обеспечение — лучший измеритель прогресса;

спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;

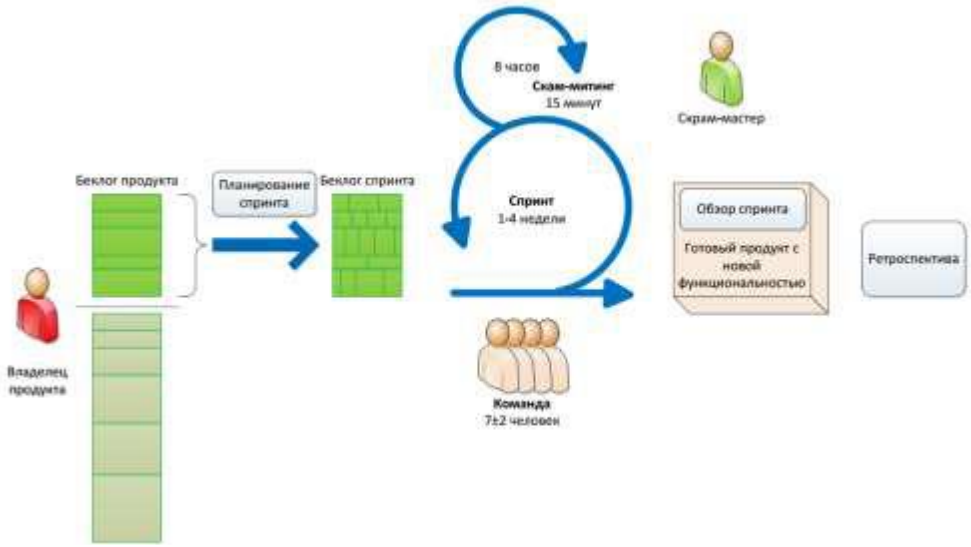
постоянное внимание к техническому совершенству и хорошему проектированию увеличивают гибкость;

простота, как искусство не делать лишней работы, очень важна;

лучшие требования, архитектура и проектные решения получаются у самоорганизующихся команд;

команда регулярно обдумывает способы повышения своей эффективности и соответственно корректирует рабочий процесс.

Раздаточный материал № 10



Раздаточный материал № 11

Роли:

- Scrum мастер (Scrum Master),
- Владелец продукта (Product Owner),
- Команда (Team).

Раздаточный материал № 12

Артефакты:

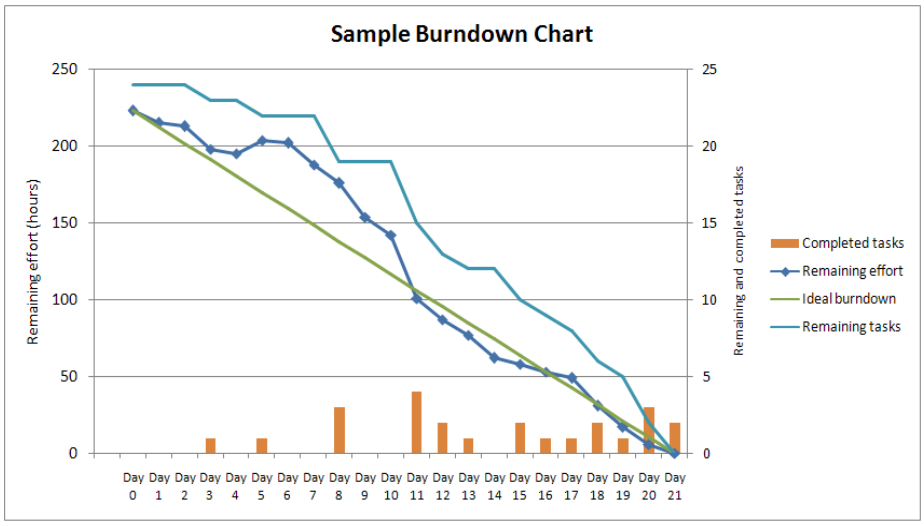
- Product Backlog (Бэклог Продукта),
- Sprint Backlog (Бэклог Спринта),
- Increment (Инкремент).

Раздаточный материал № 13

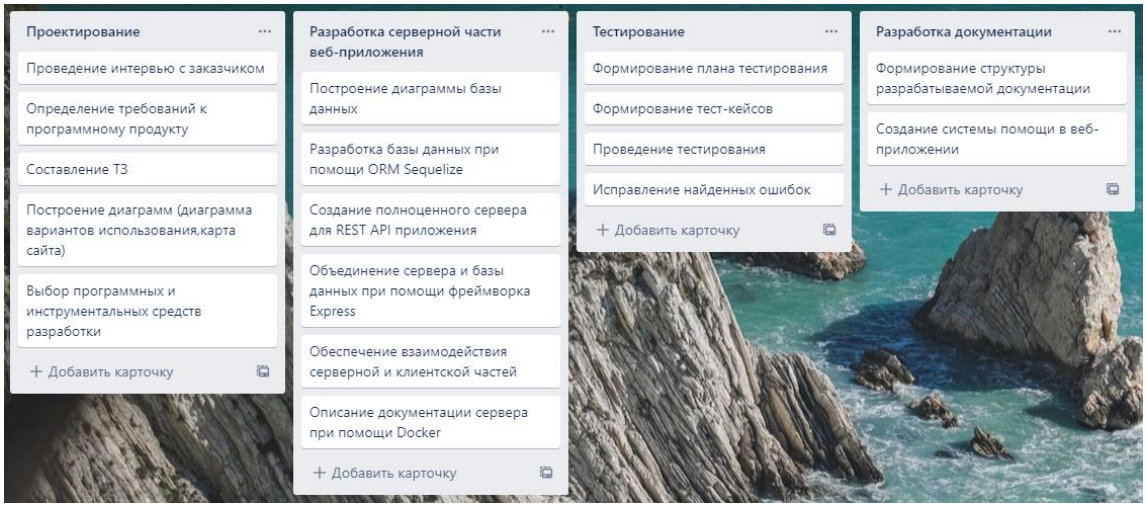
	A	B
1	**Приоритет**	**Компонент**
2	**100**	Каталог товаров
3	**99**	Корзина заказа
4	**80**	Оплата по Робокассе
5	**70**	Форма расчета стоимости доставки
6	**65**	Фильтр товаров
7	**60**	Форма обратного звонка
8	**50**	Личный кабинет
9	**40**	Поиск как на эппл.ком :)
10	**20**	Система отзывов
11	**16**	Анимированное добавление товаров в корзину
12	**15**	3d-крутилка товаров
13	**10**	Социальная сеть

