Содержание

Аннотация

В данной работе наша команда занималась разработкой прототипа коллаборативной доски для проектирования ИТ-систем и ПО. В свою очередь проектирование - важная и неотъемлемая ступень создания программного приложения, цель которой состоит в выявлении требований, структуры проекта и архитектурных решений конкретных частей. Эту часть разработки можно представить как создание плана действий, что помогает избежать глобальных ошибок в будущем времени и грамотно оценить масштабы и ресурсы для реализации проекта. В данный момент на рынке существует множество приложений для проектирования и управления проектом, однако большинство из них заточены под конкретные подходы и имеют ряд недостатков, поэтому для построения требований к нашему проекту понадобилось подробное изучение предметной области и современных решений с последующим анализом преимуществ и недостатков каждого из них. Все полученные данные сыграли роль не только в определении функционала нашего приложения, но также и в организации структуры программной части приложения. В итоге в основу проекта легло решение создания модульной архитектуры, которая с легкостью бы предоставляла возможность к расширению опций, оптимизации отдельных модулей приложения независимо от остального проекта. На данный момент программная часть проекта - прототип желаемой доски с различными, доступными для пользователя объектами: текст, разноцветные стикеры, карандаш для произвольного рисования, соединитель в виде стрелок между объектами, группа объектов, позволяющая синхронно передвигать и менять атрибуты сразу нескольким объектам. Все эти объекты можно представить в виде отдельной таблицы для наглядного просмотра объектов доски и также удобного изменения свойств. Приложение предоставляет возможность синхронизации построенных схем и решений между несколькими пользователями, что обеспечивает совместную работу в проектировании интерфейса и архитектуры систем. Таким образом на выходе разработан прототип коллаборативной доски для дальнейшего совершенствования с использованием уже более современных технологий и инструментов.

Список ключевых слов

- **Проектирование**
- **State Machine**
- **Модульность**
- **Доска**
- **Методологии проектирования**
- **Совместная работа**

1. Введение

“*Проектирование* – процесс составления описания, необходимого для создания в заданных условиях еще не существующего объекта, на основе первичного описания этого объекта и/или алгоритма его функционирования или алгоритма процесса, преобразованием (в ряде случаев неоднократным) первичного описания, оптимизацией заданных характеристик объекта и алгоритма его функционирования или алгоритма процесса устранением некорректности первичного описания и последовательным представлением (при необходимости) описаний на различных языка.” (ГОСТ 22487-77)

V-модель - модель, которая представляет собой метод описания процесса разработки программного обеспечения от этапов проектирования до полного формирования конечного продукта и направлена на упрощение этого процесса. В левой ветке происходит постепенная декомпозиция системы сверху-вниз, а в правой - пошаговая сборка и тестирование все более крупных частей вплоть до системы в целом.

Рассмотрим процесс проектирования через левую ветку V-модели.

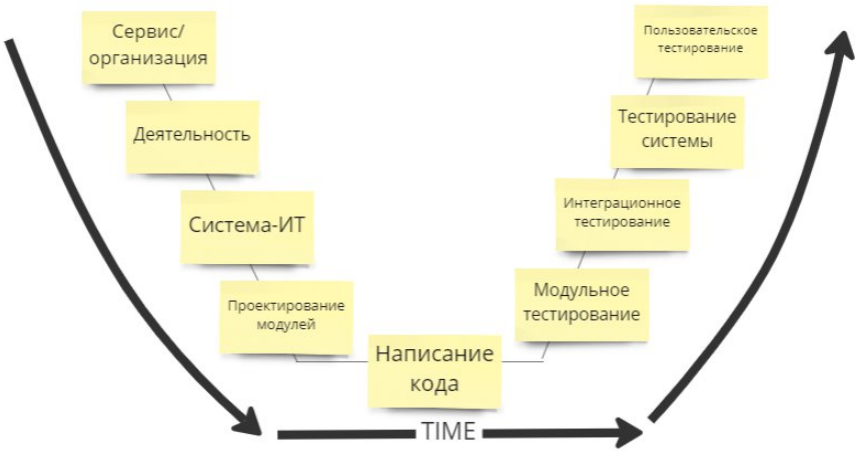


Рисунок 1.1 – V-модель

Описание левой ветки, которая отвечает за проектирование:

1. Сервис/организация. На первом этапе следует провести анализ требований к системе. Этот этап направлен на выявление свойств и функций идеальной системы.
2. Деятельность. Второй этап отвечает за анализ уже выставленных требований и оценивание методов и возможностей их реализации.
3. Система информационных технологий. Этот этап можно назвать высокоуровневым проектированием. Создается список модулей с их функциональностью, техническими деталями и зависимостями.

4. Проектирование модулей. Последний этап можно назвать низкоуровневым проектированием, где создается описание конкретных модулей, которое в итоге можно рассматривать как готовое техническое задание для программиста или командам, которым поручат этот модуль.

Предметной областью для нашего инструмента является проектирование. Рассмотрим различные подходы проектирования, для которых используются конкретные приложения автоматизации этого процесса. Во внимание возьмем 4 направления: SADT, практики проектирования Agile, DDD, OOAD.

Актуальность и новизна:

На протяжении предыдущих десятилетий инструменты и подходы проектирования ИТ-систем и ПО постоянно развивались. Был период усложнения инструментов: в них было много различных функций и улучшений. Однако к единому подходу отрасль информационных технологий так и не пришла. На данный момент гибкие практики проектирования и инструменты испытывают откат к доске, фломастеру и стикерами, при этом утрачиваются многие функции, полезные для работы с требованиями и проектными решениями. По этой причине поиски идеального инструмента и метода проектирования следует продолжать.

Одним из представителей доски для организации работы распределенной команды является Miro - его можно считать образцом для подражания. Однако после объявления ухода Miro из России, исчезает возможность использования даже его. Отечественных инструментов проектирования даже предыдущих поколений и подходов в России или мало или нет вообще. Примерами таких являются Business Studio и Elma, однако они узко специализированы и не решают всех проблем моделирования.

Цели:

1. Исследование подхода к замене дистанционной доски для совместной работы
2. Исследование возможности получения системы проектирования следующего поколения

Задачи:

1. Исследовать инструменты проектирования
2. Исследовать возможность получения системы проектирования следующего поколения
3. Получить прототип доски, пригодный для дальнейших экспериментов

4. Определить требования к "инструменту проектирования мечты".

Результаты:

1. Проведено исследование и анализ различных подходов к проектированию и используемых для них инструментов, результаты которого описаны ниже.
2. Получен прототип приложения коллаборативной доски с использованием модульной архитектуры. Каждый модуль является либо полноценным объектом, либо дополнительным функционалом и является независимым от остальных. С помощью созданного инструмента можно визуально изобразить последовательный план работы над задачами, разбить проектирование на графические секции, углубить понимание проекта у всей команды. Помимо этого, есть отображение всех объектов не только на доске, но и в виде таблицы для удобного просмотра созданных объектов и изменения их свойств.

Ход работы с указанием участия членов команды:

1. Выбор языка программирования в пользу python с активным использованием графической библиотеки tkinter. Изучение содержания библиотеки.
2. Изучение различных подходов проектирования: SADT, Agile, DDD, OOAD и оценка существующих ПО на поддержку этих подходов с целью выявления нужных нам идей для создания своего проекта. Каждый человек занимался своим подходом:
    a. Антон - OOAD.
    b. Сардор - Agile.
    c. Амина - SADT.
    d. Лилиана - DDD.
3. Создание ядра прототипа в виде проектирования и реализации структуры с примером подсоединения одного модуля - текста. Происходило в три этапа:
    a. Индивидуальная работа. Каждый участник команды получил задание: создание своего варианта ядра, чтобы в дальнейшем объединить все независимые идеи в один проект. В процессе происходили встречи с руководителем и обсуждались некоторые идеи.
    b. Парная работа. Этот этап предназначен для практики парного кодирования, и также слияния четырех версий в две.
    c. Выбор наиболее удачного решения. За основу проекта было принято решения взять начальную архитектуру Антона с некоторыми доработками в дальнейшем, принятыми в процессе обсуждений.

4. Разбиение отдельных модулей на людей:
    a. Антон - группа, обновление ядра соответствующими идеями, слияние версий.
    b. Амина - модуль стикера и текста.
    c. Сардор - меню свойств для изменения объектов, соединитель, таблица элементов.
    d. Лилиана - основное меню и модуль рисования.
5. Создание документации для итогового отчета - Амина и Лилиана.

2. Обзор литературы

Предметной областью для нашего инструмента является проектирование. Рассмотрим различные подходы проектирования, для которых используются конкретные приложения автоматизации этого процесса. Во внимание возьмем 4 подхода: SADT, Agile, DDD, OOAD.

Object Oriented Analysis and Design

UML

UML — унифицированный язык моделирования — это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования [2]. Используется для визуализации, спецификации, конструирования и документирования программных систем.

Плюсы:

- Возможность посмотреть на задачу с разных точек зрения
- Другим разработчикам легче понять суть задачи и способ ее реализации
- Диаграммы просты для чтения после быстрого ознакомления с их синтаксисом

Минусы:

- Трата времени на моделирование
- Необходимость знания различных диаграмм и их нотаций

Диаграмма вариантов использования — диаграмма, показывающая возможные варианты взаимодействия различных видов пользователей с системой. Участники – множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями. Участником может быть человек, другая система, подсистема или класс. Прецедент — описание множества последовательных событий, выполняемых системой, которые приводят к наблюдаемому участником результату [8]. Прецедент представляет поведение сущности, описывая взаимодействие между участниками и системой. Прецедент не показывает, каким образом достигается некоторый результат, он лишь описывает сам результат.

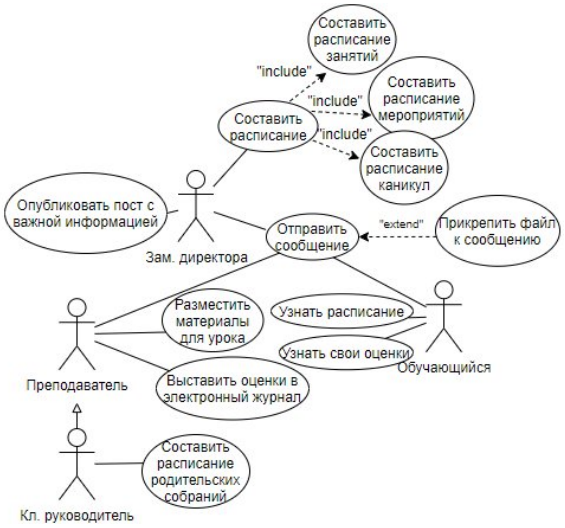


Рисунок 2.1 – Диаграмма вариантов использования для школы [4]

Диаграмма последовательности — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта и взаимодействие действующих лиц информационной системы в рамках сценария использования. Диаграммы последовательностей используются для уточнения диаграмм вариантов использования, более детального описания логики сценариев использования.

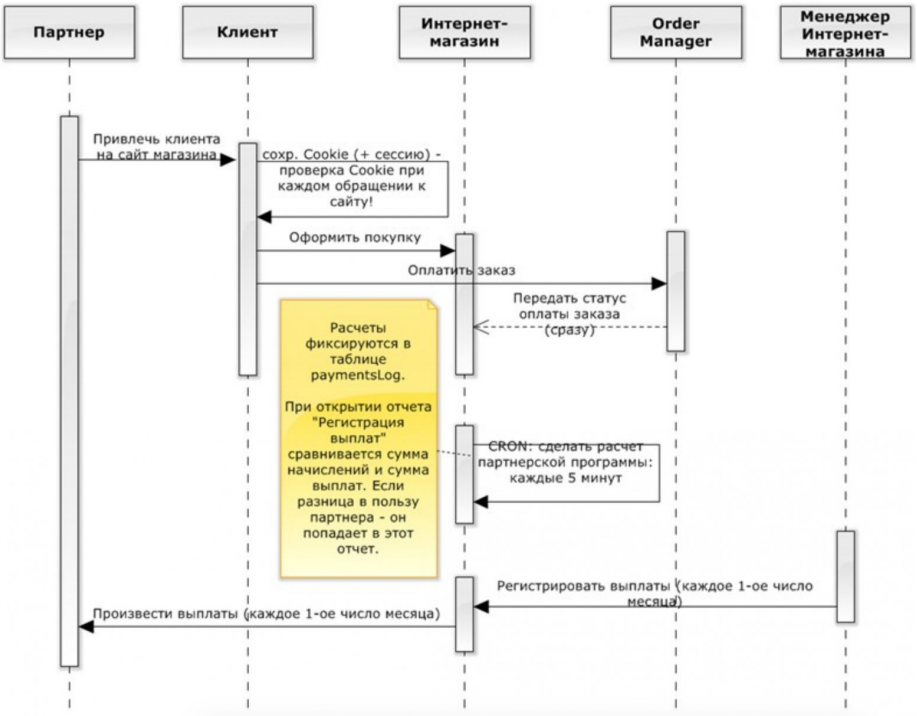


Рисунок 2.2 – Диаграмма последовательностей для интернет-магазина [21]

Диаграмма классов — наиболее распространенная разновидность диаграмм UML и фундаментальная база любого объектно-ориентированного решения. Отображает классы внутри системы, а также атрибуты, операции и отношения между классами [22]. Диаграммы классов применяются при схематизации крупных систем, так как позволяют объединять классы в группы.

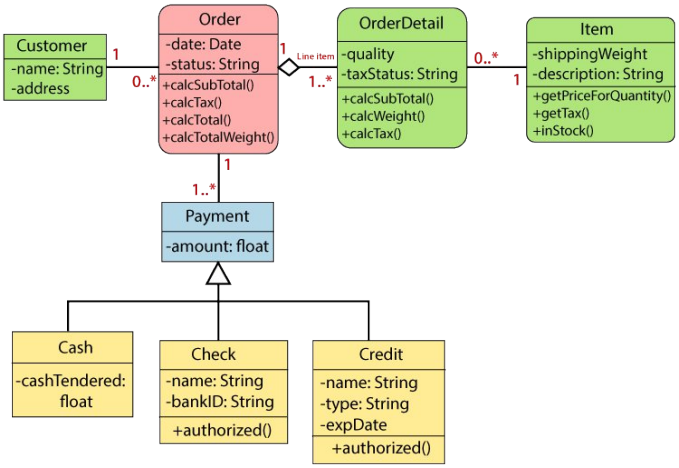


Рисунок 2.3 – Диаграмма классов для интернет-магазина [9]

Диаграмма состояний — диаграмма, отражающая состояния и переходы между ними. На диаграмме отображаются события, которые влияют на состояние системы.

События могут иметь параметры, несущие дополнительную информацию. По событию также могут выполняться определенные действия во время смены состояния (например, изменение значения переменной). На диаграмме можно указывать условные выражения. От результата их вычисления будет зависеть то, в какое состояние будет следующий переход.

