

# Содержание

<b>1 Перепись</b>	<b>2</b>
1.1 Санта-клаус . . . . .	2
1.1.1 Модель требований FURPS+ . . . . .	2
1.1.2 Use-case диаграмма . . . . .	3
1.1.3 Доменная модель . . . . .	4
1.2 Disciplined Agile Delivery, +, - . . . . .	4
1.2.1 Disciplined Agile Delivery . . . . .	4
1.2.2 + . . . . .	4
1.2.3 - . . . . .	4
1.3 Эльфы и тестовое покрытие . . . . .	4
<b>2 Рубеж</b>	<b>5</b>
2.1 Use-Case . . . . .	5
2.2 Модель Ройса . . . . .	6
2.3 Тестовое покрытие с использованием анализа эквивалентности . . . . .	7
2.4 Ресурсные риски . . . . .	7
2.5 Топ - 3 рисков по экспозиции . . . . .	8
2.6 Git pull . . . . .	8
2.7 Git commit . . . . .	8
2.8 Классификация проблемных ситуаций . . . . .	8
2.9 Скрипт сборки Gradle . . . . .	9
2.10 Тестовое покрытие . . . . .	9
2.11 Простой системы . . . . .	10
2.12 Скорость работы системы . . . . .	10
<b>3 Рубеж</b>	<b>10</b>
3.1 Доменная модель . . . . .	10
3.2 Чему уделялось внимание в спиральной модели . . . . .	10
<b>4 Термины</b>	<b>10</b>
4.1 Модель требований FURPS+ . . . . .	10
4.2 Доменная модель . . . . .	11
4.3 Риски . . . . .	12
4.4 Git rules . . . . .	12
4.4.1 Определение . . . . .	12
4.4.2 Commands . . . . .	13
4.5 SVN rules . . . . .	13
4.5.1 Определение . . . . .	13
4.5.2 Основной цикл разработчика . . . . .	14
4.6 Классификация проблемных ситуаций . . . . .	14
4.7 Use-Case . . . . .	14
4.8 Методологии разработки . . . . .	15
4.8.1 Водопадная (Каскадная) . . . . .	15
4.8.2 Методология Ройса . . . . .	15
4.8.3 Традиционная V-chart model 1977 . . . . .	15
4.8.4 Многопроходная модель . . . . .	16
4.8.5 Модель прототипирования . . . . .	16
4.8.6 Spiral model . . . . .	16
4.8.7 RAD методология . . . . .	16
4.8.8 *UP методологии . . . . .	16

# 1 Перепись

1.1 С помощью ИС «Санта-клаус» детям дарят подарки за хорошее поведение. Разработать модель требований, Use-Case модель и доменную модель.