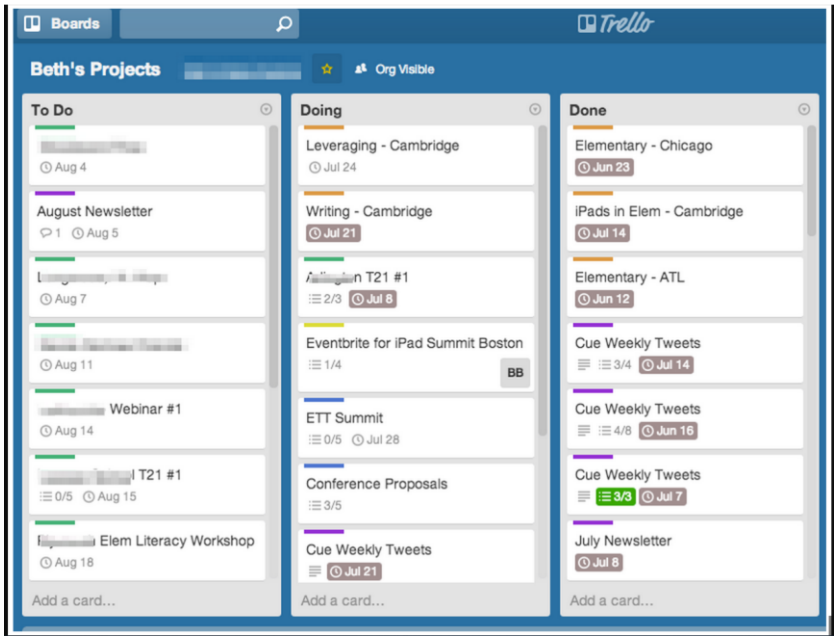
Раздаточный материал № 14

Раздаточный материал № 15



Раздаточный материал № 16

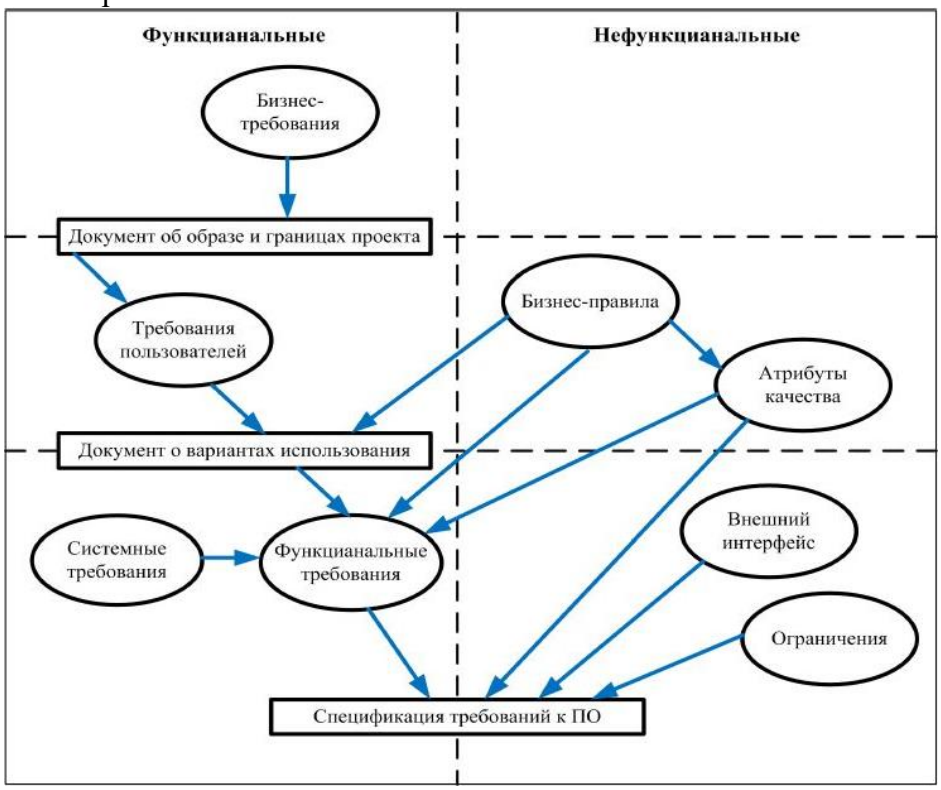




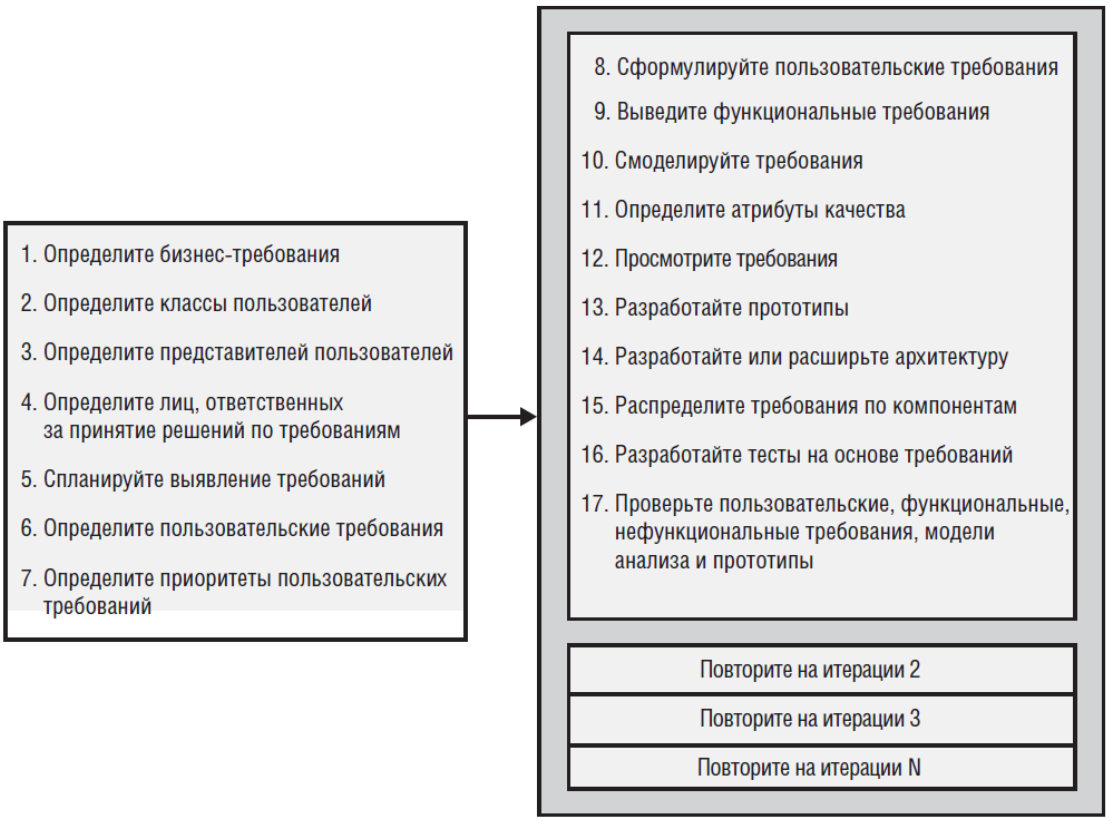
БЕЗ ТЗ – РЕЗУЛЬТАТ ХЗ

Раздаточный материал № 17

Типы требований:



Раздаточный материал № 18



Раздаточный материал № 19

Раздаточный материал № 20 (самостоятельно)

Выбор сторонника продукта (product champion) в каждом классе пользователей. Это человек, который сможет точно передавать настроения и нужды клиентов. Он представляет потребности определенного класса пользователей и принимает решения от их лица.

Проведение фокус-групп типичных пользователей. Создайте группы типичных пользователей предыдущих версий вашего продукта или похожих. Выясните у них подробности функциональности и качественных характеристик разрабатываемого продукта. Фокус-группы особенно ценны при разработке коммерческих продуктов, когда приходится иметь дело с большой и разнородной клиентской базой. В отличие от сторонников продукта, у фокус-групп обычно нет полномочий на принятие решений.

Проведение совместных семинаров. Совместные семинары по выявлению требований, где тесно сотрудничают аналитики и клиенты — отличный способ выявить нужды пользователей и составить наброски документов с требованиями.

Раздача опросных листов. Это один из способов определения потребностей больших групп пользователей. Опросные листы удобны при работе с любыми большими группами пользователей, но особенно полезны в распределенных группах. Качественные вопросы позволяют быстро выявить аналитическую информацию о потребностях. На основе результатов опросных листов можно более целенаправленно применять дополнительные усилия.

Анализ документов. Имеющаяся документация может помочь выявить, как системы работают сейчас или что они должны делать. К документации относится вся письменная информация о текущих системах и бизнес-процессах, спецификации требований, исследования конкурентов и руководства имеющихся коммерческих программных пакетов. Просмотр и анализ этих документов может помочь выявить функциональность, которая должна остаться и которая больше не нужна, а также определить, как сейчас люди выполняют свою работу, что предлагают конкуренты и что говорят поставщики о том, что должно делать их ПО.

Изучение отчетов о проблемах работающих систем с целью поиска новых идей. Поступающие от клиентов отчеты о проблемах и предложения о расширении функциональности — отличный источник идей о возможностях, которые можно реализовать в следующей версии или новом продукте. За подобной информацией стоит обратиться к персоналу службы поддержки.

Повторное использование требований. Если необходимая клиенту функциональность аналогична уже реализованной в другом продукте, подумайте, готовы ли клиенты гибко пересмотреть свои требования для использования существующих компонентов. Требования, соответствующие бизнес-правилам компании, можно применить в нескольких проектах, например, требования к безопасности, глоссарии, модели и определения данных, профили заинтересованных лиц, описаний классов и архетипы пользователей.

Раздаточный материал № 21

<Роль> должен иметь возможность <возможность> в <показатель производительности> с <момент отсчета> в <условия эксплуатации>, чтобы <ценность>

Например, Администратор клиники должен иметь возможность просмотреть данные о прошлых и запланированных посещениях пациента в течение 3 секунд после определения личности клиента по номеру телефона входящего звонка, чтобы добавить новое или изменить запланированное посещение.

<Система> должна <выполняемая функция> <объект> каждые <производительность> <единица измерения>, чтобы <ценность>.

Например, CRM-система должна отправлять СМС-напоминание клиенту о предстоящем посещении за сутки перед посещением, чтобы он помнил о приеме и пришел вовремя.

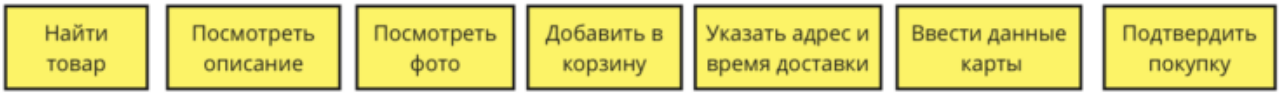
Как администратор базы данных, я хочу автоматически объединять наборы данных из разных источников, чтобы мне было проще создавать отчеты для моих внутренних клиентов.

Как руководитель удаленной группы, я хочу, чтобы в наше приложение для обмена сообщениями для команды было включено совместное использование файлов и аннотации, чтобы моя команда могла сотрудничать в режиме реального времени и хранить архив своей работы в одном месте.

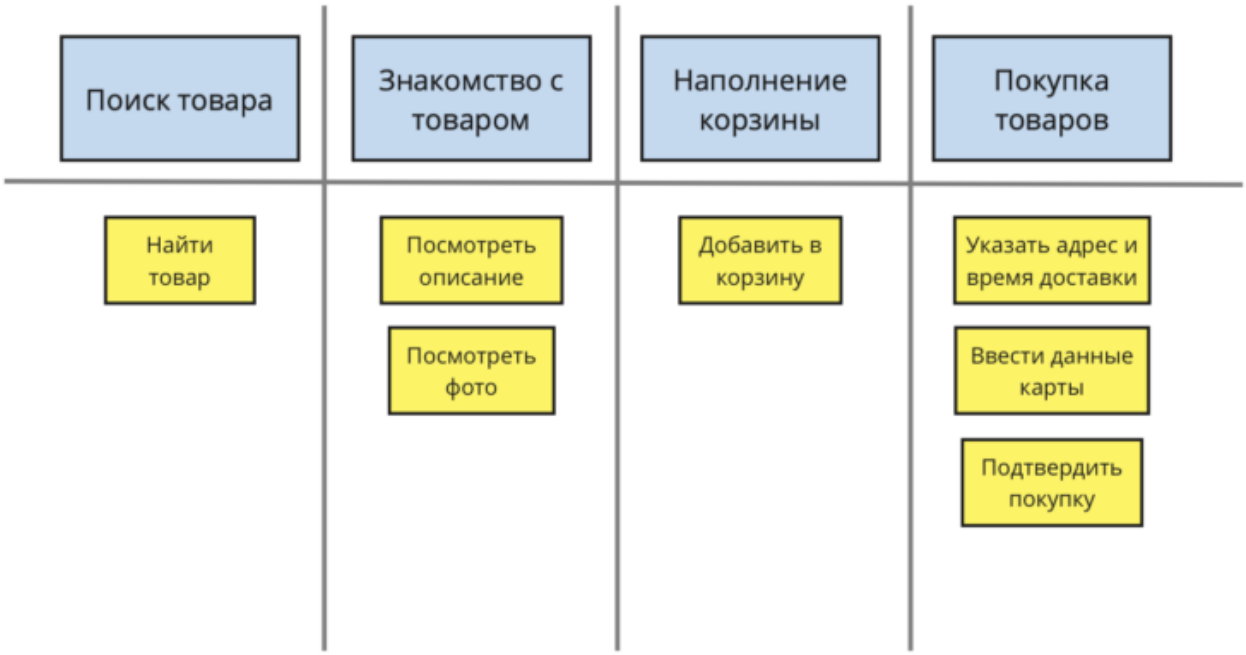
Раздаточный материал № 22

Магазин цветов решил запустить сайт. Визуализируем опыт клиентов с помощью техники USM.