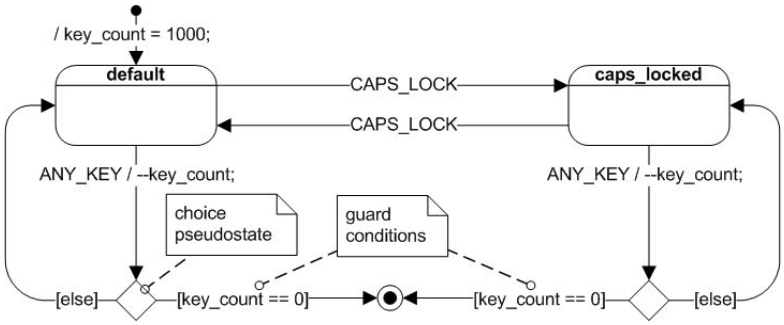


Рисунок 2.4 – Диаграмма состояний [23]

Диаграмма деятельности, как и диаграмма состояний, отражает динамические аспекты поведения системы. По существу, эта диаграмма представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой [7]. Активности на диаграмме распределены по дорожкам, каждая из которых соответствует поведению одного из объектов. Благодаря этому легко определить, каким из объектов выполняется каждая из активностей.

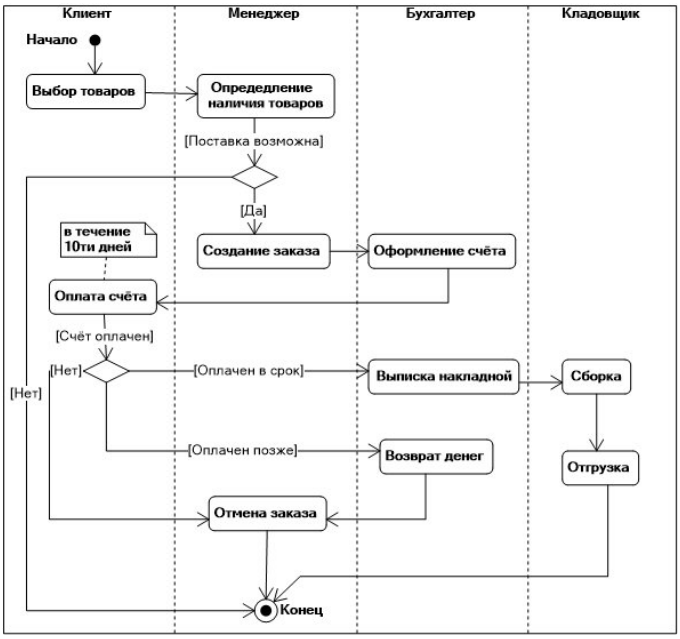


Рисунок 2.5 – Диаграмма деятельности для интернет-магазина [7]

Диаграмма компонентов используется, чтобы показать разбиение программной системы на структурные компоненты и связи между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и пр.

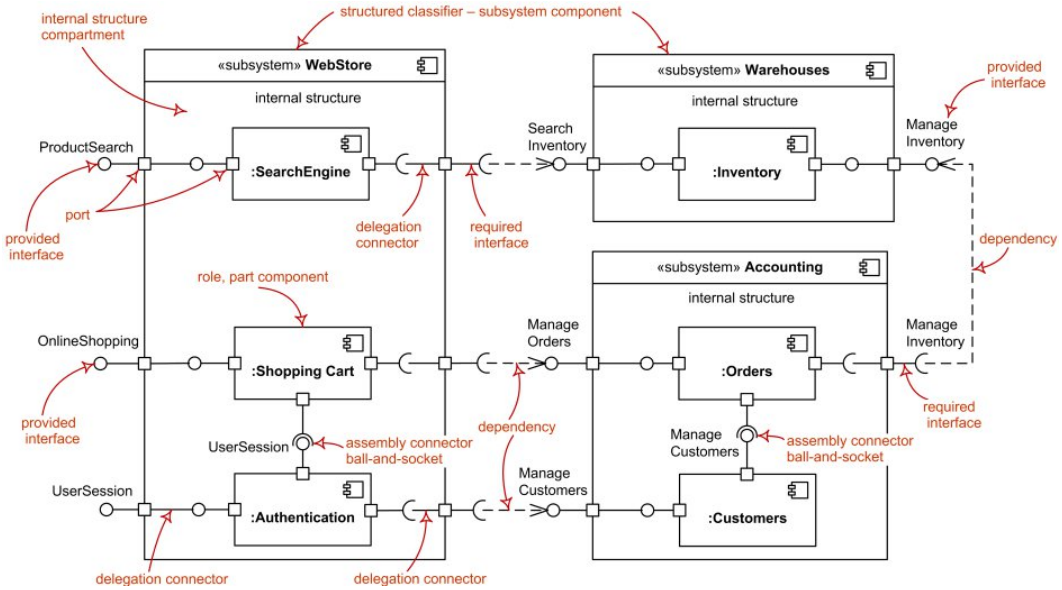


Рисунок 2.6 – Диаграмма компонентов для интернет-магазина [24]

Диаграмма развертывания используется для графического представления инфраструктуры, на которую будет развернуто приложение. Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д. Всё это может быть отражено на диаграмме развертывания.

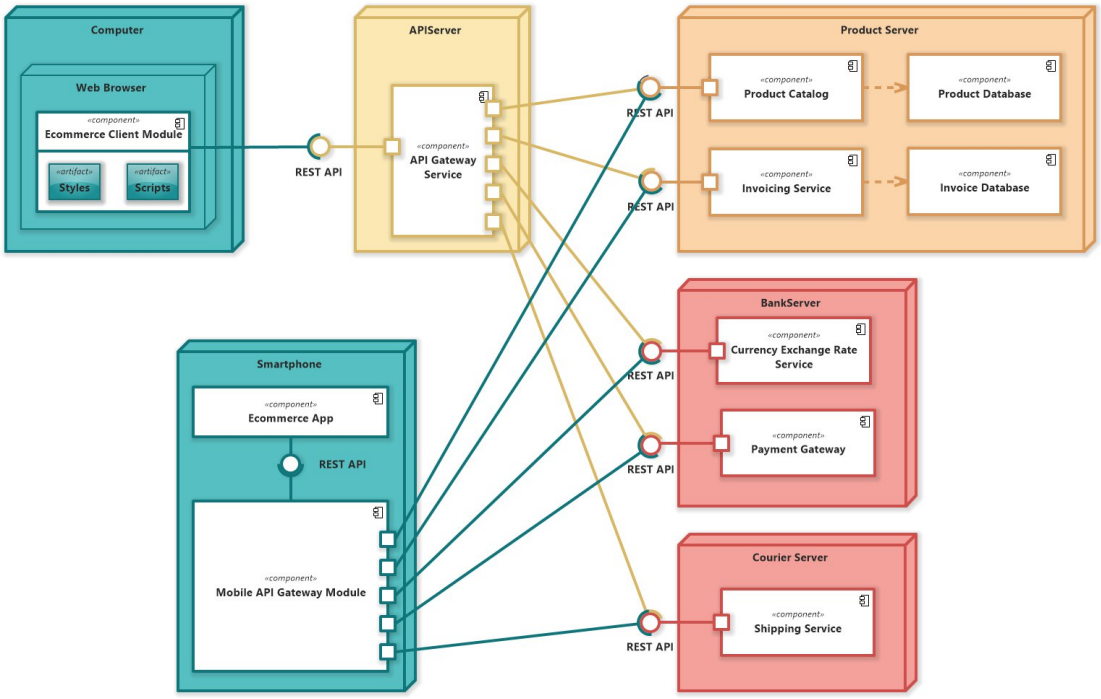


Рисунок 2.7 – Диаграмма развертывания для интернет-магазина [10]

BPMN

Основная цель BPMN – предоставить условные обозначения, которые будут понятны всем участникам бизнеса, от аналитиков, создающих черновые варианты процессов, до разработчиков, реализующих эти процессы, и до тех, кто будет управлять этими процессами и анализировать их. BPMN является "стандартизированным мостиком" между дизайном бизнес-процесса и его реализацией [1].

Процессы, описанные с помощью BPMN, условно можно поделить на исполняемые и неисполняемые. Отличие в том, что исполняемые процессы могут быть исполнены с помощью ПО (Bizagi, Comundo). Неисполняемые процессы не требуют той строгости описания, которая нужна для выполнения исполняемых.

Плюсы от использования BPMN:

- Более легкое общение и сотрудничество для достижения цели
- Простое визуальное представление этапов
- Выявление проблем в процессах, которые могут нуждаться в решении
- Понимание потенциальных областей для улучшения

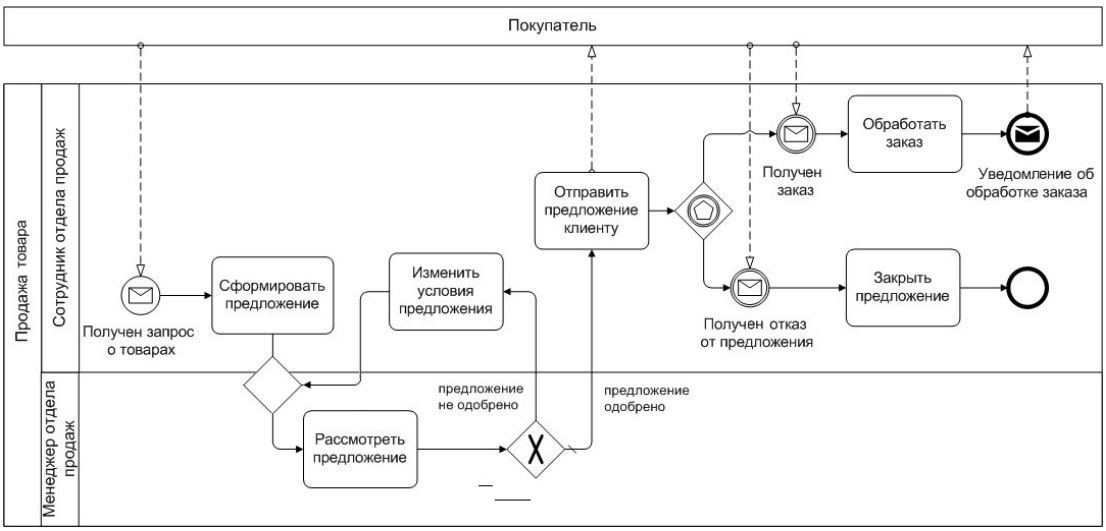


Рисунок 2.8 – Диаграмма в BPMN для интернет-магазина [25]

Краткое описание основных элементов BPMN

Events – какие-либо происшествия в мире. События инициируют действия или являются их результатами. Согласно расположению в процессе, события могут быть классифицированы на начальные, промежуточные и завершающие [5].

Activities – задачи, которые необходимо выполнить на определенном этапе бизнес-процесса. Действия могут быть элементарными, т.е. неделимыми на некоторые более простые действия, и не элементарными, т. е. такими, которые при детализации распадаются на последовательность определенных более простых действий [5].

Gateway – управляющий узел, который появляется при условном разветвлении бизнес-процесса. Шлюзы нужны в тех случаях, когда процедура зависит от определенных факторов. Например, при работе с покупателями шлюз появляется на этапе, когда клиент принимает решение о покупке — «да или нет». При положительном решении необходимо совершить покупку, при отрицательном - выяснить возможные причины отказа, работать с "отказом" и т.д. [5]

Pool – это объект, описывающий один процесс на диаграмме. На одной диаграмме может быть несколько пулов. Пул может быть расширен для просмотра деталей. Пул также может содержать так называемые «дорожки». Они нужны для того, чтобы указать участников процессов, которые скрыты в пуле [5].

Data – это элемент, указывающий, какие данные и документы необходимы для начала действия или каковы результаты завершенного действия [5].

Artefact – под артефактами в BPMN понимаются объекты, не являющиеся действиями и не имеющие прямого отношения к действиям. Это могут быть любые документы, данные, информация, не влияющие непосредственно на выполнение процесса [5].

Инструменты, подходящие для использования OOAD

- **SparX Enterprise Architect**
  - Поддержка UML и BPMN диаграмм
  - Матрица трассировки
  - Дерево связей для навигации по связям
  - Генерация кода и документации по диаграмме классов
  - Отсутствие возможности совместной работы
  - Сложность интерфейса программы
- **PlantUML**
  - Поддержка UML-диаграмм [6]
  - Хорошо подходит для контроля изменений, т.к. диаграммы генерируются из текстового формата
  - Есть дополнения для различных сред разработки и редакторов кода (VSCode, Atom, IntellijIDEA) [3]
  - С готовой диаграммой можно работать лишь как с изображением
  - Нет поддержки совместной работы
  - Необходимость знания языка для описаний диаграмм
- **Miro**
  - В платных планах есть поддержка небольшого кол-ва элементов, которые используются в диаграммах, но их можно эмулировать самому
  - Нет возможности посмотреть связи и карточки в виде списка
  - Возможность совместной работы
- **Camunda**
  - Поддержка BPMN
  - Средство для поиска ошибок и узких мест в бизнес-процессах
  - Веб-приложение, в котором исполнители выполняют задачи, поставленные на них бизнес-процессом
  - Веб-приложение для просмотра состояния процессов
- **ELMA 365**
  - Моделирование и исполнение бизнес-процессов (Поддержка BPMN)
  - Графический конструктор процессов и интерфейсов
  - Инструменты по управлению документооборотом
  - API для использования во внешних программных продуктах

- Корпоративный мессенджер для совместной работы: лента, каналы и чаты, обмен файлами и постановка задач

Agile

В разработке программного обеспечения включает в себя:

- выявления требований и улучшение продукта за счет совместных усилий разработчиков, бизнес-технологов с их конечными пользователями
- адаптивное планирование
- ранняя эксплуатация
- постоянное совершенствование
- гибкое реагирование на изменения требований.

Плюсы от использования AGILE:

- Из-за того, что на каждой итерации проектируется минимальный значимый продукт, то команда может с легкостью получить обратную связь от пользователей и, если нужно изменить будущие планы, это произойдет без потери значительных ресурсов.
- Прозрачность.
- Задействование всей команды, т.е. в каждой итерации найдется задача для всех (разработчики, тестировщики, дизайнеры).

Минусы от использования AGILE:

- Только маленькие команды могут использовать AGILE методологию.
- Отсутствие четкого плана разработки.
- Риск получить продукт низкого качества.

Impact mapping

Impact mapping - составление mind map`ов, которые позволяют визуализировать границы проекта и связывать бизнес-процессы с конкретными реализациями. [19]

Правила impact mapping - нужно ответить на 4 вопроса:

1. Зачем? Цель проекта, чего хотим достичь, сделав этот проект.
2. Кому? Действующие лица - кому мы делаем этот проект, кто будет пользоваться проектом.
3. Как? Как будет воздействовать наш проект действующим лицам и как они могут воздействовать на проект.
4. Что сделаем? Что нужно сделать, чтобы добиться такого проекта.