## 1.1.1 Модель требований FURPS+

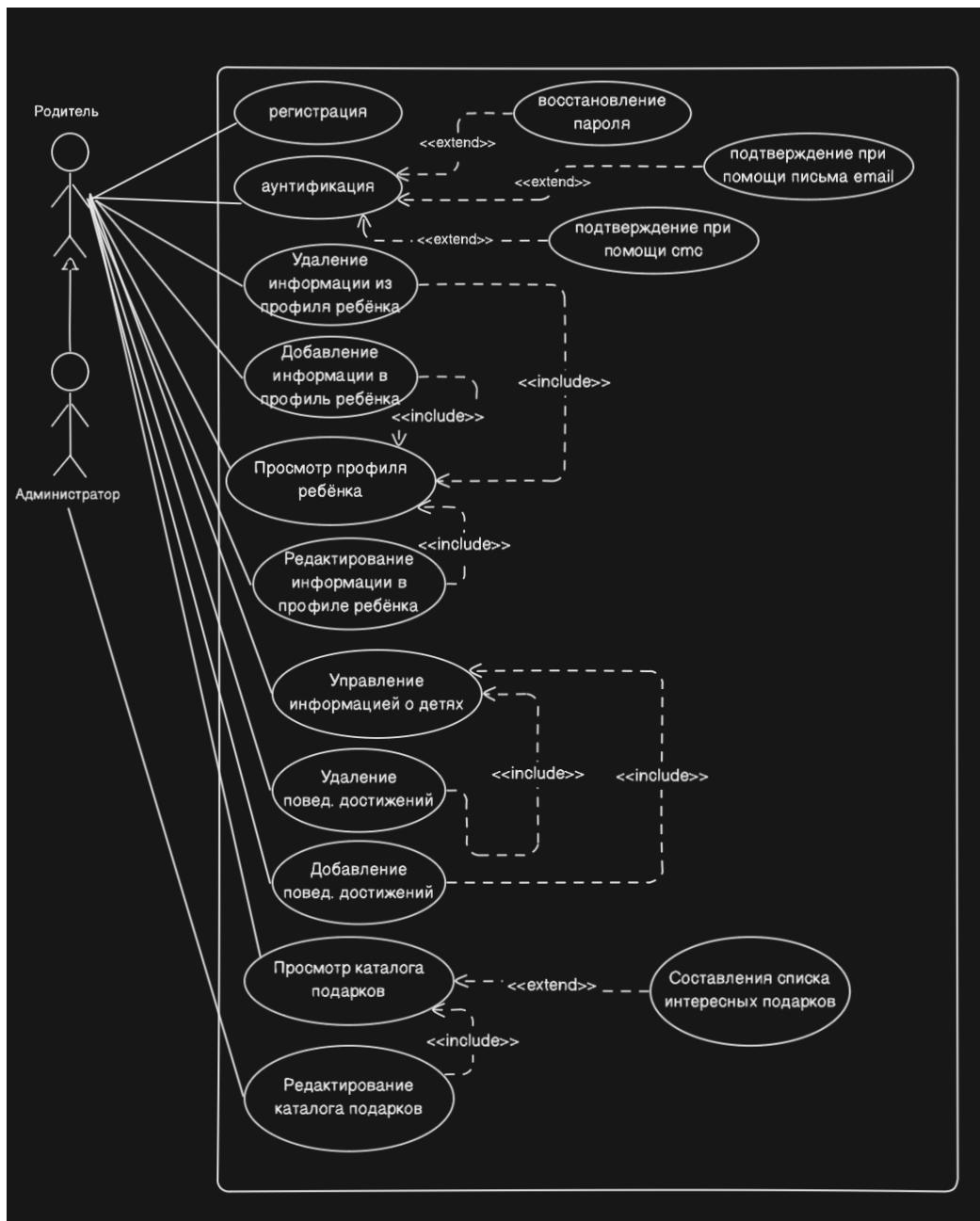
Функциональные требования

- SEC0 - Система должна обеспечивать регистрацию пользователей с помощью имени пользователя и пароля.
- SEC1 - Система должна поддерживать аутентификацию пользователей с помощью имени пользователя и пароля, также может использоваться подтверждения с помощью СМС или email письма.
- FR0 - Система должна поддерживать добавления, удаления, редактирование информации (не качества детей) о детях.
- FR1 - Система должна обеспечивать возможность отмечать, удалять поведенческие достижения их детей.
- FR2 - Система должна поддерживать просмотр каталога подарков.
- FR3 - Система должна поддерживать возможность составления списка интересных подарков для ребёнка.
- FR4 - Система должна предоставлять возможность восстановления забытого пароля.
- FR5 - Система должна поддерживать редактирование каталога подарков для админов.