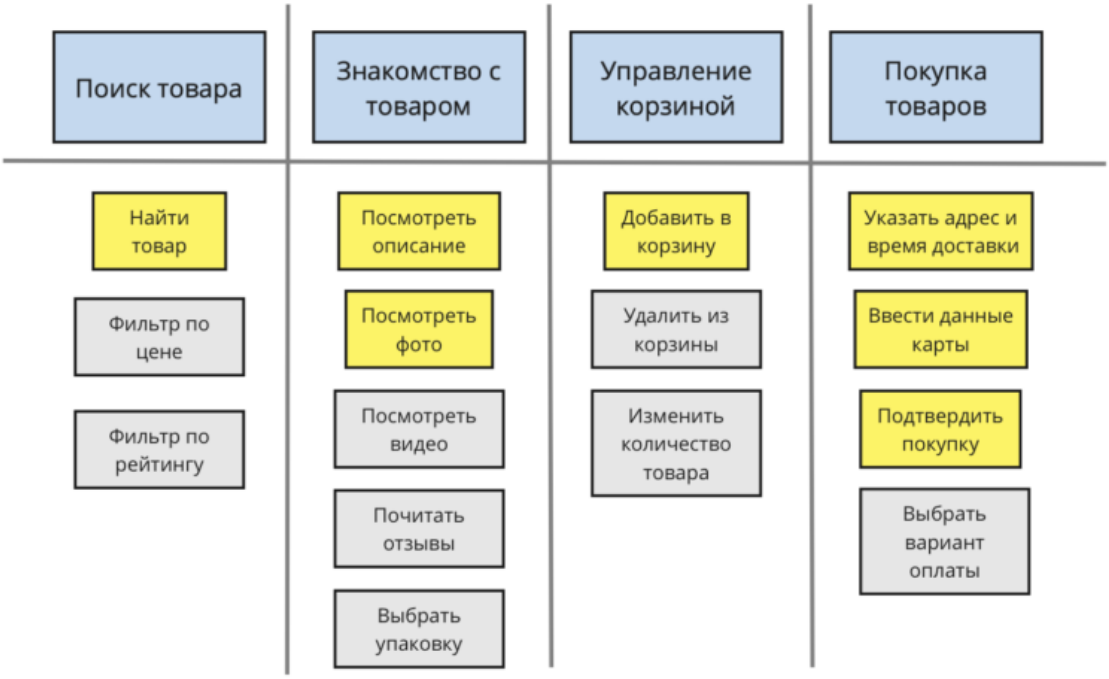
1. Расскажите историю клиента по шагам



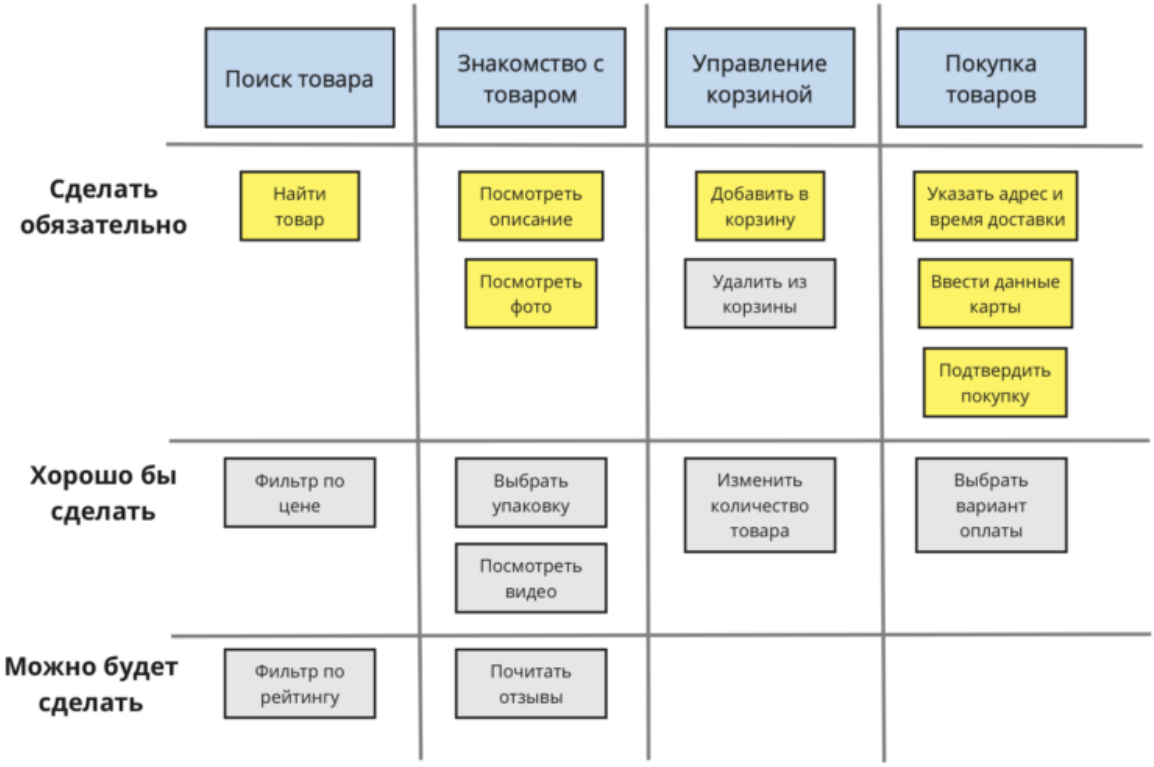
2. Сгруппируйте действия клиента в этапы