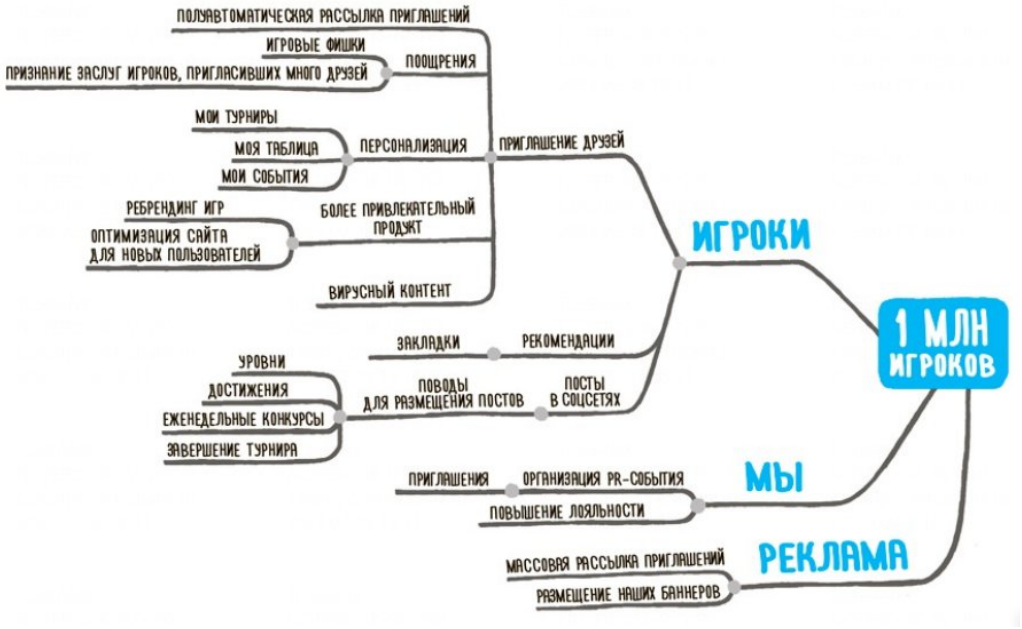


Рисунок 2.9 – Impact map для игры

На Рисунке 2.9 показан impact mapping онлайн-игры казино:

1. Цель - привлечь 1 млн онлайн-клиентов.
2. Действующие лица — это игроки, реклама и мы.
3. Как достичь цель - игроки могут приглашать друзей, мы можем организовать PR-события, а с помощью рекламы массово рассылать приглашения.
4. Что проект должен иметь чтобы лица могли так вести? Для того, чтобы игроки хотели приглашать своих друзей, нужно заинтересовать действующих игроков вирусным контентом или же поощрением за 1-го приглашенного друга

Плюсы от использования Impact Mapping:

- Легко дорабатывать.
- Понимание процессов между бизнесом и разработчиками.

Минусы от использования Impact Mapping:

- Короткое время жизни, нужно постоянно актуализировать.
- Нет четкого понимания сроков проекта.

Story mapping

User story - короткая формулировка сценария, описывающая что-то, что система должна делать для пользователя.

Story Mapping (пользовательские истории) — это метод, который позволяет дорабатывать или выявлять новые функционалы для усовершенствования продукта.[18]

Пользовательские истории выделяются в три этапа.

1. Дебют. Базовые сценарии пользователя, которые должен поддерживать проект.
2. Миттельшпиль. Дополняет и прорабатывает функционалы, которые есть в дебют и которые облегчают работу пользователя
3. Эндшпиль. Наводит блеск проекту, обогащает функционалом.

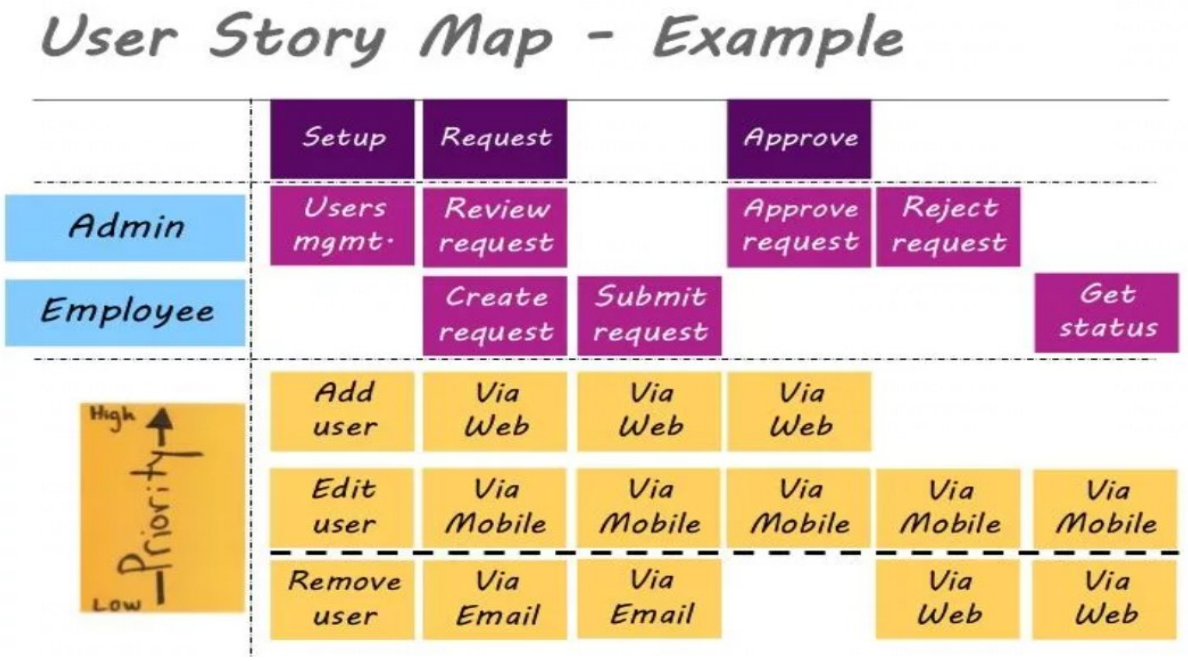


Рисунок 2.10 – User Story Map

На Рисунке 2.10 изображен пример User Story Map на основе обычного клиент-серверного приложения.

1. Голубые карточки — это типы пользователей.
2. Бордовые карточки — это базовое представление приложения.
3. Розовые карточки — это то, какие сценарии есть для конкретного пользователя с базового представления.
4. Желтые карточки — это функционалы, которые должно содержать приложение для поддержания сценария пользователя в приоритетном порядке.
5. Пунктирное разделение — это релизы приложения.

Плюсы от использования User story mapping:

- Прозрачность в смысле наглядности и приоритезации.
- Основной фокус на пользователя.

Минусы от использования User story mapping:

- Нет детальных требований к ресурсу. Повествование делается в общих чертах.

C4 (context-container-component-code)

C4 (context-container-component-code) - метод, который позволяет декомпозировать проект, отражая его с разных точек зрения.[20]

Уровни декомпозиции:

1. Диаграмма **контекста системы** обеспечивает отправную точку, показывая, как программная система по объему вписывается в окружающий мир.
2. Схема **контейнера** увеличивает масштаб программной системы, показывая высокоуровневые технические строительные блоки.
3. Диаграмма **компонентов** увеличивает масштаб отдельного контейнера, показывая компоненты внутри него.
4. Диаграмма **кода** (например, класса UML) может использоваться для увеличения масштаба отдельного компонента, показывая, как этот компонент реализован.

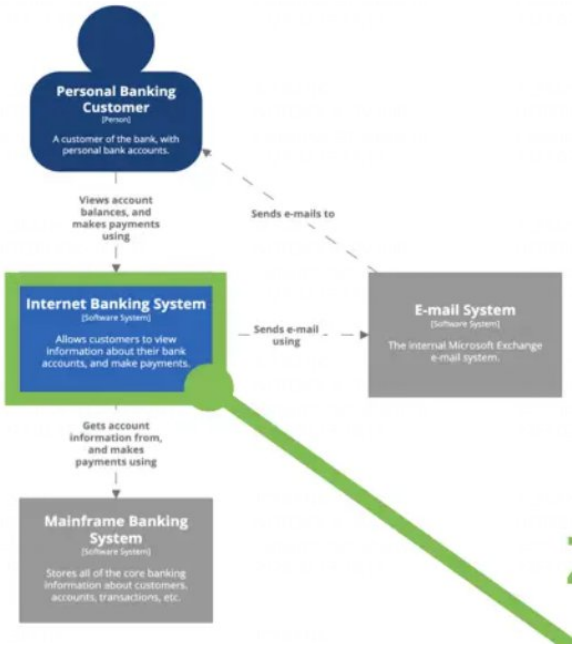


Рисунок 2.11 – Диаграмма контекста в C4

На Рисунке 2.11 показан самый верхний уровень детализации - **контекст системы.** Здесь Банковская система (зеленый прямоугольник) показана в контексты системы. Далее приблизим и контейнерную детализацию.

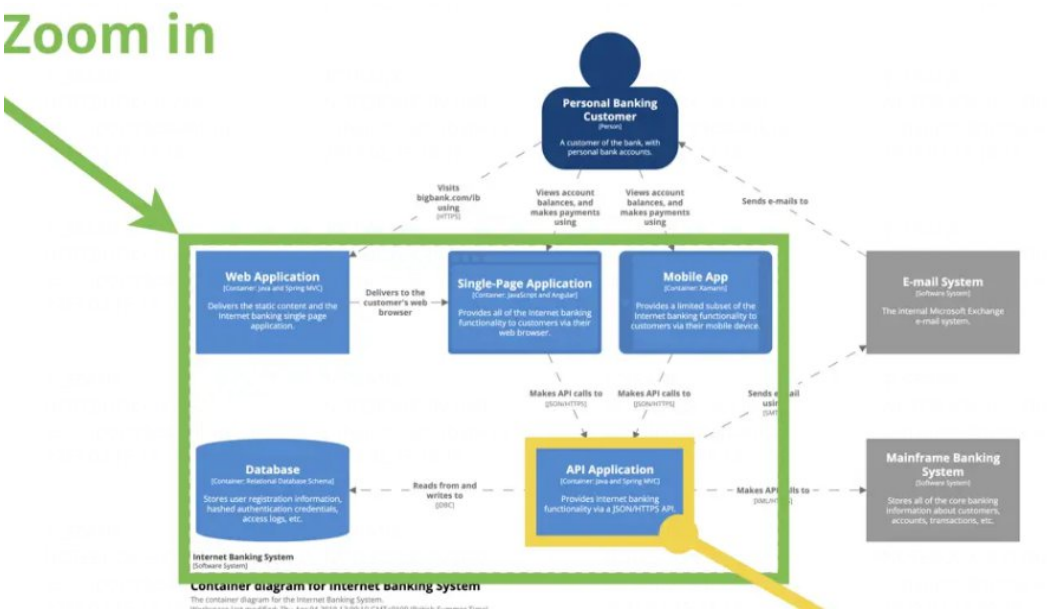


Рисунок 2.12 – Диаграмма контейнеров в C4

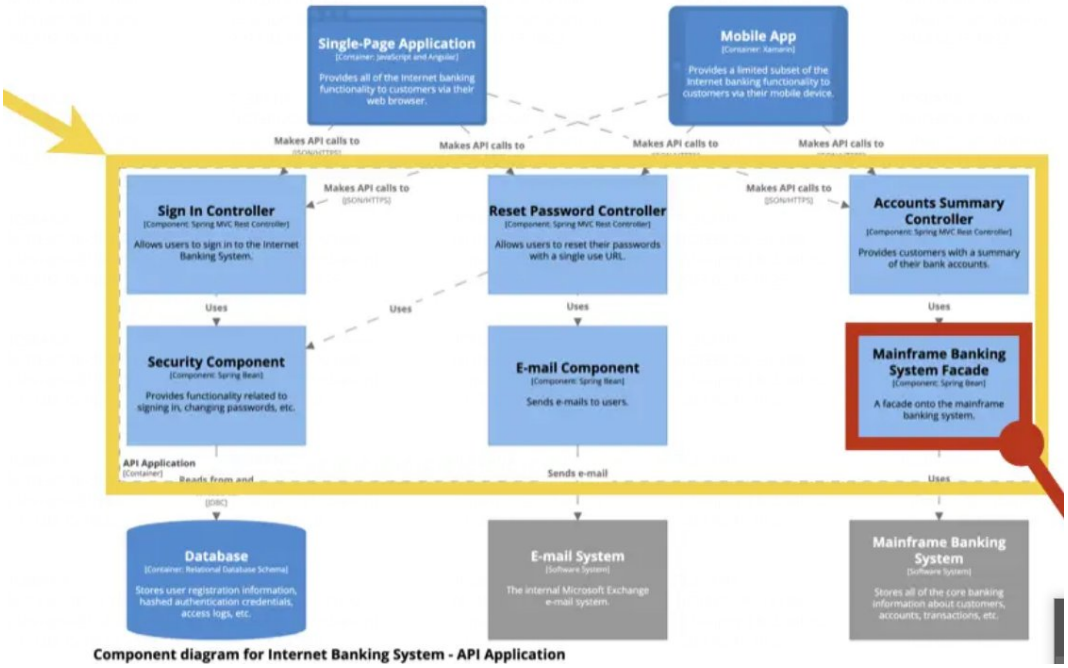


Рисунок 2.13 – Компонентная диаграмма контейнера API в C4

Как показано на Рисунке 2.12, банковскую система была разложена на контейнеры (синие фигуры внутри зеленого прямоугольника). Следующий уровень детализации — это компонентная детализация (см. Рисунок 2.13), для примера рассмотрим контейнер - API приложение (желтый квадрат).

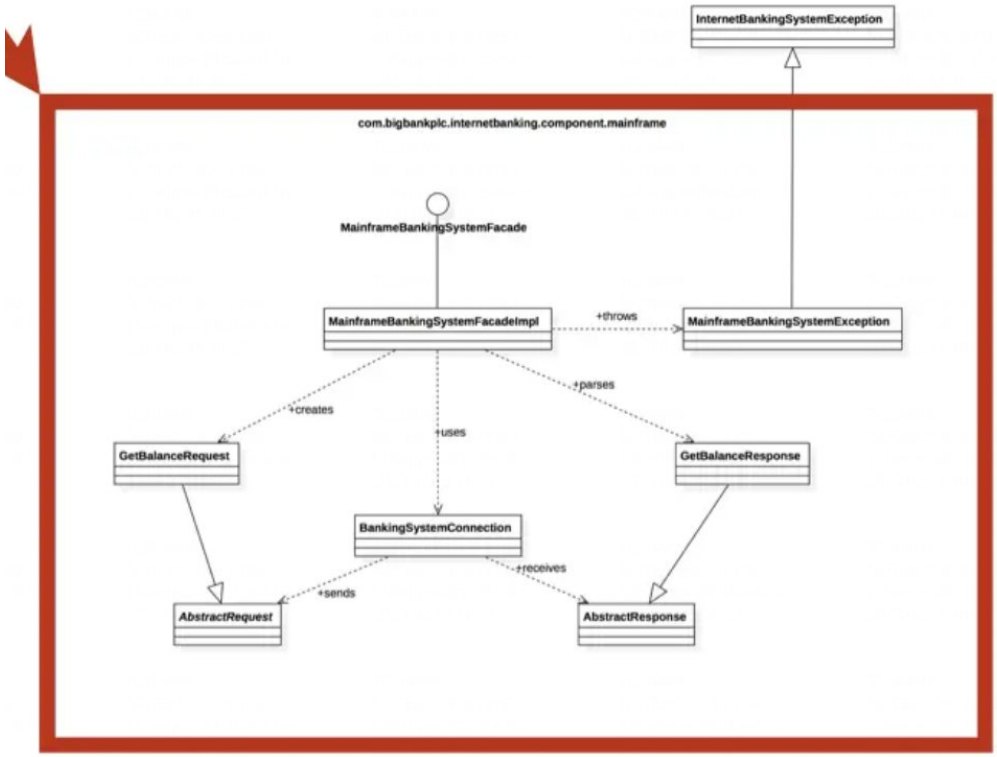


Рисунок 2.14 – Диаграмма кода в C4

Далее, как последний уровень детализации, рассмотрим компоненту “Главное меню приложения” и его диаграмму кода (см. Рисунок 2.14).