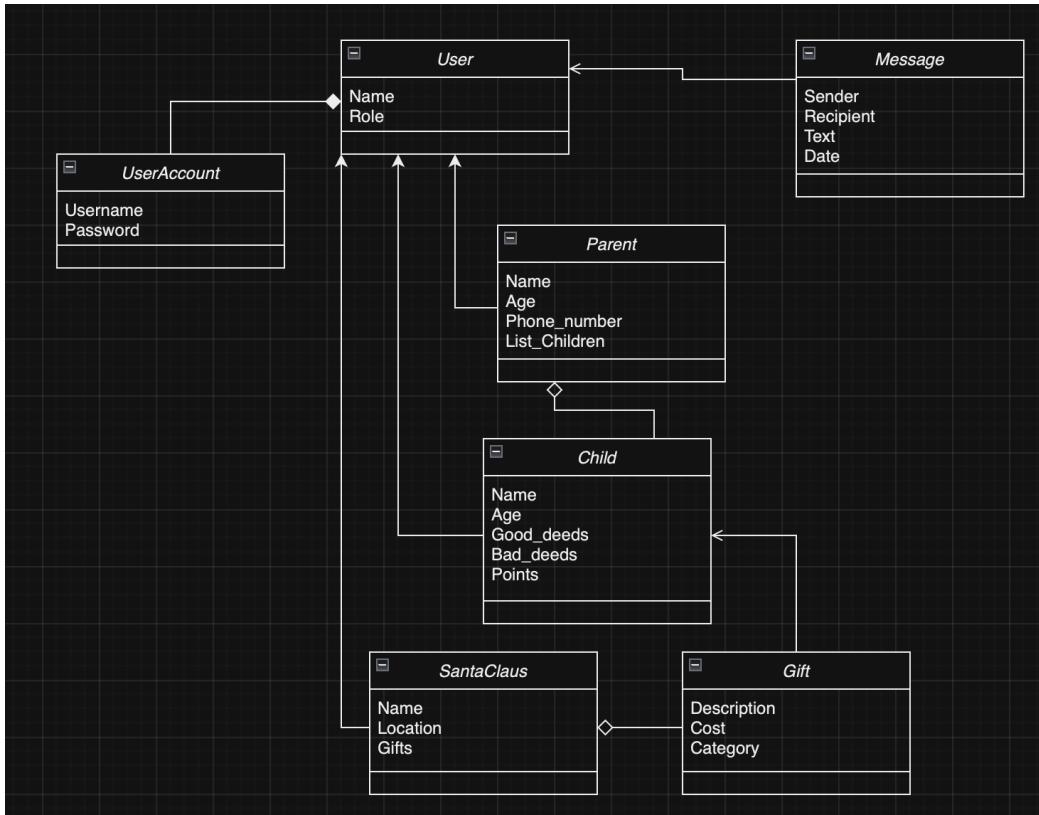
Нефункциональные требования

- USA0 - Система должна обеспечивать адаптивный дизайн для различных устройств.
- RELI0 - Система должна быть доступна 99.9% времени, с автоматическим восстановлением после сбоев.
- PERF0 - Система должна обрабатывать тысячи запросов одновременно без существенных задержек.
- SUPP0 - Система должна легко масштабироваться для поддержки увеличения числа пользователей.

### 1.1.2 Use-case диаграмма



### 1.1.3 Доменная модель



## 1.2 Модель разработки Disciplined Agile Delivery. «+» и «-» в сравнении с другими моделями

### 1.2.1 Disciplined Agile Delivery

Модель разработки разработанный Скоттом Амблером, который был вдохновлён системой деления на фазы (начало, построение и внедрение) и подходы как в RUP, но при этом основной цикл разработки построен на базе гибких методов, включая Scrum. Данная методология является относительно новой на рынке и только развивается. Так же DAD рассматривает процессы, которые выходят за его собственный процесс разработки (управление архитектурой и повторным использованием кода, управление персоналом, служба поддержки и текущих операций компании-разработчика, управление портфолио компетенций, непрерывное улучшение процессов разработки и вспомогательных процессов, и многое другое).

### 1.2.2 +

Одним из огромных плюсов (скорее даже это причина появления DAD) - на рынке требовалась методология, которая будет комфортна людям, которые, можно сказать, строят марсоход. Они не нуждаются в очень гибких методологиях, так как эти методологии не могут легко масштабироваться на большие команды разработчиков. Ещё плюсом является то, что можно комбинировать Agile и не Agile подходы в зависимости от требований проекта.