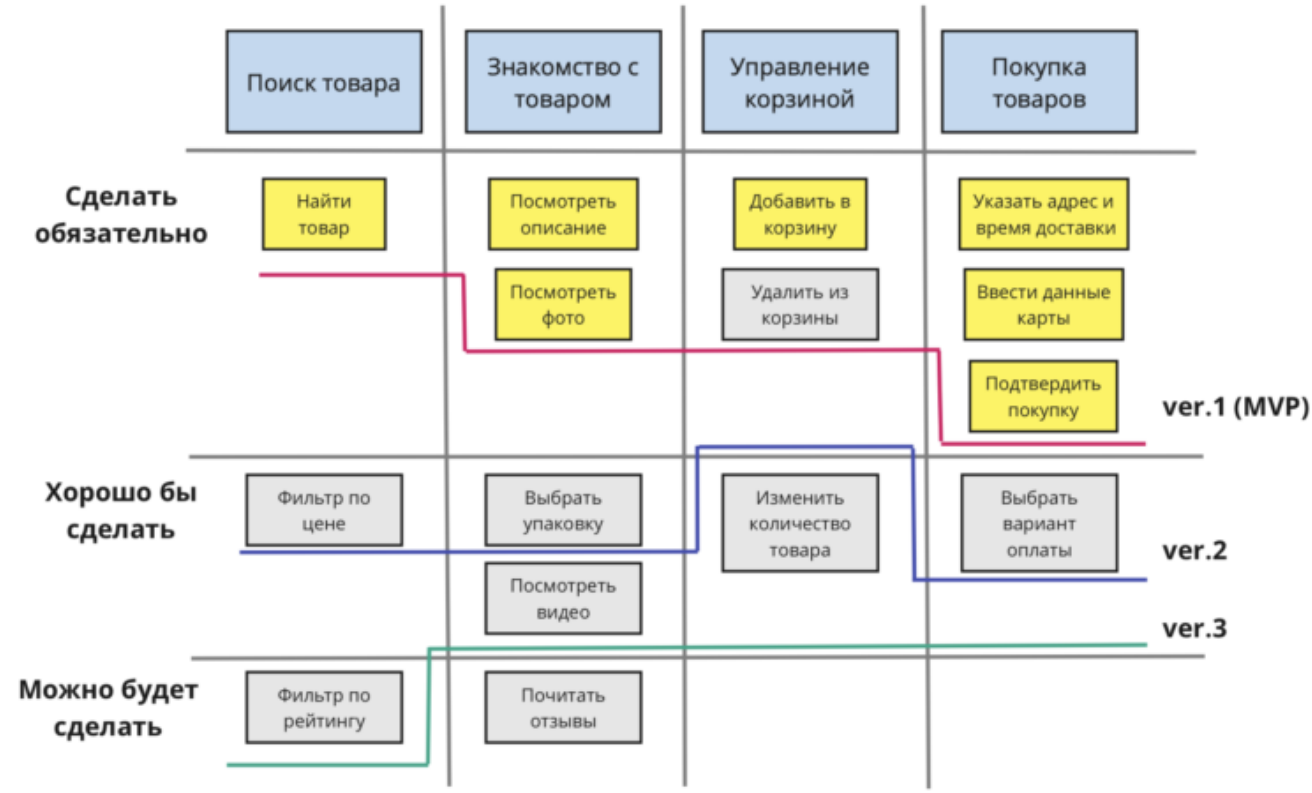
3. Заполнение пробелов в истории



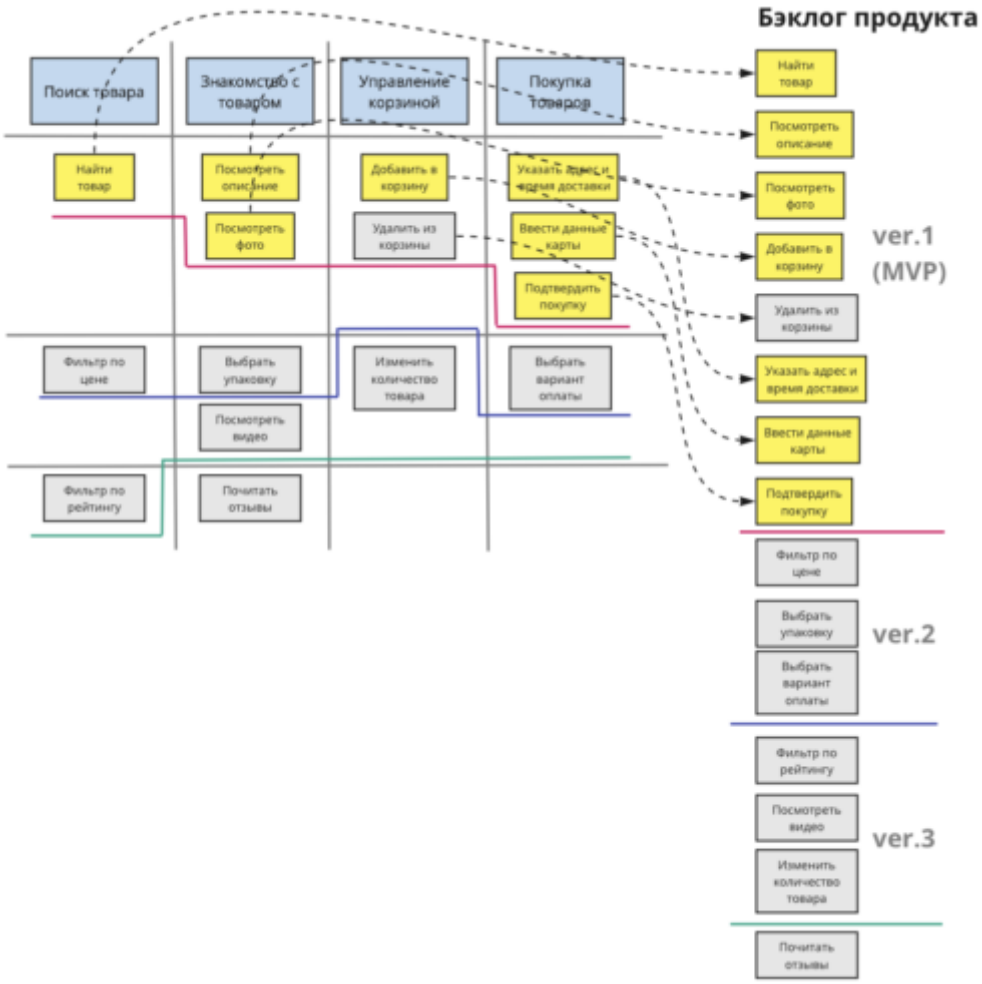
4. Приоритезируйте истории внутри каждого этапа пути



5. Выделите релизы



6. Получите приоритизированный бэклог


Поиск товара
Знакомство с товаром
Управление корзиной
Покупка товаров
Найти товар
Посмотреть описание
Посмотреть фото
Добавить в корзину
Удалить из корзины
Указать адрес и время доставки
Ввести данные карты
Подтвердить покупку
Фильтр по цене
Выбрать упаковку
Посмотреть видео
Изменить количество товара
Выбрать вариант оплаты
Фильтр по рейтингу
Почитать отзывы
Бэклог продукта
Найти товар
Посмотреть описание
Посмотреть фото
Добавить в корзину
Удалить из корзины
Указать адрес и время доставки
Ввести данные карты
Подтвердить покупку
ver.1 (MVP)
Фильтр по цене
Выбрать упаковку
Выбрать вариант оплаты
ver.2
Фильтр по рейтингу
Посмотреть видео
Изменить количество товара
Почитать отзывы
ver.3

Раздаточный материал № 23

История: как пользователь мессенджера, я хочу совершить видео-звонок другому пользователю, чтобы провести онлайн-беседу.

Название: сделать видео-звонок другому пользователю.

Цель: провести онлайн-беседу с другим пользователем.

Действующие лица: пользователи мессенджера.
1-й пользователь – инициатор звонка, 2-й пользователь – принимающий звонок.

Предварительные условия:

1. Оба пользователя имеют установленный мессенджер на своих устройствах.
2. Оба пользователя зарегистрированы в мессенджере.
3. Оба пользователя в сети и подключены к Интернету.
4. Микрофоны и веб-камеры подключены и распознаются мессенджером у всех пользователей.
5. Оба пользователя находятся в списке контактов друг друга.

Успешный сценарий:

1. 1-й пользователь выбирает 2-го из списка контактов для видео-звонка.
2. 1-й пользователь нажимает кнопку начала видео-звонка.
3. Система начинает звонить 2-му пользователю.
4. Система показывает 1-му пользователю, что 2-й пользователь получает ходящий вызов.
5. 2-й пользователь принимает входящий вызов.
6. Система показывает пользователям экран видео-звонка, чтобы оба слышали и видели друг друга.
7. Система завершает соединение после окончания сеанса видео-звонка.

Альтернативный путь № 1:

Название: у 2-го пользователя уже идет другой звонок.

1. Система прекращает вызов от 1-го пользователя.
2. Система показывает 1-му пользователю уведомление, что у 2-го пользователя сейчас занято.
3. Система показывает 2-му пользователю, что у него был входящий звонок от другого пользователя.
4. Система перенаправляет 1-го пользователя на список его контактов.

Альтернативный путь № 2:

Название: ...

1. ...

Раздаточный материал № 24

ГОСТ 34.602-89 выделяет следующие разделы в ТЗ на создание АС	**ГОСТ 19.201-78 в ТЗ на разработку программы**	**IEEE 830-1998**
1. Общие сведения 2. Назначение и цели создания системы 3. Характеристика объектов автоматизации 4. Требования к системе a. Требования к системе в целом b. Требования к функциям, выполняемым системой c. Требования к видам обеспечения 5. Состав и содержание работ по созданию системы 6. Порядок контроля и приёмки системы 7. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие 8. Требования к документированию 9. Источники разработки	1. Введение 2. Основания для разработки 3. Назначение разработки 4. Требования к программе или программному изделию 5. Требования к программной документации 6. Технико-экономические показатели (можно взять из ТЭО) 7. Стадии и этапы разработки 8. Порядок контроля и приемки 9. Приложения	1. Введение a. Назначение b. Область действия c. Определения, акронимы и сокращения d. Ссылки e. Краткий обзор 2. Общее описание a. Взаимодействие продукта с другими продуктами и компонентами b. Функции продукта (краткое описание) c. Характеристики пользователя d. Ограничения e. Допущения и зависимости 3. Детальные требования a. Требования к внешним интерфейсам i. Интерфейсы пользователя ii. Интерфейсы аппаратного обеспечения iii. Интерфейсы программного обеспечения iv. Интерфейсы взаимодействия b. Функциональные требования c. Требования к производительности d. Проектные ограничения (и ссылки на стандарты) e. Нефункциональные требования (надежность, доступность, безопасность и пр.) f. Другие требования 4. Приложения 5. Алфавитный указатель

Раздаточный материал № 25

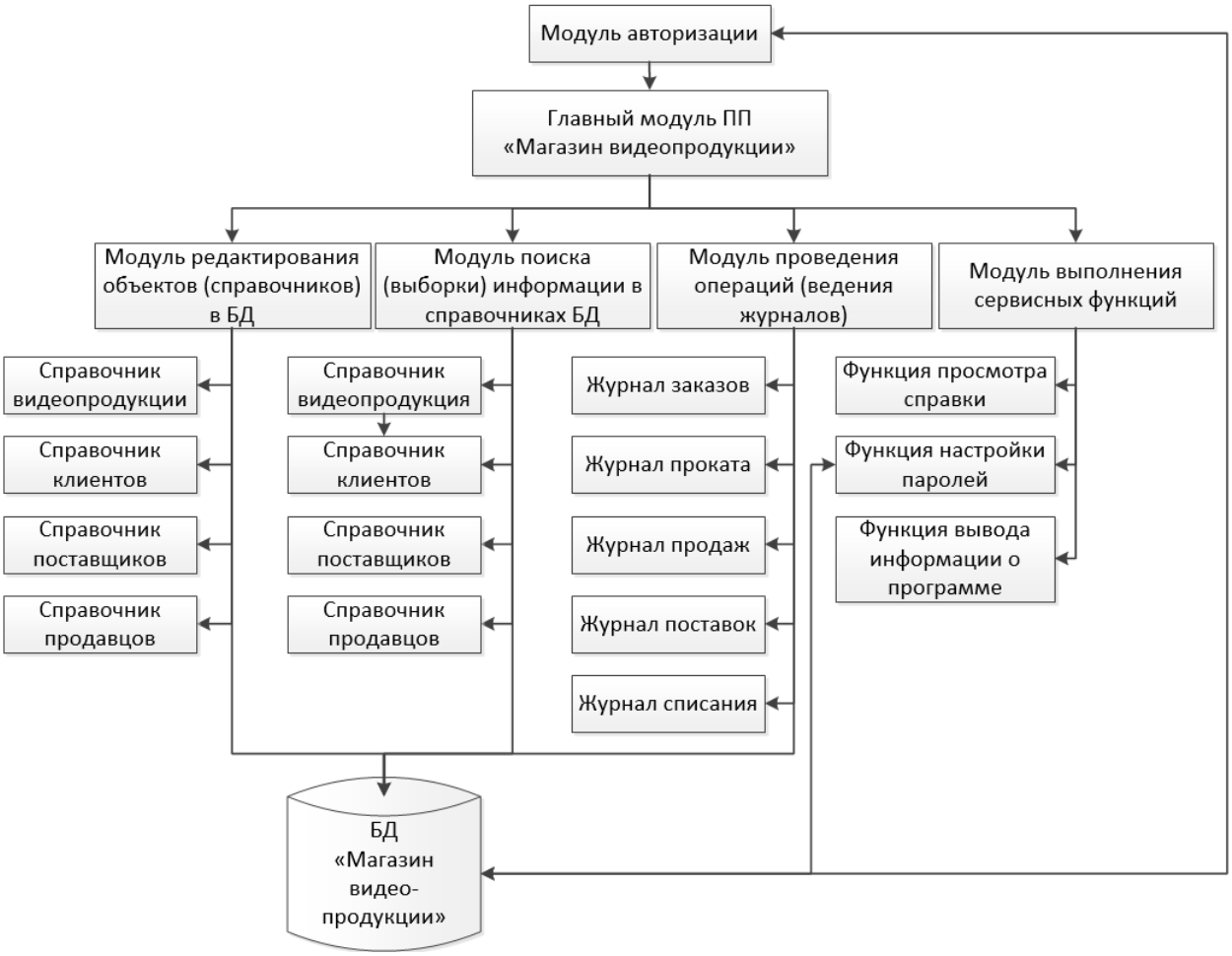
1. Введение
   1.1. Цели
   1.2. Соглашения о терминах
   1.3. Предполагаемая аудитория и последовательность восприятия
   1.4. Масштаб проекта
   1.5. Ссылки на источники
2. Общее описание
   2.1. Видение продукта
   2.2. Функциональность продукта
   2.3. Классы и характеристики пользователей
   2.4. Среда функционирования продукта (операционная среда)
   2.5. Рамки, ограничения, правила и стандарты
   2.6. Документация для пользователей
   2.7. Допущения и зависимости
3. Функциональность системы
   3.1. Функциональный блок X (таких блоков может быть несколько)
   3.2. Описание и приоритет
   3.3. Причинно-следственные связи, алгоритмы (движение процессов, workflows)
   3.4. Функциональные требования
4. Требования к внешним интерфейсам
   4.1. Интерфейсы пользователя (UX)
   4.2. Программные интерфейсы
   4.3. Интерфейсы оборудования
   4.4. Интерфейсы связи и коммуникации
5. Нефункциональные требования
   5.1. Требования к производительности
   5.2. Требования к сохранности (данных)
   5.3. Требования к качеству программного обеспечения
   5.4. Требования к безопасности системы
   5.5. Требования на интеллектуальную собственность
6. Прочее
   6.1. Приложение А: Глоссарий
   6.2. Приложение Б: Модели процессов и предметной области и другие диаграммы
   6.3. Приложение В: Список ключевых задач