CJM (Customer Journey Mapping)

CJ (Customer Journey) — это путь, который проходит клиент: от возникновения потребности в товаре до момента покупки или превращения в фаната бренда.

CJM (Customer Journey Mapping) — это визуализация пути покупателя. Она отображает этапы, которые проходит клиент и какие эмоции при этом испытывает, точки взаимодействия с брендом и сложности, которые не позволяют ему достигать своих целей.

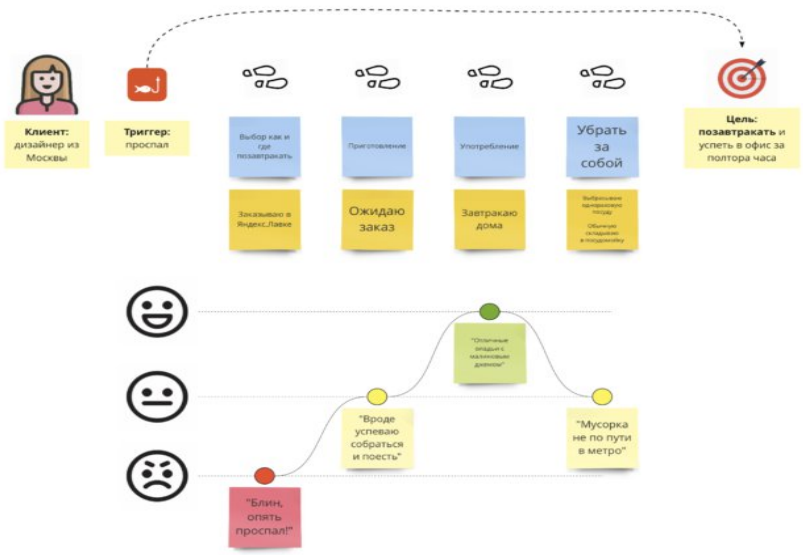


Рисунок 2.15 – CJM для проспавшего работника

Пример CJM:

1. Триггер - причина почему произошел такой сценарий
2. Голубые карточки показывает, что клиент собирается делать
3. Темно желтой карточки показывает - что клиент делает на конкретном этапе
4. В рядах со смайликами показывается то, что клиент чувствует на данном этапе.

Так благодаря CJM можно узнать точки-этапы - где меняются эмоции клиента и найти способы улучшения клиентского опыта.

Инструменты, подходящие для использования AGILE

- **SparX Enterprise Architect**
  - Управление бэклогом
  - Иерархические вложенные списки
  - Карточки
  - Настройка планов и задач
  - Канбан
  - User Story Mapping
- **MIRO**
  - Поддерживает все подходы AGILE
- **Notion**
  - Управление бэклогом
  - Настройка планов и задач
  - Классификация по карточкам
  - Карточки

- User Story Mapping

- **Trello**
  - Управление бэклогом
  - Настройка планов и задач
  - Карточки
  - Канбан
- **Google Sheets**
  - Настройка планов и задач
  - С трудом карточки в виде таблиц
- **Confluence**
  - Управление бэклогом
  - Настройка планов и задач
  - Карточки

SADT

IDEF0

IDEF0 - графическая нотация, предназначенная для формализации и описания бизнес-процессов. Модель, построенная в этой нотации, является некоторым толкованием определенной системы, что значит субъектом моделирования может быть только одна система. У каждой модели только одна цель и точка зрения, где целью является получение ответов на некоторую совокупность вопросов с заданной степенью точности, а точка зрения это отражает позицию, с которой описывается система [13].

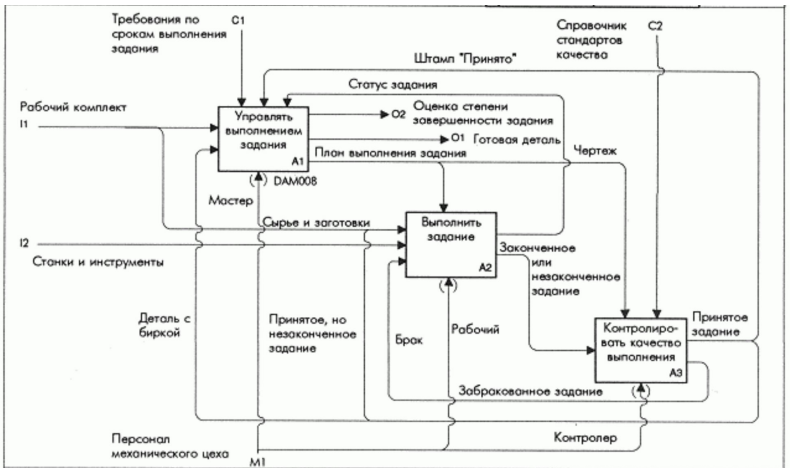


Рисунок 2.16 – Диаграмма модели в IDEF0 [13]

Диаграммы в модели состоят из блоков, описывающих определенное действие (функцию), и двунаправленных стрелок, связывающих блоки и отображающих их взаимодействие посредством объектов. Каждый блок можно детализировать в диаграмме-потомке, чем ниже

уровень - тем больше деталей о соответствующей функции системы. В контекстной диаграмме существует единственный блок, обозначающий границу системы: все, лежащее внутри него, является частью описываемой системы, а все, лежащее вне него, образует внешнюю среду системы. На диаграммах нижнего уровня 3-6 блоков:

Рисунок 2.17 – Блоки и стрелки в IDEF0 [13]

Виды стрелок (см. Рисунок 2.17):

- I - Input - Вход - объекты потребляемое и преобразуемое функцией.
- C - Control - Управление - информация, которая управляет действиями функций
- O - Output - Выход - результат функции, часто является преобразованием вводных объектов
- M - Mechanism - Механизм - объекты используемые, но не потребляемые функцией

IDEF1

IDEF1 - методология моделирования информационных потоков внутри системы, позволяющая отображать и анализировать их структуру и взаимосвязи [14].

Блоки в этой нотации представляют сущность, которая соответствует конкретному объекту (или группе) и описывает совокупность информации. У сущностей есть атрибуты, являющиеся их свойствами и признаками. Атрибут состоит из названия атрибута и его значения для этой сущности. Атрибуты с уникальными для каждой сущности значениями называются ключевыми атрибутами.

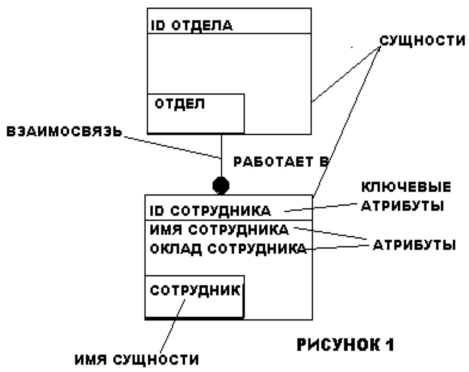


Рисунок 2.18 – Блоки и их атрибуты в IDEF1 [14]

IDEF1X

IDEF1X (IDEF1 extended) - методология позволяющая создавать семантические модели данных, которые могут служить для поддержки управления данными как ресурсами, интеграции информационных систем и построения компьютерных баз данных [15].

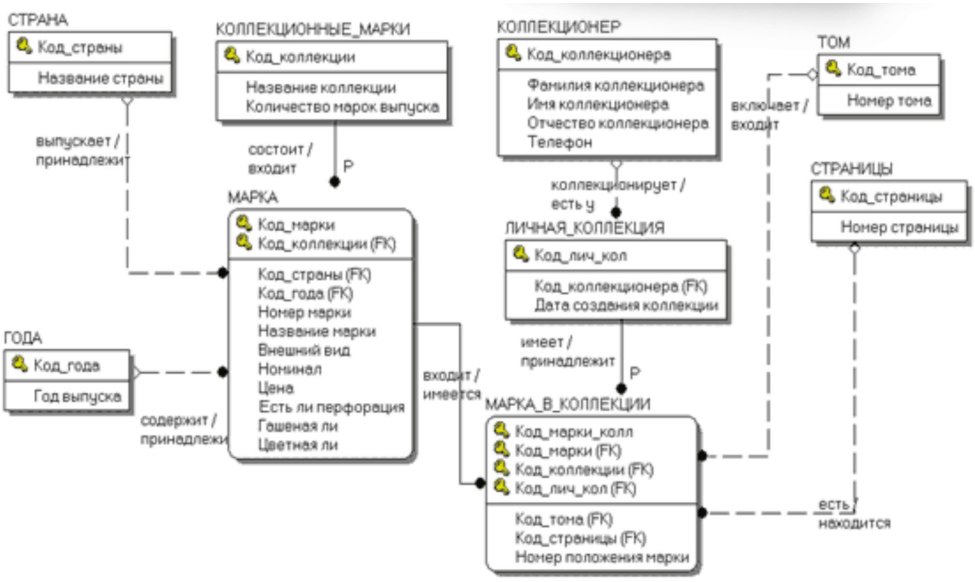


Рисунок 2.19 – Диаграмма модели в IDEF1 [27]

Блоки в это нотации так же описывают сущности, однако в отличии от IDEF1, тут они описывают конкретный набор объектов, не включающий в себя абстрактный набор информационных отображений реального мира. Первичным ключом называется атрибут или набор атрибутов с уникальными для определенной сущности значениями, по которым можно идентифицировать сущность. Внешние ключи - переданные через связь первичные ключи родительского объекта. Атрибуты мигрируют, если передаются через связь между объектами. Если уникальность дочерней сущности зависит от внешнего ключа, то она зовется зависимой сущностью. В нотации IDEF1X зависимые сущности часто представлены в виде закругленных прямоугольников.

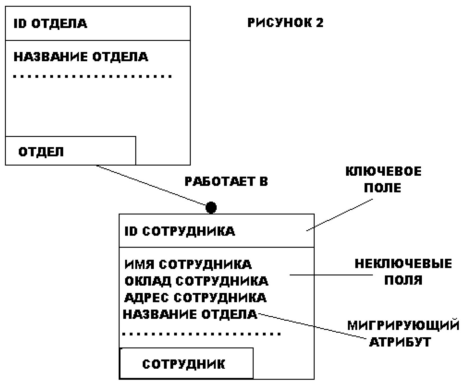


Рисунок 2.20 – Блоки и их атрибуты в IDEF1X [15]

IDEF3

IDEF3 - стандарт для документирования технологических процессов, происходящих на предприятии, который позволяет наглядно изобразить их сценарии.

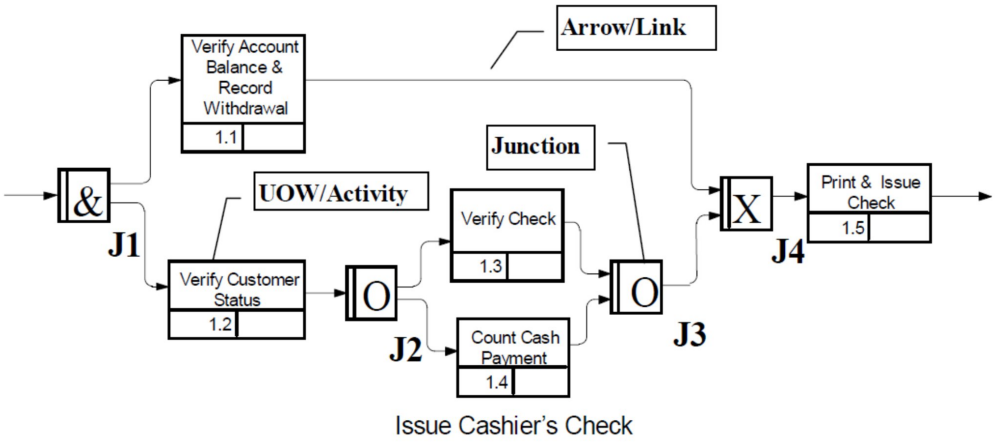


Рисунок 2.21 – Диаграмма в модели IDEF3

Диаграммы в этой модели состоят из блоков, односторонних стрелок, перекрестков стрелок и объектов ссылок. Блоки представляют собой функции (Unit of Work/activity), перекрестки отображают логику слияния/разветвления стрелок, объект ссылки включает в себя идеи/данные/концепции, не покрываемые остальными объектами нотации.

Виды стрелок:

- Старшая - Temporal Precedence - временное предшествие между UOW и UOW. Блок-источник сначала должен завершиться для выполнения блока-цели;
- Стрелка отношений - Relational - нечеткое отношение между двумя блоками, а также между блоками и объектами ссылок (пунктирная стрелка);
- Поток объектов - Object Flow - поток объектов между блоками, пример: объект является выводом одной функции и используется в другой (стрелка с двойным наконечником)

Объектами ссылки являются: объект (Object), заслуживающий отдельного внимания; ссылка (GOTO) для реализации циклов; единица действий (UOB) позволяющая многократно отображать одно и то же действие без цикла; заметка (Note) для документирования важной информации общего характера, уточнение (ELAB) для подробного описания диаграммы.

DFD

DFD - Иерархия диаграмм потоковых данных, описывающих асинхронный процесс преобразования информации от ее входа в систему до выдачи пользователю [26]. Диаграммы состоят из внешних сущностей, процессов, хранилищ данных и потоков данных. Внешние сущности представляют объекты или физические лица, которые являются источниками или

приемниками информации, к тому же сущность может быть использована многократно [12]. Потоки данных определяют поток объектов между частями системы. Процессы преобразуют входные данные из потоков, а хранилища данных являются устройствами хранения объектов.

Инструменты для реализации SADT:

- SparX Enterprise Architect
  - ER-диаграммы
  - DFD
  - Работы с реляционными базами данных.
- Miro
  - Эмуляция диаграмм и связей
- Draw IO и Confluence
  - Диаграммы (с трудом)

Инженерия Требований

Инженерия Требований (Requirements Engineering) - подраздел системной инженерии, занятый выявлением, разработкой, прослеживанием, анализом, проверкой соответствия, установлением взаимосвязей и управлением требованиями, которые определяют систему на последовательных уровнях абстракции [11].