### 1.2.3 -

Минусы: сложность из-за большого количества инструментов и практик, отсюда ватикает следующий минус: высокий уровень компетенций команды для верного построения процессов.

## 1.3 Разработать тестовые сценарии (положительный и отрицательный) для следующего сценария

-Эльфы получают количество добрых дел у ребенка (K) по его имени и вводят его в систему;

-Если  $K \geq 10$ , то система назначает ему подарок

-Если  $K < 10$ , то система назначает ему уголек

-Эльф передает назначение на утверждение Санта-Клаусу

Оформить в виде таблицы тестовых случаев (Начальное состояние, ввод пользователя, вывод системы, конечное состояние)

Тест 1:

Начальное состояние: Готово и ожидает получение данных

Ввод пользователя: Иван, 10

Вывод системы: подарок

Конечное состояние: запрос на утверждение у Санта-Клауса

Тест 2:

Начальное состояние: Готово и ожидает получение данных

Ввод пользователя: Иван, 9

Вывод системы: уголёк

Конечное состояние: запрос на утверждение у Санта-Клауса

Тест 3:

Начальное состояние: Готово и ожидает получение данных

Ввод пользователя: Иван, -1

Вывод системы: неверный ввод данных

Конечное состояние: Готово и ожидает получение данных (Начальное состояние)

Тест 4:

Начальное состояние: Готово и ожидает получение данных

Ввод пользователя: Иван, "двадцать один"

Вывод системы: неверный ввод данных

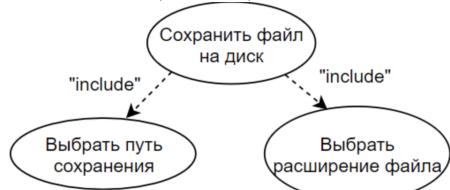
Конечное состояние: Готово и ожидает получение данных (Начальное состояние)

## 2 Вариант - 1

### 2.1 Use-Case

РАМКА БЕЗ АКТОРОВ!

Включение (Include) — обязательная, неотъемлемая связь между use-кейсами.



Расширение (Extend) — необязательное отношение. Если use-кейс не является обязательным, то актор может выбирать.



Отношение обобщения



Отношение ассоциации



Покупатель

## 2.2 Модель Ройса

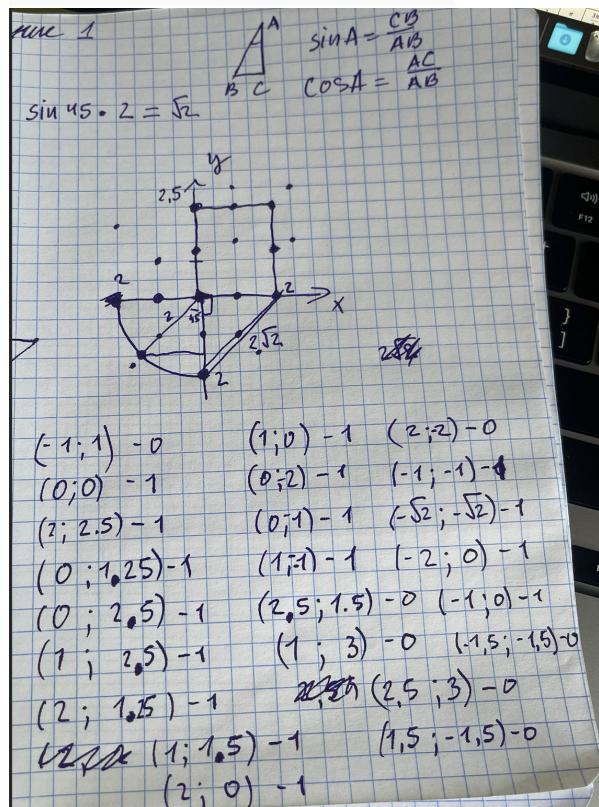
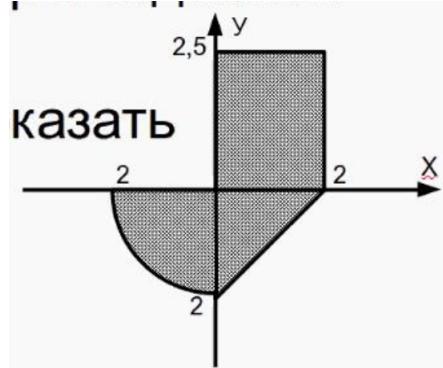
Если кратко, то он предложил разбиение на следующие шаги:

- Предварительный дизайн
- Документирование дизайна
- Just do it nike tw
- Тестирование
- Подключение пользователя

Подробнее

- Предварительный дизайн - занимаются только дизайнеры, цель: спроектировать, определить и создать модели обработки данных, даже если они будут требовать переделок. Разработать док - обзор будущей системы
- Документирование дизайна, которое включает: требования к системе, спецификация предварительного дизайна, спецификация дизайна интерфейсов, финальные спецификации дизайна системы, план тестирования, инструкция по использованию.
- Just do it nike tw - Предлагается запустить тестовую разработку параллельно основному процессу и взять это за каскадёра с сокращённым временем разработки, который позволит подтвердить основные характеристики ПО или найти ошибки.
- Тестирование - наиболее рискованная и дорогостоящая фаза, а также последний шанс для выбора альтернатив. Также при планировании тестирования исключается дизайнер и осмотр всех логических путей проводит другое лицо для инспекции кода. После исправления ошибок нужно провести checkout (тестирование)
- Подключение пользователя - это должно произойти на ранних этапах перед финальной поставкой продукта. Необходимо 3 точки: опыт, оценка и подтверждение пользователем - предварительный, критический и финальный просмотр.

## 2.3 Тестовое покрытие с использованием анализа эквивалентности



## 2.4 Ресурсные риски

Какие из перечисленных ниже рисков являются примерами ресурсных рисков? (перечислить номера вариантов ответа)

1. Риск поломки оборудования разработчиков.
2. Риск возникновения конфликта между менеджером исполнителя и заказчиком.
3. Риск отсутствия необходимых кадров при переносе производства в другой город.
4. Риск отсутствия спроса на произведенную продукцию.
5. Риск запрета продажи автомобилей с ДВС для снижения негативного экологического воздействия на планету

Ответ:

1. Риск поломки оборудования разработчиков.
3. Риск отсутствия необходимых кадров при переносе производства в другой город.

## 2.5 Топ - 3 рисков по экспозиции

- Определите топ-3 рисков по экспозиции? Риски сортировать по уменьшению.

Номер риска	Название	Вероятн ость, %	Стоимость, млн руб.
1	Риск закрытия программы полётов на Марс на государственном уровне.	10%	20'000
2	Риск возникновения пылевой бури в атмосфере Марса в период посадки	25%	100'000
3	Риск использования в конструкции корабля неэффективных двигателей	20%	100'000
4	Риск нехватки запасов продовольствия на время полёта	15%	10'000
5	Риск выбора неудачного стартового окна для полёта	25%	20'000

Вероятность натсупления \* потери = экспозиция

- $0.1 * 20000 = 2000$
- $0.25 * 100000 = 25000$
- $0.2 * 100000 = 20000$
- $0.15 * 10000 = 1500$
- $0.25 * 20000 = 5000$

Ответ: 235

## 2.6 Git pull

Какие действия выполняет команда git pull

Ответ: скачать и применить к своему локальному репозиторию последнюю версию удаленного репозитория

## 2.7 Git commit

Приведите команды, которые приведут к записи в репозиторий проекта нового коммита, содержащего файлы A,C,E

git reset HEAD D  
или git restore --staged D  
git add C  
git commit -m "commit0"

svn revert D  
svn add C  
svn commit -m

## 2.8 Классификация проблемных ситуаций

К какой категории классификации проблемных ситуаций жизненного цикла ПО с точки зрения тестирования относится приведённый ниже пример?