Раздаточный материал № 26

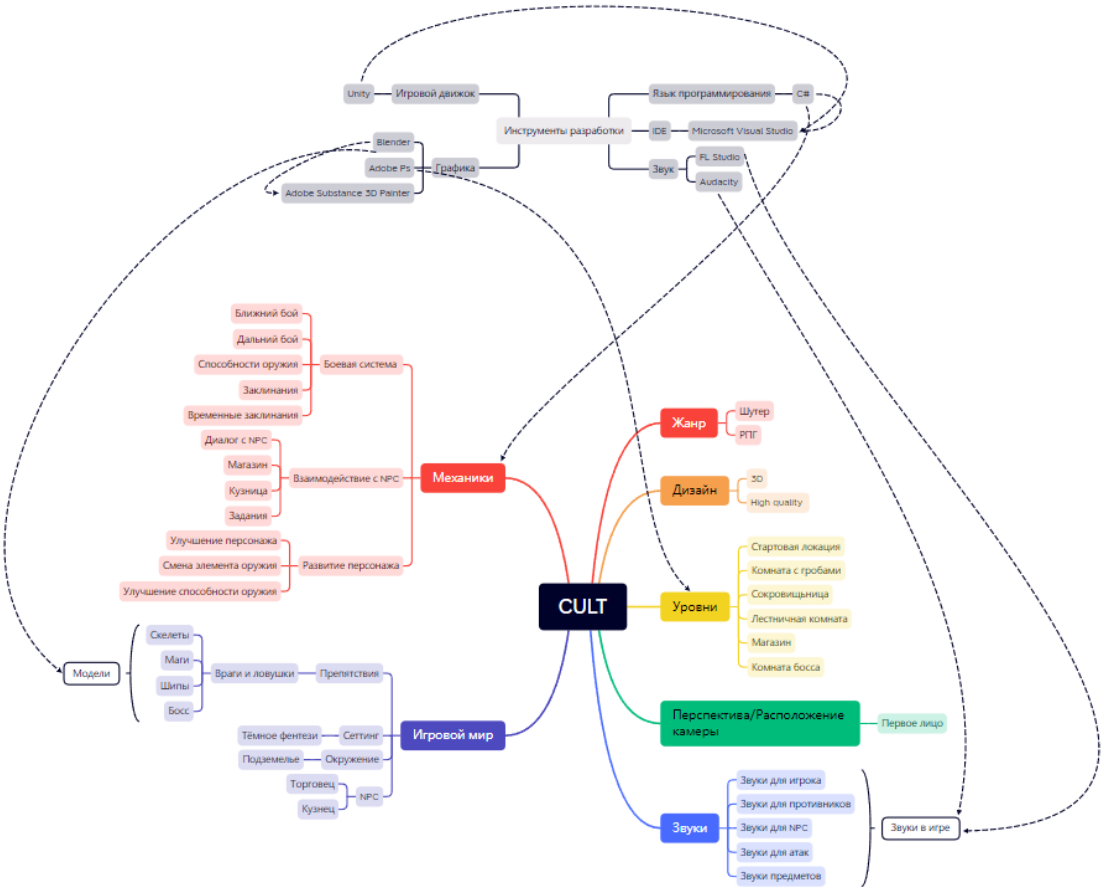
Вид требования	Неправильная формулировка	Комментарий и как можно было сформулировать
Функциональность	Сумма затрат должна корректно распределяться по соответствующим товарам	Понятное ли это требование? В общем-то понятное, речь идет о распределении неких затрат по группе товаров. Конкретное ли это требование? Не сказано, как должна распределяться затрата, по сумме, по количеству, равномерно или как-то иначе? Тестируемое ли это требование? Вроде бы простая вещь, но как ее проверять, если нет конкретики? Как можно было бы это переформулировать: «Сумма затрат, указанная в документе, должна распределиться на все товары, указанные в данном документе пропорционально стоимости этих товаров». Получилось и понятно, и конкретно. Как проверить тоже не составит труда.
Эргономичность	Программа должна иметь удобный интерфейс	Тут нет не конкретики, ни возможность проверить это требование. Хотя, безусловно, понятное (субъективно). Тут переформулировать никак нельзя, надо подробно расписывать каждый элемент «удобности». Например,: - Строки в документ должны добавляться как по нажатию на кнопку «Добавить», так и при нажатии на клавиши «insert», а также вводе пользователем части наименования; - При просмотре списка товаров должна быть возможность поиска по наименованию, штрихкоду и артикулу; - и пр.
Разграничение прав доступа	Доступ к данным по прибыли должен быть доступен только финансовому директору	Понятно? Почти. Правда, прибыль бывает разная, надо уточнить. Конкретно? Конечно нет. Как это видится в реализации? Если речь идет о валовой прибыли, то значит необходимо ограничивать доступ к данным о стоимости закупки, т.к. в противном случае валовую прибыль вычислить не составит труда, поскольку данные о стоимости реализации известны широкому кругу лиц. К тому, что относится к правам доступа, надо относиться очень аккуратно. А если у менеджеров по продажам мотивация построена на валовой прибыли, так эти требования еще и

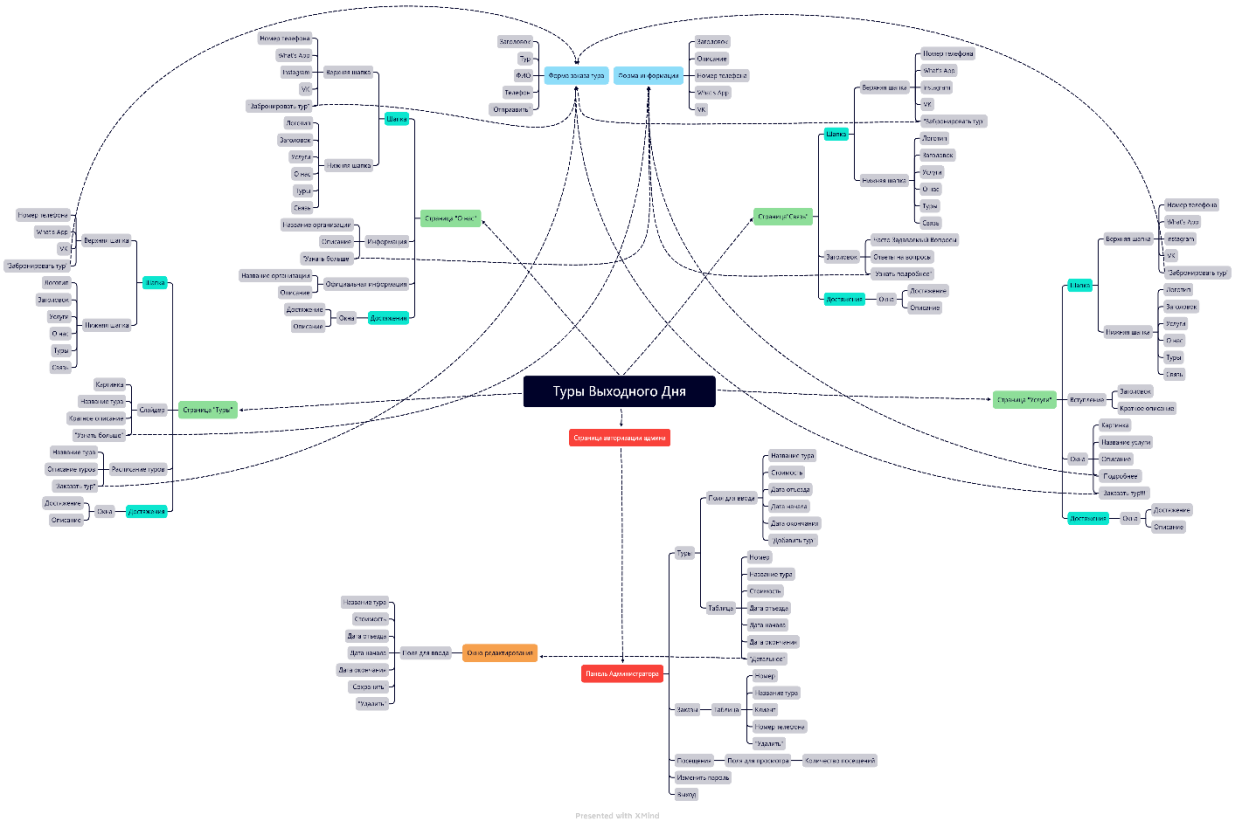
		противоречат друг другу, т.к. менеджеры никогда не смогут это проверить. Если уж включать такое требование, то нужно указывать конкретные отчеты и объекты системы, в которых указывать, какая часть данных должны быть доступна отдельным категориям лиц. И рассматривать каждый такой случай индивидуально.
Производительность	Отчет по продажам должен формироваться за 1 минуту.	Да, понятно. И даже есть конкретное ограничение по времени: 1 минута. Но не известно, какая детализация при этом предполагается: по каждому товару, группам товаров, клиентам или как-то еще? Можно сформулировать примерно так: «Отчет по продажам в разрезе клиентов с детализацией до каждой товарной позиции должен выводится не более, чем за 1 минуту при условии, что количество товаров в выборке не превышает 5000 строк».