Рисунок 2.22 – V-модель инженерии требований [11]

- Выявление требований - процесс ревью, формулировки и понимания потребностей клиента и ограничений программного обеспечения.
- Анализ требований - анализ потребностей клиента для определения требований
- Уточнение требований - процесс разработки документа (спецификация требований к ПО) в котором четко излагаются требования

- Проверка требований - процесс проверки соответствия требований и спецификации требований к программному обеспечению потребностям клиентов и системы.
- Управление требованиями - планирование и контроль процессов Инженерии Требований.

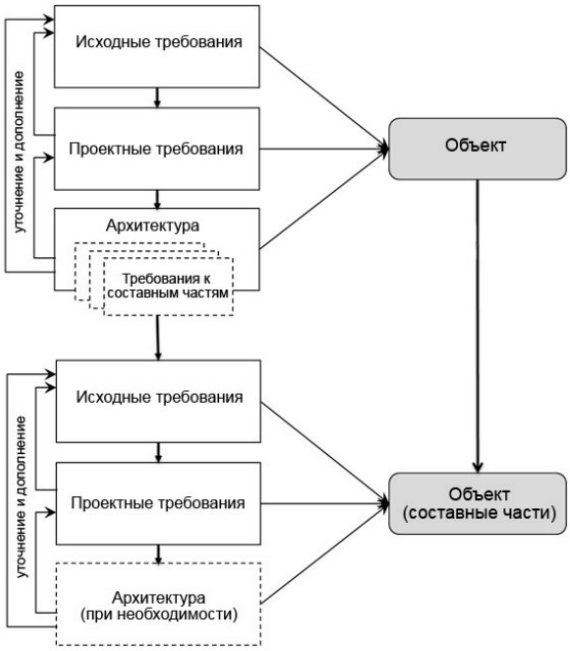


Рисунок 2.23 – Инженерия требований [16]

В результате анализа получают исходные требования: потребности заинтересованных сторон и требования к системе конкретного типа обусловленные документами по стандартизации. На основе исходных требований разрабатываются проектные требования, состоящие из описания функции системы, технических характеристик, описание принципов и режимов работы, условий эксплуатации, характеристики взаимодействия с внешними объектами (интерфейсы) и т.д... В случае сложных систем разрабатывается архитектура в ходе проектирования, и разрабатываются требования для составных частей системы.

Инструменты для Инженерии Требований:

- SparX Enterprise Architect
  - Работа со списками
  - Классификация элементов
  - Трассировочные таблицы
  - API для управления и доступа к информации
  - Хороший импорт/экспорт списков и связей
  - Авторасклaдка элементов со связями на диаграмме
- PlantUML
  - Контроль изменений в GIT
  - Генерация представления кода в различных видах, в том числе как диаграмма

- Miro
    - Ограниченная работа со списками
    - Расширенные функции упорядочивания и перехода от конвергенции к дивергенции при обсуждении
- Notion, Trello, Google Sheets
    - Списки
    - Связи
- Confluence
    - Списки
    - Связи
    - Контроль изменений в GIT
    - Создание документации
    - Трассировки

Предметно-ориентированное проектирование (*domain-driven design*, DDD)

DDD – подход к проектированию, направленный на создание высококачественной модели ПО, которая максимально точно соответствует поставленному в основание бизнес-процессу[17]. Подход полезен для правильной и корректной организации структуры модели, когда программист не является специалистом в предметной области поставленной задачи.

Единый Язык (Ubiquitous Language)

Единый язык можно представить как совокупность фраз, терминов ми различных понятий, которые использует команда при создании модели. Концепция единый язык помогает найти договоренность между экспертами в предметной области и программистами и привносит более глубокое понимание процессов всем участникам команды. Важно отметить, что DDD стремится четкому следованию архитектуры ПО за строением предметной области. Это несет за собой два полезных свойства: минимальное сопротивление архитектуры перед изменениями; в устоявшихся предметных областях сразу получается более качественная модель. Что стоит за этими двумя свойствами? Уменьшение избыточных обобщений и привнесенных жестких связей там, где их нет в предметной области. К примеру, можно привести жесткую связь один заказ - один человек в приложении такси. Однако в реальной жизни возникает необходимость заказывать такси не только себе, но и другим людям, что противоречит нашей поставленной связи. Решением такой ситуации является вынесение подобной связи в бизнес-правило, с возможностью его изменить.

Существует множество методов, способных помочь и упростить процесс создания модели. Самым ярким и наиболее важным примером таких методов является Событийный штурм (Event storming).

Так как единый язык можно представить в виде глоссария, то в качестве инструментов, предоставляющих такую возможность хранения отдельных слов и словосочетаний, можно привести Notion и Trello.

Стратегическое моделирование

Ограниченный контекст является одним из шаблонов в этом направлении и представляет собой некую границу, в которой существует модель, с отображением единого языка в ней. Таким образом в любом контексте каждая модель однозначно определена и имеет свои свойства. Например, для описания какой-либо работы организации модуль сотрудника, которая в разных контекстах его работы содержит свои атрибуты.

Работа над задачей в совокупности намного сложнее, чем работа с той же задачей, но разбитой на несколько частей. Таким образом, стратегическое моделирование помогает разбить предметную область на подобласти, ссылаясь на функциональность каждой, с учетом ограниченного контекста. Такое разбиение позволяет выделить смысловое ядро, служебные подобласти, неспециализированные подобласти, правильно распределить специалистов и бюджет на создание каждой.

Рассмотрим подробнее эти подобласти:

- Смысловое ядро - подобласть, которая выделяет весь бизнес и имеет первостепенное значение для организации. Если мыслить стратегически, то основные ресурсы стоит направить именно сюда.
- Служебная подобласть - подобласть, которые являются второстепенными внутренними вещами, которые также важны, но не являются смыслом создания модели предметной области, например формирование заказа в приложении.
- Неспециализированные области - подобласти, без специального назначения, но требуемые для существования. Такие области обычно отдаются на аутсорс (например, оплата через какую-нибудь систему).

Тактическое моделирование

Предоставляет шаблоны программистам как строительные блоки, которые просты в тестировании, помогают допускать меньше ошибок, и доступны в понимании бизнесу. Такие шаблоны имеют технический характер и используются сугубо в реализации решения задач ограниченного контекста. Примеры таких шаблонов: сущности, объекты-значения, службы,

события, агрегаты и т.д. Каждый шаблон используется в конкретной ситуации и имеет свои характеристики. Рассмотрим подробнее:

- Сущность - объект, модель которого уникальна и дает возможность однозначно идентифицировать. Например, счет в банке, конкретный человек в системе.
- Объект-значение - чаще неизменяемый объект, который полностью определяется своими атрибутами. Например, {50 долларов}, где атрибуты — это число и валюта.
- Служба предметной области - операция, которая выполняется над объектами, но не может быть приписана как метод к одному из них.
- Событие - работает в связке с шаблоном Наблюдатель, который публикует произошедшие действия и передает этот факт в данный шаблон для обработки.
- Модуль - контейнер с определенным именем, содержащий некоторые, тесно связанные между собой объекты, с целью ослабления связи между классами.
- Агрегат - кластер объектов, которые рассматриваются как единое целое. Обращение к происходит через корень (сущность с уникальным идентификатором).
- Фабрика - объект, целью существования которого является создание сложных объектов, таких как агрегаты и сущности, которые почти невозможно задать через конструктор.
- Хранилища - область памяти, которая нужна для безопасного хранения содержащихся в ней элементов, в основном используется для агрегатов.

Плюсы использования:

- Упрощение реализации сложных проектов
- Низкая стоимость дальнейшей разработки в связи с поддержанием модульности

Минусы использования:

- Сложности коммуникации между экспертами и программистами
- Отсутствие должного опыта при работе с данным подходом

Метод Событийного штурма.

Метод, который позволяет нам создать и организовать устойчивую модульную систему.

Как можно кратко описать? Сессия, при проведении которой нам необходимы большая доска, стикеры, команды разработчиков и экспертов в предметной области, которые могут помочь с ответами на вопросы разработчиков.

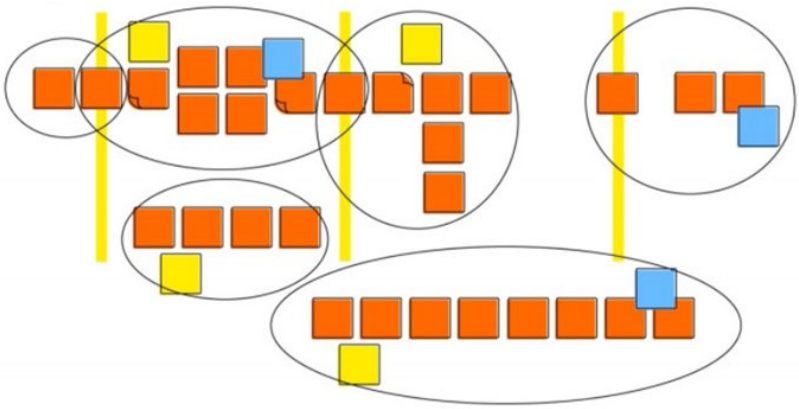


Рисунок 2.24 – Метод Событийного штурма

При проведении этой сессии в конечном итоге получается модель с агрегатами, командами, областями и ограниченным контекстом. Агрегаты и ограниченный контекст — это, в том числе, кандидаты на микросервисы или сервисы. Таким образом шторминг позволяет сразу наметить архитектуру и ответственность отдельных сегментов. Размеры таких досок могут сильно варьироваться в зависимости от количества человек, участвовавших в сессии, и размеров предметной области, которую нужно описать с помощью кода.

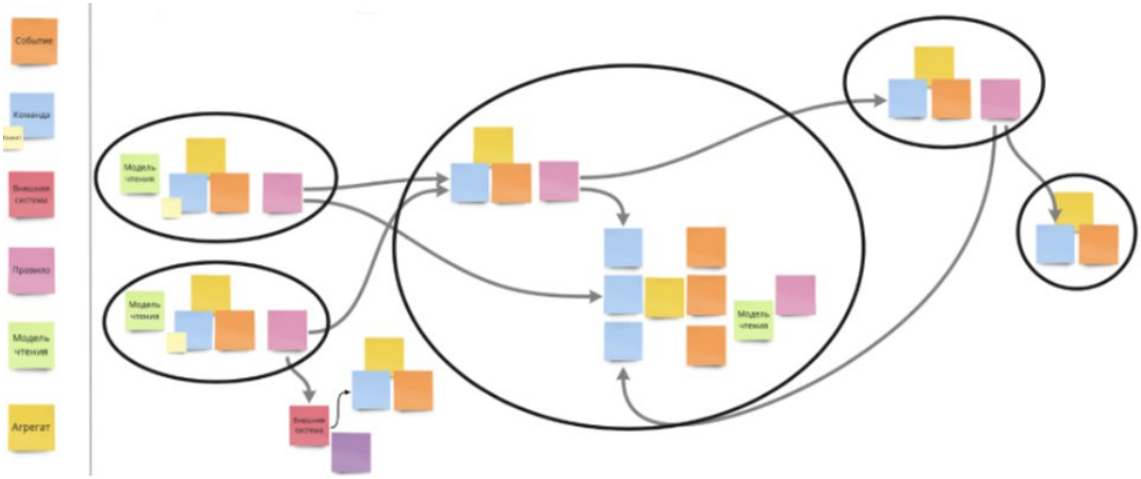


Рисунок 2.25 – метод Событийного штурма

Что происходит на этой сессии, указанной на Рисунке 2.25?

1. Оранжевыми карточками генерируются все события в хронологическом порядке, уточняется смысл каждого события, что помогает программистам лучше понять суть. Как пример: в онлайн-магазине (продукция добавлена в корзину-> … -> заказ оплачен -> …).

2. Голубыми карточками к каждому событию приписывается команда по его воспроизведению. Довольно простой пример: продукция добавлена в корзину (событие) -> добавить в корзину(команда).

3. Сиреневыми карточками описываются правила, которые нужны для обработки события. Также на этом и следующем этапах появляются стрелочки взаимодействие вырисовывающихся модулей.

4. Малиновые карточки отвечают за поступление данных к объектам из внешней системы.

5. Зеленые карточки означают модель чтения-представления. Добавляются и обновляются на основе событий с целью помощи в принятии решений клиенту и выполнении следующих команд.

4. Желтыми карточками обозначаются агрегаты (кластер объектов), связанные с остальными карточками. Также агрегаты являются кандидатами на микросервисы и сервисы. Т.е. в процессе шторминга можно выделить основные объекты и контексты. Пример: заказ разделяется на товар, адрес и т.д. При чем при создании доски их можно оставлять пустыми, а уже в конце дополнять.

5. Выделение ограниченных контекстов (показано кружочками), где каждый контекст является автономным и содержит в себе не противоречащие друг другу элементы, то есть сохраняется единый язык, и каждая сущность не имеет разных вариантов существования. Также можно выделить тип взаимодействий этих контекстов, что приводит к созданию такого шаблона стратегического программирования как карта контекстов.

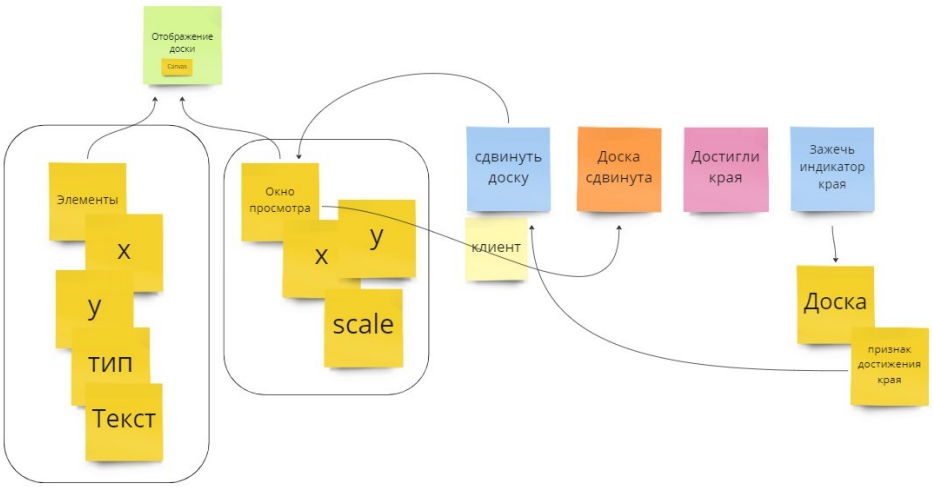


Рисунок 2.26 – Описание отдельного действия на доске

Инструменты для DDD:

- Физическая доска
  - Самым простым примером в данной ситуации является именно физическая доска или даже одноцветная пустая стена, которая предоставляет шанс наклеивать стикеры целой команде.
- Miro
  - Среди аналогов к созданию приложения для совместной работы можно выделить Miro. Функционал данного аналога позволяет создавать: карточки различного цвета с редактируемым текстом; стрелочки, которые помогают устанавливать связь между объектами; различные доски; фигуры для

объединения в контекст; а также поддерживает совместную работу большого количества человек. То есть сессия может быть проведена с помощью Miro и любого сервиса для видеоконференций.

- Confluence/Draw IO
  - Можно рассмотреть подобные инструменты, в качестве редакторов, предоставляющих возможность рисовать диаграммы, в том числе и что-то близкое к тому, что позволяется в физическом виде. Однако эти инструменты содержат большой минус для команды - неспособность одновременной работы.

Сравнительный анализ функций инструментов

Таблица 1 – Сравнительный анализ функций инструментов. Анализ на наличие определенных функций у популярных инструментов проектирования, прототипа и идеала.

Инструменты	SparX EA	PlantUML	Miro	Notion	Trello	Google sheets	Confluence	DrawIO	Прототип	Идеал
Диаграммы	Есть	Есть	Есть	Нет	Нет	Нет	С трудом	С трудом	Есть	Есть
Карточка	Есть	Нет	Есть	Есть	Есть	Нет	Есть	Нет	Нет	Есть
Список объектов	Есть	Нет	Недостаточно хорош	Есть	Есть	Нет	Есть	Нет	Есть	Есть
Связи	Есть	Только графические	Только графические	Нет	С помощью тегов	Нет	Нет	Нет	Есть	Есть
Простота использования	Нет	Нет	Есть	Есть	Есть	Есть	Есть	Есть	Есть	Есть
Совместная работа	Нет	Нет	Есть	Частично	Есть	Есть	Нет	Нет	Есть	Есть

Контроль изменений	Есть	Нет	Есть	Есть	Нет	Есть	Есть	Нет	Нет	Есть

3. Как видит пользователь?

Основа самого приложения - доска, на которой размещаются компоненты в конкретных координатах. Для удобства использования в рамках проектирования в различных методологиях требуется представлять данные не только в графическом, но и в табличном виде, что поддерживает приложение. Доступны различные манипуляции с доской, а именно сдвиг, уменьшение и увеличение видимого пространства с объектами.

Доступные к нанесению на доску независимые визуальные объекты:

- текст - содержит в себе текстовую информацию
- стикер - основан на тексте и прямоугольнике определенного цвета
- рисунок - интуитивно понятное рисование, объектом которого считается линия полученная за одно безотрывное касание

Доступные к нанесению на доску зависимые визуальные объекты:

- соединитель - показывает графически зависимость между двумя объектами. Поддерживает вариант самостоятельного существования, без присоединения к объекту на уровне кода.

Общий функционал:

- перемещение, то есть изменение координат
- появление дополнительного меню свойств, для регулирования внешнего вида объектов
- соединение нескольких объектов в один с сохранением общих свойств - группировка

Теперь рассмотрим возможные действия в формате вопрос-ответ:

➔ Каким образом объекты появляются на доске?

Текст и стикер имеют одинаковый путь появления: в меню выбирается конкретный интересующий элемент и кликом выбирается место появления объекта на доске.

Для рисования используется состояние нажатия кнопки, то есть, когда рисование активно, остальные функции отключаются и можно спокойно рисовать в интуитивно понятном формате.

Соединитель доступен для графической связи двух объектов: выбираем интересующий нас главный в связи объект и при нажатии ведем к второстепенному.

➔ Каким образом объекты изменяют свои свойства?

Местоположение объектов изменяется зажатием и перемещением по доске. Основные свойства, например, цвет шрифта или стикера, изменяются в всплывающем меню, где указаны актуальные свойства объектов, для этого нужно нажать на объект, взяв его в фокус (появится видимая рамка вокруг). Также, при выделении объекта на доске, он автоматически подсвечивается в таблице справа, где его можно найти и в нужной колонке изменить желаемое свойство.

➔ Как получить табличный вид объектов?

Автоматически видно справа.

➔ Как происходит группировка объектов?

Интуитивно понятное выделение с помощью окна с последующей возможностью изменения только общих свойств сгруппированных объектов.

4. Архитектура

Ядро

Основные данные и составляющие приложения хранятся в отдельном объекте класса Context, который содержит в себе объекты классов:

- EventsHistory - класс, отвечающий за сохранение истории действий пользователя, ей синхронизацию с репозиторием.
- EventHandlers - содержит функционал обработки событий из лога приложения, вспомогательный класс для EventsHistory.
- ObjectsStorage - хранит в себе объекты Object, созданные в процессе работы приложения. Заполняется автоматически при загрузке из лога или при создании пользователем. Object - базовый класс всех графических объектов со своими стандартными функциями, нужными для общей работы приложения.
- StateMachine - класс, отвечающий за переключение состояний, например, редактирование и добавление конкретного объекта. Каждое состояние отвечает имеет свой ограниченный функционал и существует для избежания наложений действий друг на друга.
- logging.Logger - осуществление записи логов работы.
- tkinter.Canvas - инструмент графического отображения.
- ttk.Frame - размещение меню свойств конкретного объекта.

- Menu - основное меню приложения для выбора элементов для отображения на доске.
- Broker - вспомогательный класс для осуществления поведенческого шаблона проектирования передачи сообщений издатель-подписчик.

При старте программы происходит инициализация контекста и создание подобъектов, требуемых для запуска графической составляющей проекта.

Модульность

Одной из задач в построении приложения стала независимость частей, отвечающих за свои направления, чтобы вести параллельную разработку без поддержания постоянной связи, легкого введения расширений и улучшений уже существующих модулей. Помимо основного ядра, в существующее приложение удалось внедрить конкретные модули, отвечающие за:

- Стикер
- Текст
- Карандаш
- Коннектор
- Меню свойств
- Движение доски
- Движение объектов
- Группировка объектов
- Уменьшение, увеличение видимого пространства

При инициализации модуля, в качестве аргумента отдается контекст для успешной регистрации и заполнения специфических свойств. Для того, чтобы зарегистрировать модуль в приложении, требуется наличие конкретных действий:

- Определение состояний модуля, организация кода по шаблону: наличие уникальных предикатов перехода между состояниями, возможные действия, которые должны выполняться при входе или выходе. Организация обработчиков записи в лог, если изменения, происходящие в текущем состоянии, подлежат фиксации.
- Если требуется добавление в меню кнопки, то, используя контекст, самостоятельно позаботиться о ее возникновении с подходящим по смыслу названием.
- Регистрация типов визуальных объектов, например, стикер, текст, для наличия типизации объектов в ObjectsStorage.

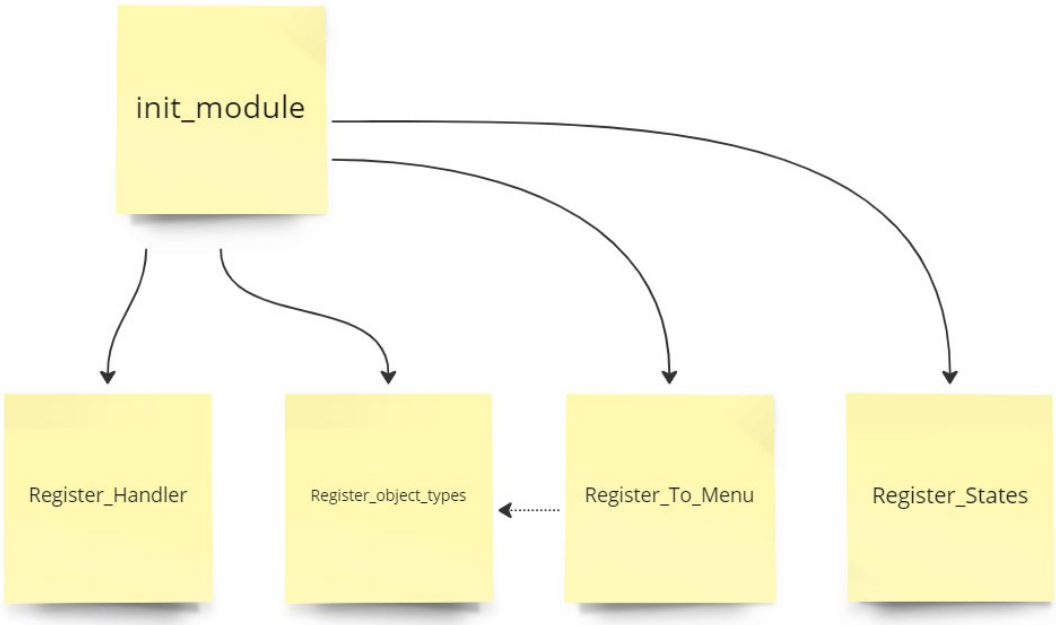


Рисунок 4.1 – Процесс инициализации модуля

Модули подгружаются автоматически из старта программы, если соблюдены все требования к ним.

Состояния или StateMachine, чем является и для чего существует

В процессе работы с tkinter-ом возникло множество проблем с наложением событий. Решением этой проблемы стало решение ввести состояния под действия, которые доступны пользователю. Каждое состояние имеет свои предикаты - условия входа и выхода из него, а также функции, отвечающие за обработку действий, произведенных при входе, в самом состоянии и на выходе. Каждое состояние ограничивается своей функциональностью. Схема условий переходов между всеми состояниями:

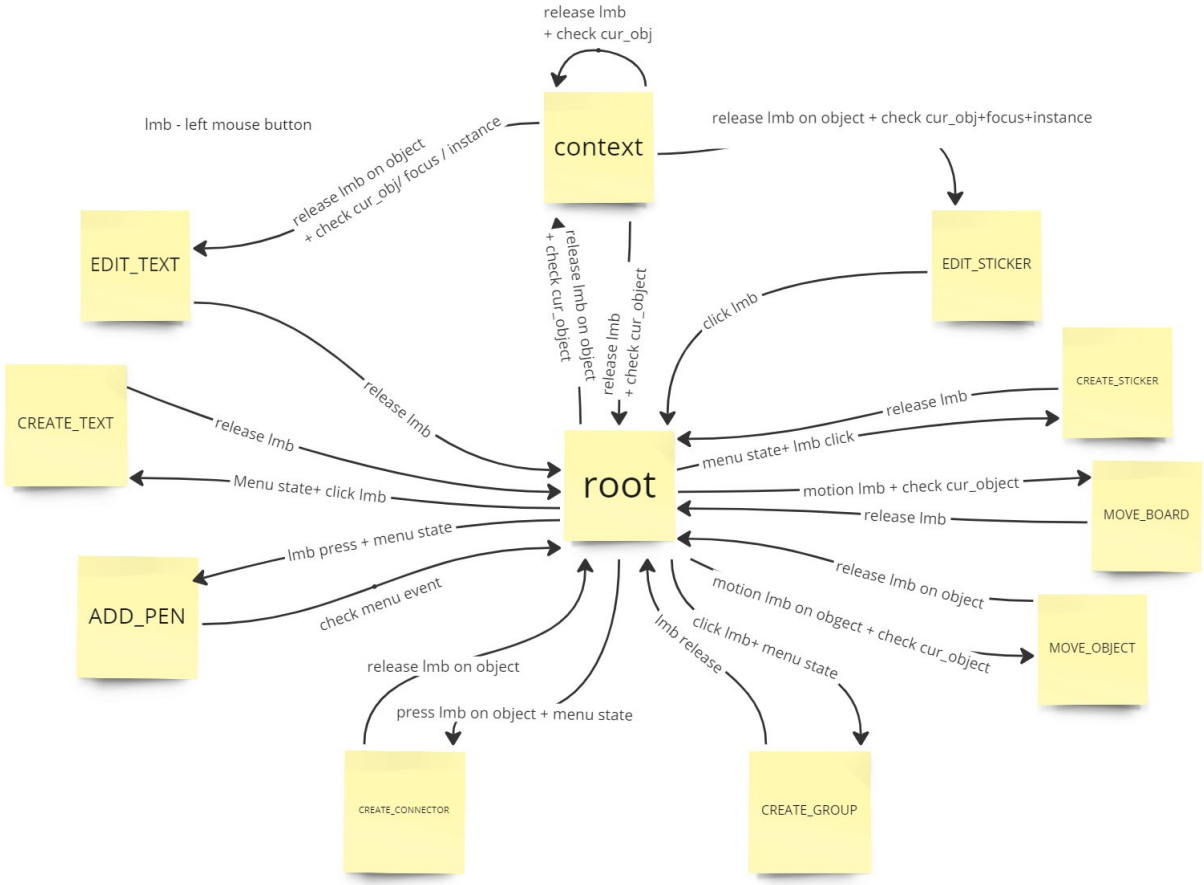


Рисунок 4.2 – Состояния и переходы между ними

Основные переходы осуществляются с помощью левой кнопки мыши, проверки текущего объекта и изменения состояния меню. Происходят в большинстве своём между начальным состоянием и состоянием, необходимым для заданного изменения, за исключением редактирования текста: на обработку текста накладывается дополнительная проверка при переходе из промежуточного состояния.

Всю работу по переключениям выполняет объект класса StateMachine, хранящийся в контексте программы. Во внимание принимаются фиксированные события и действия пользователя, которые отслеживаются и перенаправляются в стандартный обработчик. Тем временем обработчик проходится по предикатам переходов и, если предикат вернул истинное значение, то происходит переход в это состояние. В машину автоматически подгружаются все состояния и действующие переходы между ними, о чем заботится создатель очередного нового модуля.

В течении жизни программы фиксируются текущие изменения и действия в EventsHistory с помощью поддержания листа из объектов класса EventInfo, которые содержат в себе информацию о конкретном одном изменением от пользователя. Поэтому при совершении

некоторых действий требуется как сохранение события в истории, так и обеспечение обратной выгрузки в обработчиках.

Сохранение созданных объектов

Каждый объект содержит свой уникальный идентификационный номер (id в последующем), свойства и переменную состояния фокуса, от которой зависит поведение предмета. Сохранение происходит в ObjectsStorage с помощью словаря, где по id можно достать определенный объект. В дальнейшем объект может описываться своими дополнительными свойствами с помощью наследования от базового класса. Объект может принимать действия в зависимости от действий другого объекта, если является его наблюдателем, например, коннектор отслеживает передвижения объектов, которые он соединяет. При создании модуля происходит регистрация визуальных объектов, поэтому в методе create в классе ObjectsStorage, можно ссылаться на конструкторы конкретных нужных нам инструментов.

Таблица

Визуальное представление объектов в табличном виде с возможностью изменения свойств - при нажатии строку, отвечающую за элемент, появляется окно с описанием и функцией изменения свойств объекта. Является элементом контекста, находясь визуально на одном уровне с доской. Автоматически обновляет содержимое при добавлении элементов на доску или изменении характеристик при помощи всплывающего меню свойств. Содержит связь с состоянием context из StateMachine - при выделении объекта на канвасе, автоматически подсвечивается отвечающая за него строка. Одним из дополнительных функционалов таблицы является сортировка по значению выбранной колонки свойств (для этого требуется нажать на название столбца).