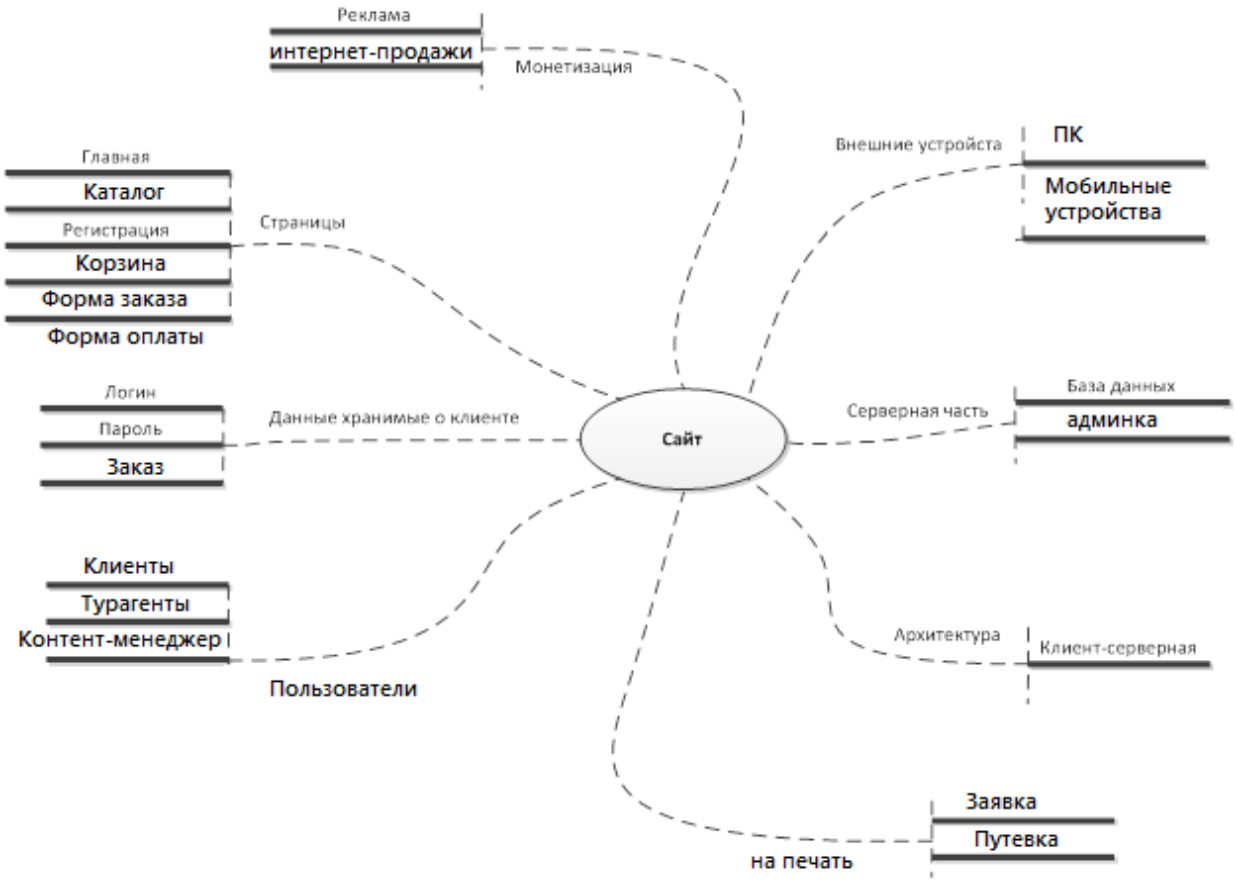
Раздаточный материал № 27



Раздаточный материал № 28





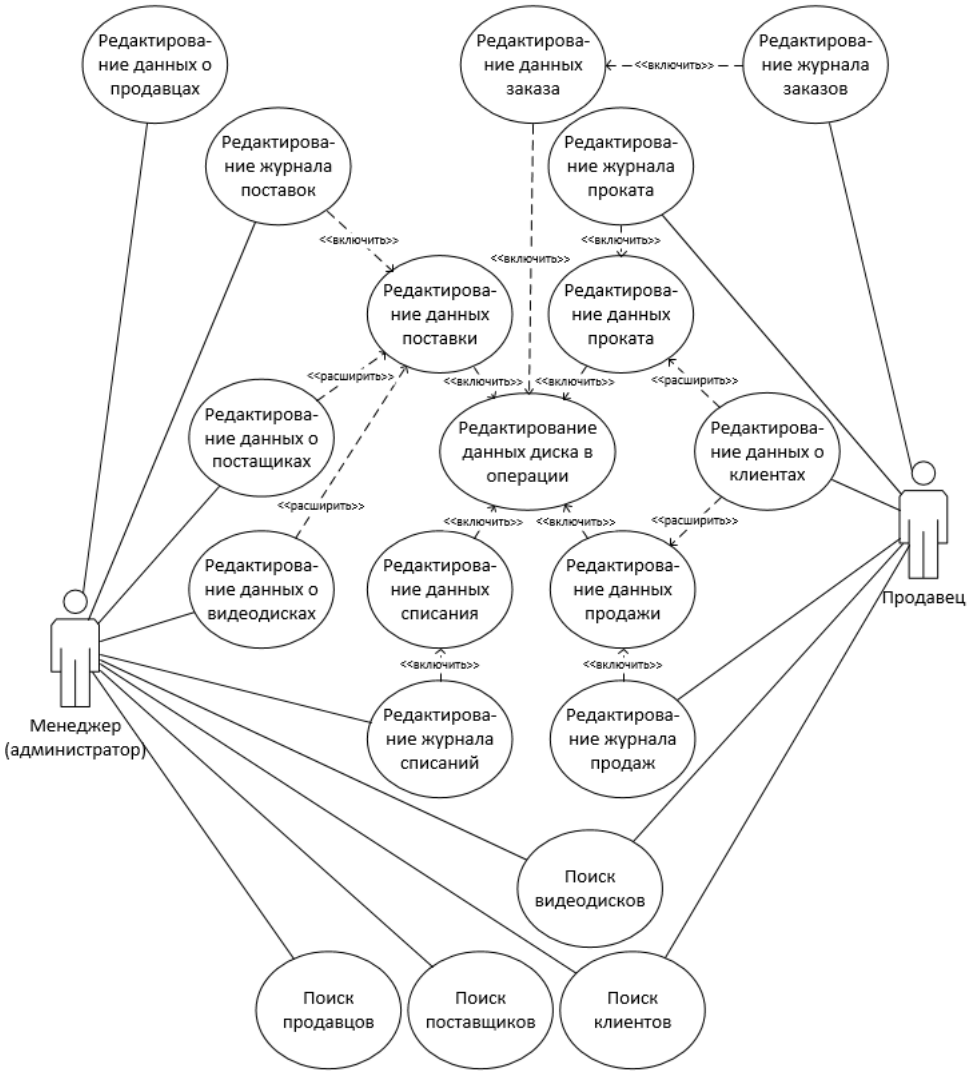


Раздаточный материал № 29

В языке UML определены следующие виды канонических диаграмм:

1. вариантов использования (use case diagram)
2. классов (class diagram)
3. кооперации (collaboration diagram)
4. последовательности (sequence diagram)
5. состояний (statechart diagram)
6. деятельности (activity diagram)
7. компонентов (component diagram)
8. развертывания (deployment diagram)

Раздаточный материал № 30



Раздаточный материал № 31 (самостоятельно)

Рекомендации по графическому изображению диаграмм языка UML

При графическом изображении диаграмм следует придерживаться следующих основных рекомендаций:

Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области. Речь идет о том, что в процессе разработки диаграммы необходимо учесть все сущности, важные с точки зрения контекста данной модели и диаграммы. Отсутствие тех или иных элементов на диаграмме служит признаком неполноты модели и может потребовать ее последующей доработки.

Все сущности на диаграмме модели должны быть одного уровня представления. Здесь имеется в виду согласованность не только имен одинаковых элементов, но и возможность вложения отдельных диаграмм друг в друга для достижения полноты представлений. В случае достаточно сложных моделей систем желательно придерживаться стратегии последовательного уточнения или детализации отдельных диаграмм.

Вся информация о сущностях должна быть явно представлена на диаграммах. В языке UML при отсутствии некоторых символов на диаграмме могут быть использованы их значения по умолчанию (например, в случае неявного указания видимости атрибутов и

операций классов), тем не менее, необходимо стремиться к явному указанию свойств всех элементов диаграмм.

Диаграммы не должны содержать противоречивой информации. Противоречивость модели может служить причиной серьезных проблем при ее реализации и последующем использовании на практике. Например, наличие замкнутых путей при изображении отношений агрегирования или композиции приводит к ошибкам в программном коде, который будет реализовывать соответствующие классы. Наличие элементов с одинаковыми именами и различными атрибутами свойств в одном пространстве имен также приводит к неоднозначной интерпретации и может быть источником проблем.

Каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов. Любые пояснительные тексты, которые не являются собственными элементами диаграммы (например, комментариями), не должны приниматься во внимание разработчиками. В то же время общие фрагменты диаграмм могут уточняться или детализироваться на других диаграммах этого же типа, образуя вложенные или подчиненные диаграммы. Таким образом, модель системы на языке UML представляет собой пакет иерархически вложенных диаграмм, детализация которых должна быть достаточной для последующей генерации программного кода, реализующего проект соответствующей системы.