Перед сортировкой элементы добавляются в порядке создания:

Тип объекта	Шрифт	Размер шрифта	Насыщенность ш	Наклон шрифта	Цвет шрифта	Ширина карточк	Цвет карточк
STICKER	Calibri	8	bold	italic	light pink	100	light yellow
STICKER	Arial	14	bold	italic	black	100	light yellow
STICKER	Leelawadee	10	normal	roman	black	100	light green
TEXT	Arial	36	normal	roman			
TEXT	Arial	10	normal	roman			

Рисунок 4.3 – Таблица с сортировкой по порядку создания

После сортировки по шрифту:

Тип объекта	Шрифт	Размер шрифта	Насыщенность ш	Наклон шрифта	Цвет шрифта	Ширина карточк	Цвет карточк
TEXT	Arial	10	normal	roman			
STICKER	Arial	14	bold	italic	black	100	light yellow
TEXT	Arial	36	normal	roman			
STICKER	Calibri	8	bold	italic	light pink	100	light yellow
STICKER	Leelawadee	10	normal	roman	black	100	light green

Рисунок 4.4 – Таблица с сортировкой по шрифту

Примечание к картинкам: для перемещения по таблице есть полоса прокрутки, так как все свойства объектов имеют большую размерность, чем представление на стандартном экране.

Дополнительные модули

Как уже было сказано, основа модуля строится на его регистрации и разбиении функционала на состояния, с определенной структурой описания требуемых действий и возможностей. Модули на данный момент программы:

- текст

Содержит в себе текст, содержание и свойства которого можно менять. Состоит из двух состояний: само создание с использованием стандартного текста “new text” и изменение содержания с помощью функции фокусировки и выставления курсора от tkinter-a. Свойства, которые можно изменять в тексте с помощью всплывающего меню: наклон, цвет, тип, толщина шрифта.

а) б)

Рисунок 4.5 – а), б) Пример использования текста

- стикер

Модуль стикера схож с модулем текста, однако к нему добавляется рамка наподобие реального бумажного стикера, цвет которого можно менять. Содержит в себе описание двух таких же состояний создания и изменения. Свойства, доступные пользователю: наклон, цвет, тип, толщина шрифта; цвет и длина карточки.

а) б)

Рисунок 4.6 – а), б) Примеры использования стикера

- карандаш

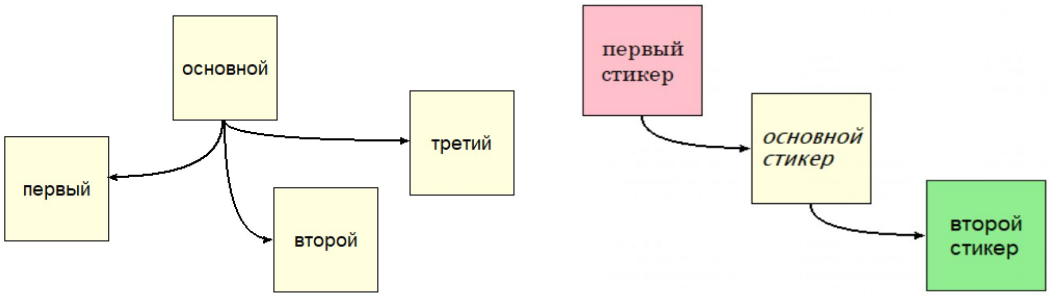
Модуль создан для простого рисования, которое может быть полезно для проведения границ или быстрого визуального наброска в проектировании. Построен на состоянии создания линии и предоставляет к изменению свойства толщины и цвета линии.

а) б)

Рисунок 4.7 – а), б) Пример использования карандаша

- соединитель

Модуль создан для удобного создания графических связей визуальных объектов. Состоит из состояния создания связи, для чего нужно выбрать в меню “connector” и соединить два нужных предмета, ведя от основного к второстепенному.



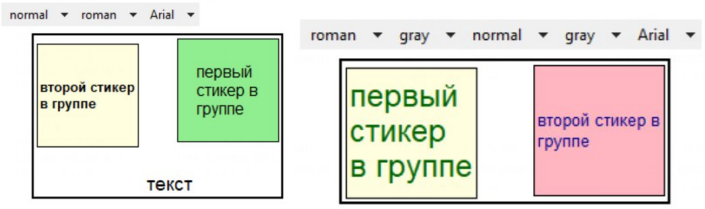
а) б)

Рисунок 4.8 – а), б) Примеры использования соединителя

- группа

Используется для соединения выделенных объектов в одну сущность с сохранением возможности изменений общих характеристик. Хранит в себе список выделенных объектов и

подписывается на их обновления с помощью шаблона издатель. Поддерживает изменение свойств, общих для всех объектов, находящихся в группе.



а) б) с)

Рисунок 4.9 – а), б), с) Примеры использования группы

- передвижение доски

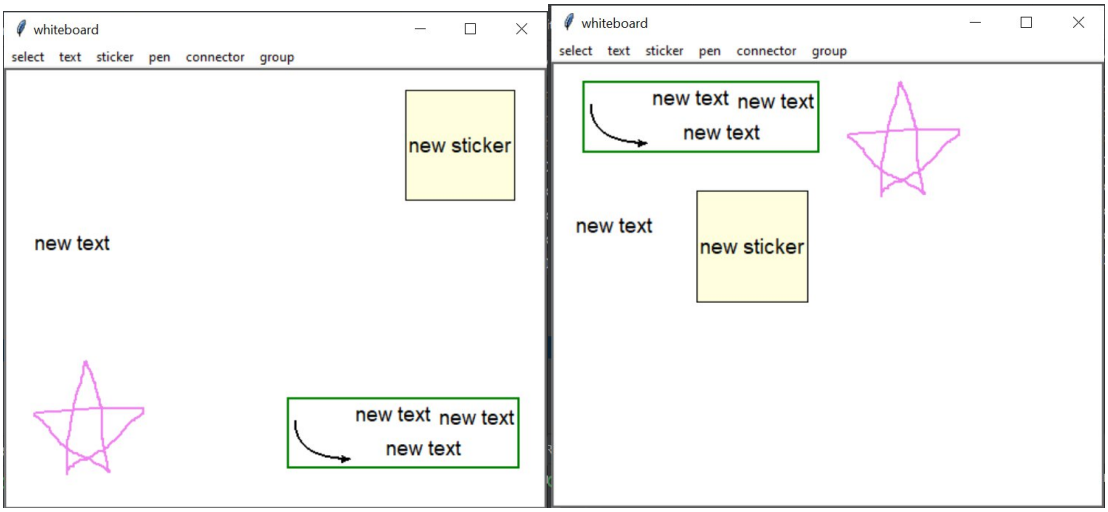
Доска в данном прототипе является безграничной, но отобразить все сразу физически невозможно. Для этого существует функция, активируемая по нажатию и передвижению доски в нужном направлении, состояние только одно и сохраняется оно, пока нажата левая кнопка мыши на доске. Так открывается визуальный доступ ко всем объектам на любой зоне доски.

а) б)

Рисунок 4.10 – а) доска до передвижения, б) доска после передвижения

- передвижение объектов

Данный модуль отвечает за то, что позиция любого объекта не перманентна, а изменяема. Объекты можно организовать в различных порядках, что помогает лучше выразить графическую информацию. При нажатии и передвижении любого объекта, аналогично с передвижением всей доски, активируется и деактивируется единственное состояние этого модуля.



а) б)

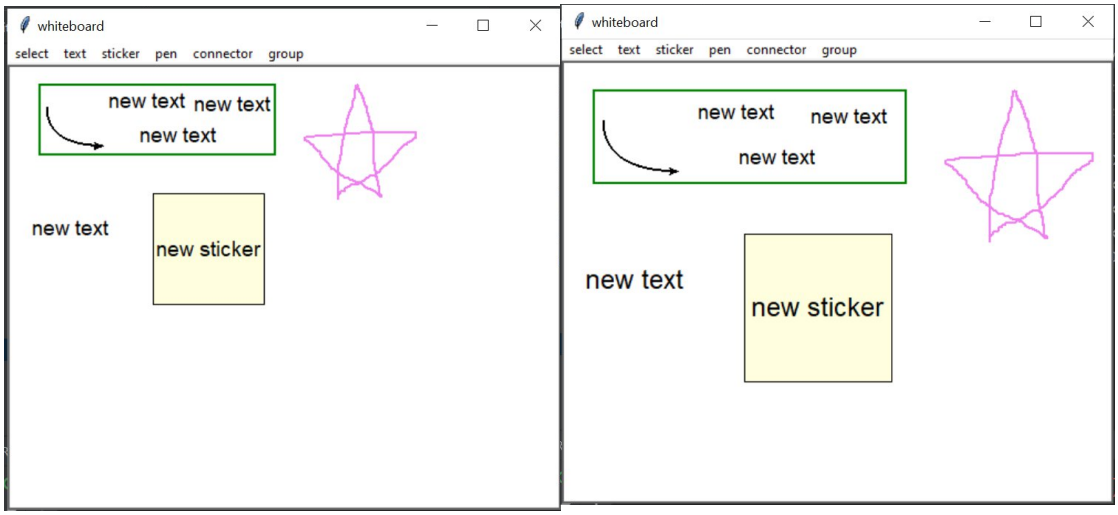
Рисунок 4.11 – а) до передвижения объектов б) после передвижения объектов

- удаление объектов

Удаление из состояния context с помощью кнопки Delete.

- изменение масштаба доски (приближение или удаление)

Является единственным модулем без состояний, что значит функция, за которую он отвечает выполняется вне зависимости от того, в каком состоянии сейчас находится машина. Приближение и удаление осуществляется щипковым движением внутрь/наружу (pinch in/out) на тачпаде, либо комбинацией Control + движение колесика мыши.



а) б)

Рисунок 4.12 – а) до приближения объектов б) после приближения объектов

- меню свойств

Является всплывающим окном со свойствами, которые предоставляют отдельные визуальные модули. Помогает контролировать внешний вид каждого объекта.

Содержит в себе одно состояние - в описании StateMachine оно называется context. В этом состоянии над объектом появляется всплывающее меню с доступными для изменения

свойствами, которые предоставляет каждый класс по отдельности. Для входа в это состояние, нужно выбрать кликом левой клавиши мыши над выбранным объектом. При входе в context, создается меню свойств и сохраняется до выхода из этого состояния с помощью соответствующих действий: клика левой клавиши мыши в любом месте (над объектом - произойдет переход в другой context, в случайном месте на доске - переход в корневое состояние приложения). Сам объект Submenu устроен на инициализации с помощью подгрузки поддерживаемых свойств от класса объекта; функции, отвечающей за изменение свойств и своем удалении.

а) б)

Рисунок 4.13 – а), б) Примеры отображения меню свойств

Совместная работа с доской

Изначально у нас было несколько вариантов реализации совместной работы с доской, их можно поделить на две группы: с использованием самописного сервера и с использованием Git’а с репозиторием на сервере. Исследование подходов проектирования, программ-аналогов; проработка и реализация архитектуры заняли много времени, поэтому было решено остановиться на более простом варианте с использованием Git.

Синхронизация в приложении работает следующим образом:

1. При запуске приложения происходит pull основной ветки репозитория. Пользователь может выбрать имя существующего файла с событиями или создать новый. Т.е. есть возможность работы с несколькими досками.
2. После выбора доски из списка событий строится доска. Также вводится понятие “версии объекта”: она равна количеству событий, которые произошли с объектом.
3. Раз в 30 секунд начинается процесс синхронизации событий:
    a. Происходит pull списка событий из репозитория. Строится обновлённый список версий объекта.
    b. К актуальному списку событий из репозитория пытаемся добавить события, которые произошли локально. Пусть хотим добавить событие для определенного объекта. Если оказывается так, что с предыдущего момента синхронизации версия этого объекта из репозитория изменилась, то произошёл конфликт, в таком случае все локальные события, произошедшие с этим объектом, не будут добавлены.

c. Создаётся коммит с измененным логом событий. Происходит попытка push’а в удалённый репозиторий. Если происходит мёрж-конфликт, то процесс происходит повторно (начиная с пункта a).
d. Если при синхронизации появились какие-то новые события, то доска перестраивается с нуля по обновленному списку событий.

5. Шаблоны проектирования

Популярный способ создания чего-либо - использование базовых конструкций и решений, которые упрощают код и делают его более читаемым и простым для понимания. С ростом опыта программирования и проектирования, росло и количество таких архитектурных решений, которые в дальнейшем называются шаблонами. У использования уже готовых абстракций кода есть свои преимущества и недостатки: снижается количество допускаемых ошибок и сложность кода, код становится переиспользуемым и легко встраиваемым в другие проекты с похожими задачами, однако при этом слепое доверие шаблонам может загнать и в тупик, а также возникнуть желание использовать их там, где этого делать не стоит. Рассмотрим пару подходов, которые оказались полезны для развития проекта.

Фабрика

Является порождающим шаблоном проектирования, поэтому основное применение ушло в часть с созданием визуальных объектов. Как обсуждалось ранее, каждый визуальный класс является наследником класса Object со своим базовым интерфейсом. Таким образом основа каждого объекта определена в родительском классе, но может быть изменена и дополнена в дочерних классах в уже новых модулях. С помощью регистрации типов таких объектов, ObjectsStorage имеет универсальный метод создания для всех элементов - create, который делегирует это своим дочерним классам, а также знания о том, кто примет в обработку это задание из подклассов, хранятся только в ObjectsStorage с помощью словаря. Что также помогает узнать этот шаблон - у программы нет заранее готовой информации, какие объекты ей придется создавать, что применимо и в нашем случае: пользователь отрисовывает объекты по мере необходимости в его графической работе.

Наблюдатель

Появляется свойство объектов - подписываться на конкретное действие другого предмета. При работе появилось такое решение из-за зависимости коннектора от местоположения предметов, которые он соединяет. Каждый Object является по умолчанию подписчиком, а также может зарегистрироваться как издатель определенного события. В реализации присутствует

класс для использования этого шаблона - Broker. Он хранит в себе сеть существующих связей, позволяет создавать новые, инициализирую новых издателей со своими событиями и разрешая подписываться любому другому на события издателя. При разрушении созданной нами связи есть возможность отписаться.

Шаблон состояния

Популярный шаблон проектирования, который заключает в себе зависимость действий объектов от его внутреннего состояния. В реализации коллаборативной доски он встречается несколько раз. Одно из его использований завязано на пометке объекта, находится ли он в фокусе рассмотрения или нет. Второй применяется более обширно: меню строится на хранении состоянии нажатой опции, то есть полностью зависит от выбора пользователя. От состояния меню зависит по какому предикату перейдет StateMachine при проверке возможности перехода между состояниями.

Декоратор

Является структурным шаблоном проектирования, который заключается в динамическом добавлении новых свойств объектам с помощью обертки над ними. Аналогичная ситуация прослеживается в добавлении функциональных модулей, которые отвечают за перемещение объектов и изменения их свойств.

6. Применение для разных подходов проектирования

Продукт данной проектной работы может быть полезен для проектирования в разных методологиях. Для некоторых из нижеперечисленных подходов проектирования представлен пример, созданный на разработанном прототипе.