Количество типов диаграмм для конкретной модели приложения строго не фиксировано. Для простых приложений нет необходимости строить все без исключения типы диаграмм. Некоторые из них могут просто отсутствовать в проекте системы, и это не будет считаться ошибкой разработчика. Например, модель системы может не содержать диаграмму развертывания для приложения, выполняемого локально на компьютере пользователя. Важно понимать, что перечень диаграмм зависит от специфики конкретного проекта системы.

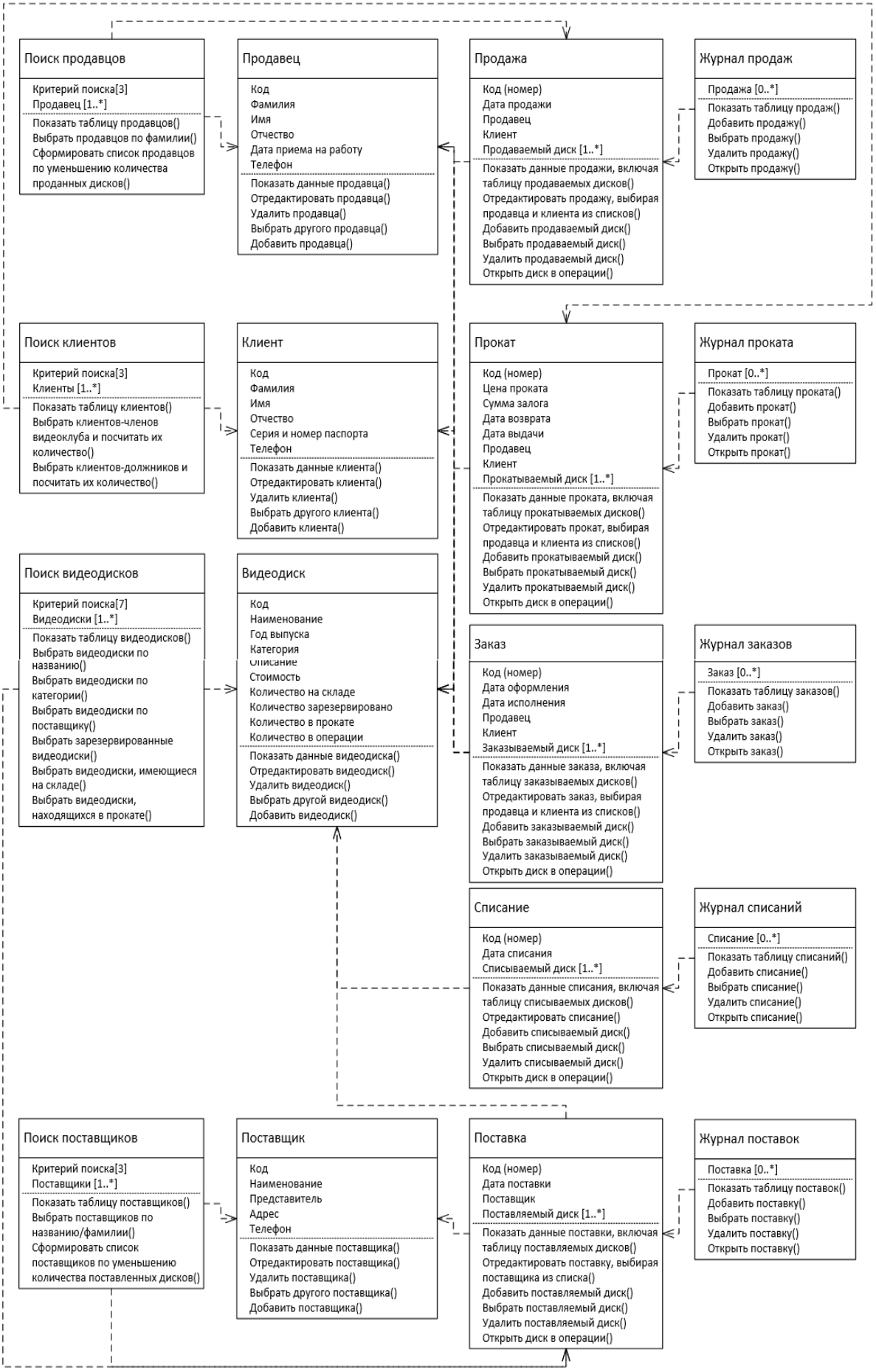
Любая модель системы должна содержать только те элементы, которые определены в нотации языка UML. Имеется в виду требование начинать разработку проекта, используя только те конструкции, которые уже определены в метамодели UML. Как показывает практика, этих конструкций вполне достаточно для представления большинства типовых проектов программных систем. И только при отсутствии необходимых базовых элементов языка UML следует использовать механизмы их расширения для адекватного представления конкретной модели системы. Не допускается переопределение семантики тех элементов, которые отнесены к базовой нотации метамодели языка UML.

Раздаточный материал № 32 (самостоятельно)

Для идентификации актеров в процессе проектирования системы могут быть рекомендованы вопросы, ответы на которые должны помочь разработчикам на начальных этапах выполнения проекта.

1. Какие организации или лица будут использовать проектируемую систему?
2. Кто будет получать пользу от использования системы?
3. Кто будет использовать информацию от системы?
4. Будет ли использовать система внешние ресурсы?
5. Может ли один пользователь играть несколько ролей при взаимодействии с системой?
6. Могут ли различные пользователи играть одну роль при взаимодействии с системой?
7. Будет ли система взаимодействовать с законодательными, исполнительными, налоговыми или другими органами?

Раздаточный материал № 33



Раздаточный материал № 34 (справочно)

В качестве примера - следующие варианты задания кратности атрибутов:

[0. .1] — означает, что кратность атрибута может принимать значение 0 или 1. При этом 0 означает отсутствие данного атрибута у отдельных объектов рассматриваемого класса.

[0. .*] — означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа — [*].

[1..*] — означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.

[1. .5] — означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.

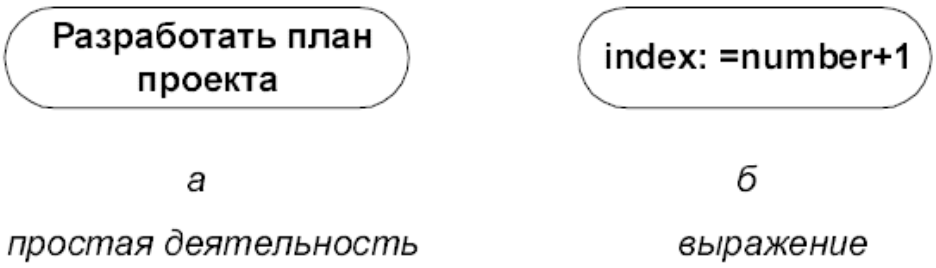
[1. .3,5,7] — означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7.

[1. .3,7. .10] — означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.

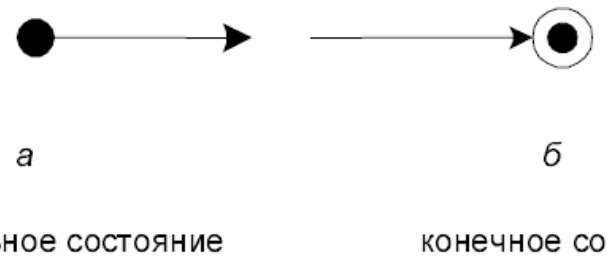
[1. .3,7.. *] — означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное [1.. 1], т. е. в точности 1.