Story mapping

Является мощным инструментом, основанном на пользовательском пути. Такой подход применяется для достижения единого понимания потребностей пользователя, построения требований к минимальной работоспособной версии, а также возможностей дополнения проекта в будущем. Для создания путей использования приложения пользователем понадобятся стикеры и возможно соединители для этих стикеров. Далее каждые шаги нужно подробно уточнять и искать пути и нюансы в их реализации, для чего также понадобятся лишь стикеры и чуть больше места для их присоединения. Также одной из особенностей такого похода является приоритизация задач, а также поиск похожих действий и мест в схеме для их дальнейшего возможного объединения. Для таких целей понадобятся разноцветные стикеры, что также присутствует в работе. Story mapping помогает разделить все идеи на релизы, что делается

графически пунктирной линией. Такой функционал предоставляется карандашом, либо объединением карточек в одну группу, которая показывает границы выбранных элементов.

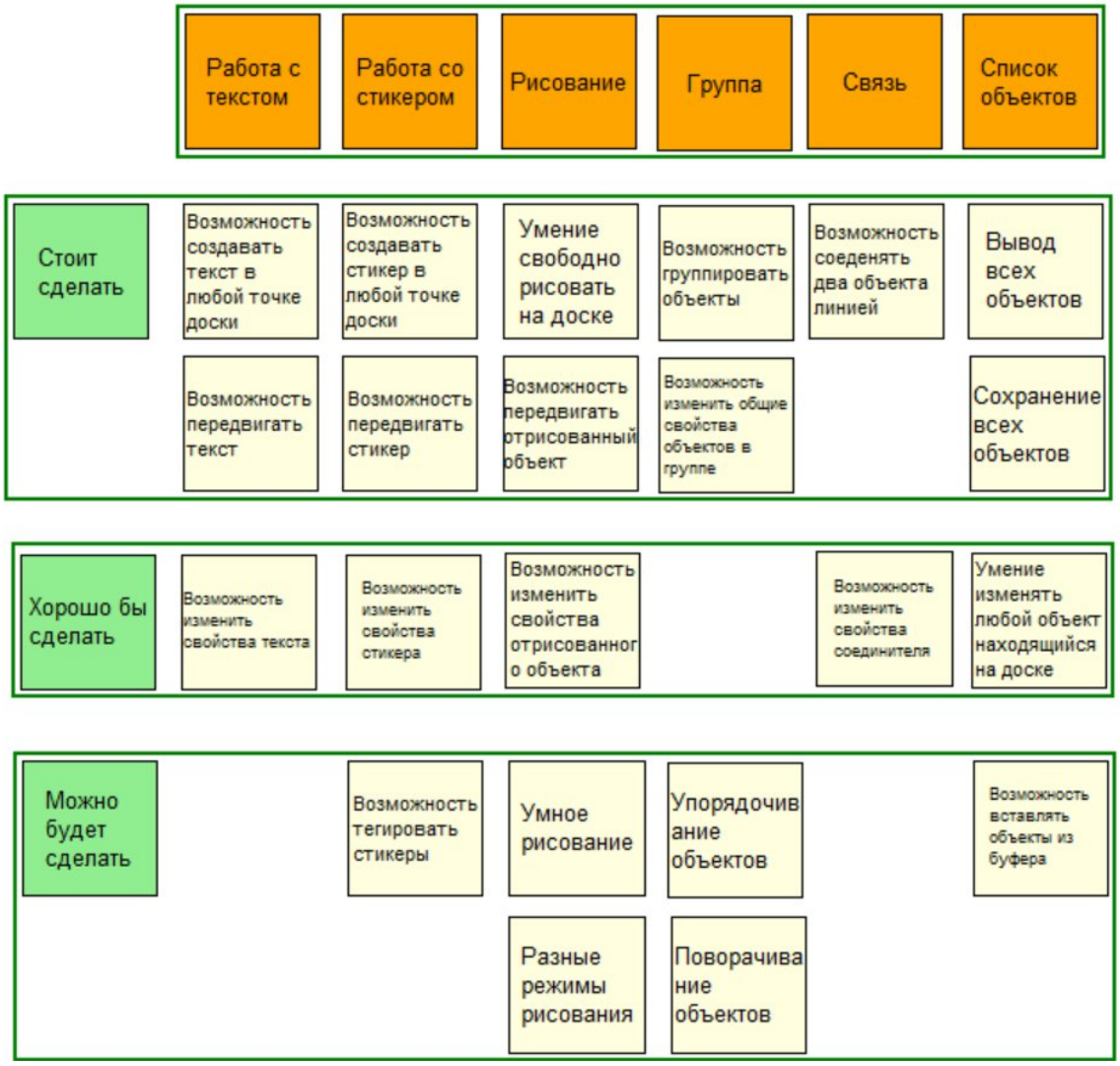


Рисунок 6.1 – Story Map данного проекта. Основные темы карты в оранжевых, степень важности в зеленых, а требования в бежевых стикерах.

Метод событийного штурма

Описание метода событийного штурма происходит в главе с подходом к проектированию DDD. Требования к элементам, нужным для проведения такой сессии с наметкой архитектуры будущего приложения, схожи с требованиями к встрече со Story mapping: разноцветные стикеры с возможностью размещения в произвольном месте, инструмент для объединения и программного и визуального этих объектов вместе. Такими инструментам могут являться также группировка и визуальное очерчивание зоны со стикерами карандашом.

CJM (Customer Journey Mapping)

Это карта, описывающая шаги клиента на пути к определенной цели. Триггер, карточки с намерениями клиента, карточки с его действиями и эмоциями в CJM можно отобразить стикерами, а смайлы, используемые в качестве шкалы эмоций, можно нарисовать или

использовать их текстовый вариант: **“:)”** (довольное выражение), **“:/”** (нейтральное выражение), **“:(”** (недовольное выражение)

IDEF0

Нотация IDEF0, упоминавшаяся в главе SADT, предназначена для описания бизнес-процессов в пределах одной системы. Границы диаграмм можно выделить группировкой объектов, а блоки, стрелки и дополнительную информацию изобразить стикерами, соединителями и текстом.

C4

Данная модель декомпозирует архитектуру проекта с разных точек зрения. Систему разбивают на 4 уровня: контекст, контейнеры, компоненты и код. Аналогично с IDEF0, можно разделить диаграммы разных уровней группировкой объектов, а как блоки использовать стикеры.

DFD (Data Flow Diagram)

Диаграмма описывает потоки данных и процесс их преобразования в системе. Сущности и процессы можно отобразить как стикеры разных цветов, а потоки нарисовать с помощью соединителей.

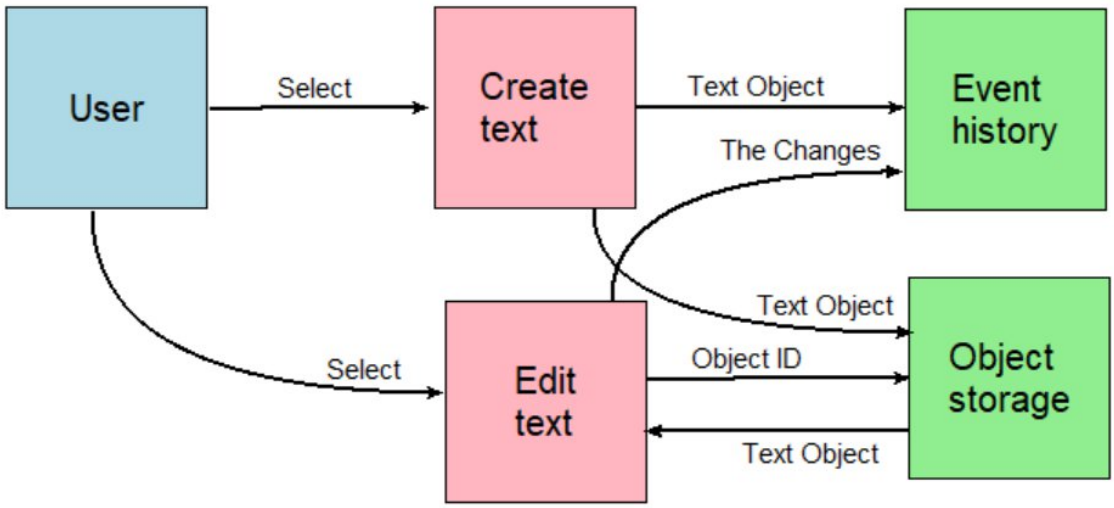


Рисунок 6.2 – Data Flow Diagram модуля текста, выполненная в продукте данной проектной работы. Внешние сущности в голубом, процессы в розовых, а хранилища данных в зеленых стикерах.

Диаграмма последовательности

Полезна для детального описания логической цепочки различных сценариев использования и выявления объектов, которые затрагиваются во время выполнения сценария. Основными компонентами такой диаграммы являются: объекты взаимодействия, которые можно представить в виде карточек; сообщения и обратные результаты, которые ходят от одного объекта к другому, что можно показать, как стрелки и текст над ними; для показания временных промежутков работы объекта можно использовать прямоугольники, однако это не является обязательным действием.

7. Заключение

В настоящей проектной работе был успешно реализован прототип коллаборативной доски, широко используемой в области проектирования. Перед созданием программы был проведен тщательный анализ распространенных методологий проектирования с целью определения необходимого функционала для будущей доски. На основе полученных знаний были разработаны основные компоненты, характерные для данной сферы, такие как текстовые элементы, стикеры, карандаши, соединители и группы объектов, которые эффективно функционируют.

Программа предполагает органическое развитие и добавление новых функций и компонентов, для чего требуется обеспечить расширяемость кода. Расширяемость была реализована путем использования модульной архитектуры, где внедрение нового объекта или метода подразумевает добавление модуля, написанного для ядра системы. Таким образом, каждый модуль функционирует независимо от других.

Существует множество потенциальных направлений развития данного проекта, которые не были учтены в связи с изначальной целью исследования различных подходов к реализации и их последующего тестирования. Например, для прототипа была выбрана библиотека tkinter на языке Python из-за ее доступности и полной документации, однако для достижения лучшей графики и производительности рекомендуется рассмотреть более современные подходы.

Кроме того, существует множество функций, которые были выделены в качестве требований к "идеальному инструменту", но не были реализованы в рамках данного проекта:

- Объекты, такие как фигуры и карточки с наборами свойств.
- Инструменты классификации на диаграмме.
- Дерево связей.
- Возможность совместной удаленной работы, аналогичная функциональности Miro и Google Docs

Список литературы

[1] Официальная спецификация BPMN // Object Management Group [Электронный ресурс]. Режим доступа: https://www.omg.org/spec/BPMN/2.0/PDF (дата обращения 12.02.2023).

[2] Зачем нам UML? Или как сохранить себе нервы и время // Хабр [Электронный ресурс]. Режим доступа: https://habr.com/en/post/458680/ (дата обращения 12.02.2023).

[3] PlantUML – инструмент продуктового разработчика // Блог компании «Qiwi» на портале «Хабр» [Электронный ресурс]. Режим доступа: https://habr.com/en/company/qiwi/blog/577606/ (дата обращения 12.02.2023).

[4] Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр [Электронный ресурс]. Режим доступа: https://habr.com/en/post/566218/ (дата обращения 12.02.2023).

[5] Краткое описание нотации BPMN // Блог компании «Auriga» на портале «Хабр» [Электронный ресурс]. Режим доступа: https://habr.com/en/company/auriga/blog/667084/ (дата обращения 12.02.2023).

[6] Документация PlantUML // Веб-сайт PlantUML [Электронный ресурс]. Режим доступа: https://plantuml.com/ (дата обращения 12.02.2023)

[7] Диаграмма активностей // Платформа Flexberry [Электронный ресурс]. Режим доступа: https://flexberry.github.io/ru/fd_activity-diagram.html (дата обращения 12.02.2023)

[8] Диаграмма вариантов использования // Платформа Flexberry [Электронный ресурс]. Режим доступа: https://flexberry.github.io/ru/fd_use-case-diagram.html (дата обращения 12.02.2023)

[9] UML Class Diagram // Ресурс с туториалами Javatpoint [Электронный ресурс]. Режим доступа: https://www.javatpoint.com/uml-class-diagram (дата обращения 12.02.2023)

[10] Deployment Diagram for E-commerce Microservices Architecture // Software Ideas Modeler [Электронный ресурс]. Режим доступа: https://www.softwareideas.net/a/1580/e-commerce-microservices-uml-deployment-diagram- (дата обращения 12.02.2023)

[11] Халл Э,, Джексон К., Дик Дж. Инженерия требований / пер. с англ. А. Снастина; под ред. В. К. Баторвина. – М. : ДМК Пресс, 2017.

[12] BPWin Methods Guide. Logic Works Inc., Princeton, New Jersey, 1997.

[13] Марка Д., МакГоуэн К. Методология структурного анализа и проектирования (SADT) / Пер. с англ. - М: МетаТехнология, 1993.

[14] Основы методологии IDEF1 // Корпоративный менеджмент [Электронный ресурс]. Режим доступа: https://www.cfin.ru/vernikov/idef/idef1.shtml (дата обращения 12.02.2023)

[15] Основы методологии IDEF1X // Корпоративный менеджмент [Электронный ресурс]. Режим доступа: https://www.cfin.ru/vernikov/idef/idef1x.shtml (дата обращения 12.02.2023)

[16] ГОСТ Р 59194–2020. - Издание официальное. - М. : Стандартинформ, 2020. - 29 с.

[17] Вон Вернон. Реализация методов предметно-ориентированного проектирования/ пер. с англ. под ред. Д.А. Клюшина, 2016

[18] Джефф Паттон. Пользовательские истории

[19] Гойко Аджич. IMPACT MAPPING. Как повысить эффективность программных продуктов

[20] 4-контекстная диаграмма // [Электронный ресурс]. Режим доступа: https://c4model.com/ (дата обращения 12.02.2023)

[21] Диаграмма последовательности UML // Планёрка [Электронный ресурс]. Режим доступа: https://planerka.info/item/diagrammy-posledovatelnosti/ (дата обращения 12.02.2023).

[22] Что такое унифицированный язык моделирования // Lucidchart [Электронный ресурс]. Режим доступа: https://www.lucidchart.com/pages/ru/uml (дата обращения 12.02.2023).

[23] UML State Machine // Wikipedia [Электронный ресурс]. Режим доступа: https://upload.wikimedia.org/wikipedia/commons/4/45/UML_state_machine_Fig1.png (дата обращения 12.02.2023).

[24] UML Component Diagrams // UML Diagrams [Электронный ресурс]. Режим доступа: https://www.uml-diagrams.org/component-diagrams.html (дата обращения 12.02.2023).

[25] BPMN Example // Wikipedia [Электронный ресурс]. Режим доступа: https://upload.wikimedia.org/wikipedia/commons/9/98/Quotation_BPMN_Example2.png (дата обращения 12.02.2023).

[26] Назаров С.В. Архитектуры и проектирование программных систем: монография /С.В. Назаров. — М.: ИНФРА-М, 2013. — *** с.

[27] Краткий путеводитель по семейству нотаций IDEF // AntonPiskun.pro [Электронный ресурс]. Режим доступа: https://www.antonpiskun.pro/kratkij-putevoditel-po-semejstvu-notaczij-idef/ (дата обращения 12.02.2023).