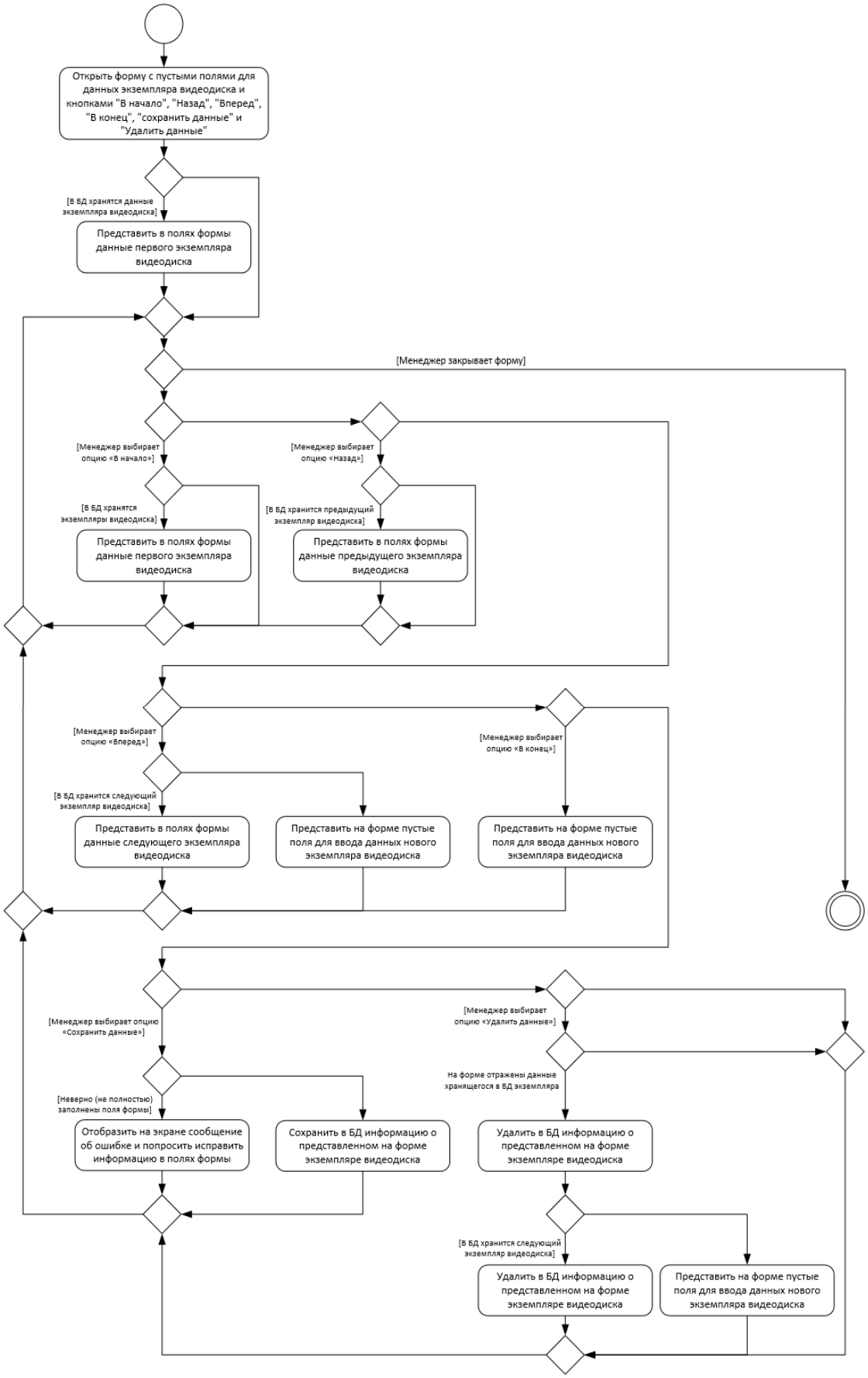
Раздаточный материал № 35



Раздаточный материал № 36



Раздаточный материал № 37



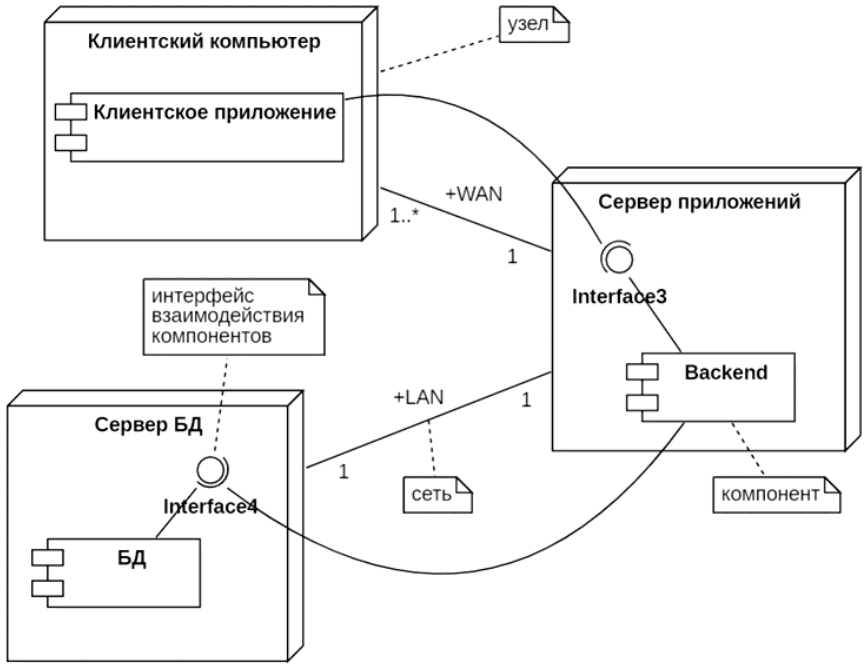
Раздаточный материал № 38

Раздаточный материал № 39 (справочно)

<table>
<tr><th rowspan="3">Компонент и критерий оценки</th><th colspan="3">Архитектура</th></tr>
<tr><th rowspan="2">Файл-сервер</th><th colspan="2">Клиент-сервер</th></tr>
<tr><th>Двухзвенная</th><th>Трехзвенная</th></tr>
<tr><td>Слой представления</td><td>На клиенте</td><td>На клиенте</td><td>На клиенте</td></tr>
<tr><td>Слой бизнес-логики</td><td>На клиенте</td><td>Частично на клиенте, частично на сервере</td><td>На сервере приложений</td></tr>
<tr><td>Слой доступа к данным</td><td>Частично на клиенте, частично на сервере</td><td>На сервере</td><td>На сервере данных</td></tr>
<tr><td>Достоинства</td><td>Низкая стоимость и высокая скорость разработки</td><td>Гарантия целостности данных</td><td>• Тонкий клиент
• минимальная передача данных между клиентом и сервером по сети (только аргументы функций и результаты вычислений)
• высокая масштабируемость
• небольшой трафик между серверами
• снижение нагрузки на сервер данных
• простота расширения функциональных возможностей и обновления</td></tr>
<tr><td rowspan="3">Недостатки</td><td rowspan="2">• Низкая производительность
• Низкая надежность
• Низкая масштабируемость и расширяемость</td><td>• сложность изменения бизнес-логики
• слабая защита данных от взлома</td><td></td></tr>
<tr><td colspan="2">• сложность администрирования и разработки
• высокие расходы на администрирование и разработку серверной части</td></tr>
<tr><td colspan="2">• высокие требования к вычислительной мощности клиента (ЦП, ОЗУ, диск)
• большой объем данных, передаваемых от клиента к серверу
• высокие требования к пропускной способности сети и клиентам</td><td></td></tr>
</table>

Раздаточный материал № 40

Диаграмма развертывания UML пример для трехзвенной монолитной архитектуры ИС



Раздаточный материал № 41

Раздаточный материал № 42

Раздаточный материал № 43

Сквозное тестирование vs системное тестирование

Сквозное тестирование	Системное тестирование
Проверяет программную систему, а также взаимосвязанные подсистемы.	Проверяет только программную систему в соответствии со спецификациями требований.
Проверяет весь сквозной поток процессов.	Проверяет функциональные возможности и функции системы.
Для тестирования рассматриваются все интерфейсы и серверные системы.	Рассматриваются функциональное и нефункциональное тестирование
Выполняется после завершения тестирования системы.	Выполняется после интеграционного тестирования.
Сквозное тестирование включает в себя проверку внешних интерфейсов, которую сложно автоматизировать. Следовательно, ручное тестирование предпочтительнее.	Для тестирования системы можно выполнять как ручное, так и автоматизированное тестирование.

Раздаточный материал № 44

- Не запускался. Тест-кейс создали, но тестирование по нему не проводили.
- Пройден успешно. Ожидаемые и фактические результаты работы ПО совпадают.
- Провален. Обнаружили дефект: ожидаемый результат минимум по одному шагу тест-кейса не совпадает с фактическим.
- Пропущен. Тест-кейс отменили. Например, потому что изменили требования к ПО.

Раздаточный материал № 45

✓ Уникальный идентификатор — некое уникальное значение. По нему на тест-кейс ссылаются из других документов или тест-кейсов. Бывает буквенным, числовым, буквенно-числовым. Чаще всего применяют простую сквозную нумерацию.

✓ Краткое описание — лаконичное описание сути тест-кейса. Может содержать ссылку на требование к ПО.

✓ Входные данные — сведения о первоначальном состоянии системы, которое важно для тест-кейса. А еще значения для ввода или передачи ПО.

✓ Шаги — полная последовательность действий. Ее выполняют, чтобы провести описываемую тест-кейсом проверку.

✓ Ожидаемый результат — описание планируемого поведения или результата ПО. Может базироваться на требовании к программному обеспечению, общей логике работы.

✓ Фактический результат — описание итогового поведения или результата ПО. Если они совпадают, это указывают. Когда не совпадают, подробно описывают расхождения. Пометка «не совпадает», «отличается» — это грубая ошибка.

✓ Статус — текущее состояние тест-кейса.

Раздаточный материал № 46

Шаблон и пример тест-кейса

Идентификатор	Описание	Шаги	Входные данные	Ожидаемые результаты	Фактические результаты	Статус
TU01	Проверка входа пользователя с существующими логином и паролем	Откройте сайт http://blahblahblah.ru ↓ Введите логин ↓ Введите пароль ↓ Нажмите кнопку «Войти»	Логин = user99 Пароль = pass99	Пользователь должен попасть на главную страницу	Как ожидали	Пройден успешно
TU02	Проверка входа пользователя с несуществующими логином и паролем	Откройте сайт http://blahblahblah.ru ↓ Введите логин ↓ Введите пароль ↓ Нажмите кнопку «Войти»	Логин = user99 Пароль = badlass99	Пользователь должен остаться на странице логина. Появится сообщение «Неверные логин или пароль»	Как ожидали	Пройден успешно

Раздаточный материал № 47 (самостоятельно)

Правила составления тест-кейса

☞ Создавайте простые тест-кейсы. То есть лаконичные и понятные не только вам. Используйте повелительное наклонение, например: «перейдите на домашнюю страницу», «введите данные», «нажмите здесь». Шаги должны быть четкие, без лишних деталей. Так проще понять шаги теста и ускорить работу.

☞ Учитывайте интересы конечного пользователя. Конечная цель любого программного проекта — простое и понятное приложение, отвечающее запросу клиентов. Тестировщик создает тест-кейсы с учетом мнения конечного пользователя.

☞ Избегайте повторов. Если тест-кейс нужен, чтобы выполнить другой тест-кейс, оставьте ссылку по идентификатору в столбце предварительного условия.

☞ Не предполагайте. Не додумывайте функциональность и возможности ПО. Строго придерживайтесь спецификации.

☞ Пишите тестовые примеры. Они должны покрывать все требования к ПО из спецификации. Используйте чек-листы и автоматизированные средства учета покрытия тестами. Это гарантия того, что ни одна функция или условие не останутся непроверенными.

☞ Задавайте идентификатор тест-кейса. Так, чтобы его было легко идентифицировать. Например, когда отслеживают ошибки или определяют требования к ПО на более позднем этапе.

☞ Внедряйте методы тестирования. Эти техники помогают спланировать несколько тест-кейсов и находить ошибки:

анализ граничных значений — проверяйте верхние и нижние границы для допустимого диапазона значений;

эквивалентное разделение — разбивайте диапазон всевозможных тест-кейсов на равные части/группы с одинаковым поведением;

техника перехода состояний — создавайте тест-кейсы, которые покроют поведение ПО при переходе из одного состояния в другое.

☞ Внедряйте самоочистку. Тест-кейс должен возвращать среду в предтестовое состояние. Особенно это касается тестирования конфигураций.

☞ Создавайте повторяемые и самостоятельные текст-кейсы. Они должны всегда генерировать одинаковые результаты: независимо от того, кто их тестирует.

☞ Проводите экспертную оценку. Отправляйте текст-кейсы на проверку коллегам. Они могут обнаружить ошибки в дизайне тест-кейса, которые вы пропустили.