"Программист 3 категории Харитонов Фотий Аркадьевич, придя утром на работу в плохом настроении, заявил своим коллегам, что он их всех ненавидит, и случайно добавил в свою программу некорректно работающий код".

- Mistake (Error) — ошибка, просчёт (человека).
- Fault — дефект, изъян (ПО в результате ошибки). Неверный шаг в алгоритме. Следствие ошибки.
- Failure — неисправность, отказ, сбой (внешнее проявление дефекта). Крах, падение программы.
- Error — невозможность выполнить задачу с использованием программы.

- BUG — используется неформально (они не подходят друг к другу). Может обозначать: дефект, отказ, невозможность выполнить задачу.

Ответ: mistake

## 2.9 Скрипт сборки Gradle

Приведите скрипт сборки для Gradle, который устанавливает следующую последовательность целей сборки COMP, RESOURCE, LIB, BUILD, DEPLOY. Цели печатают на консоли только свое имя.

Gradle:

```
task COMP { doLast { println("COMP") } }
task RESOURCE { doLast { println("RESOURCE") } }
task LIB { doLast { println("LIB") } }
task BUILD { doLast { println("BUILD") } }
task DEPLOY { doLast { println("DEPLOY") } }
build.dependsOn COMP, RESOURCE, LIB, BUILD, DEPLOY
```

Make:

```
.PHONY: all COMP RESOURCE LIB BUILD DEPLOY
all: COMP RESOURCE LIB BUILD DEPLOY
COMP: @echo "COMP"
RESOURCE: @echo "RESOURCE"
LIB: @echo "LIB"
BUILD: @echo "BUILD"
DEPLOY: @echo "DEPLOY"
```

Ant:

```
<project name="example build script">
<target name="COMP">
<echo message="COMP"/>
</target>
<target name="RESOURCE">
<echo message="RESOURCE"/>
</target>
<target name="LIB">
<echo message="LIB"/>
</target>
<target name="BUILD">
<echo message="BUILD"/>
</target>
<target name="DEPLOY">
<echo message="DEPLOY"/>
</target>
</project>
```

## 2.10 Тестовое покрытие

Программисты Банка "Ваше Богатство" определяет тестовое покрытие для функции вычисления размера процентов на остаток по счету клиента. При наличии на счете от 50 000 до 100 000 руб. ежемесячно начисляется сумма эквивалентная 5.5% годовых; если на счете от 100 001 до 500 000 то 4%; если больше 500000 то 0.01%. При наличии на счете меньше 50 000 руб., проценты не начисляются. Сколько эквивалентных участков должно содержать тестовое покрытие, если в функцию могут поступать только корректные числовые данные?

Ответ: 4

## 2.11 Простой системы

Разработчик информационной системы заявил, что готовность его системы составляет 99.92%. Какое максимальное целое количество минут в невисокосный год данная система может простоять?

$$365 \cdot 24 \cdot 60 \cdot (1 - 99.92)/100 = 420.48 = 420$$

## 2.12 Скорость работы системы

Программа вычисляет значение по алгоритму, обрабатывая данные на уровне процессора (1 команда = 1 нс), уровне кэш-памяти второго уровня (20 нс), оперативной памяти (100 нс), ssd (50мкс). Для вычисления значения алгоритма используются 1000 команд на уровне процессора, 100 обращений к памяти, из которых 90% кешируются в кэш-памяти второго уровня, а также 4 обращения к диску. На сколько процентов увеличится скорость работы алгоритма, если программист смог уменьшить количество обращений к SSD до одного раза? Ответ выразить в процентах.

Ранее:  $1000 \cdot 1 + (100/100 \cdot (100 - 90)) \cdot 100 + (100/100 \cdot 90) \cdot 20 + 4 \cdot 50 \cdot 10^3 = 203800$

Будет:  $1000 \cdot 1 + (100/100 \cdot (100 - 90)) \cdot 100 + (100/100 \cdot 90) \cdot 20 + 1 \cdot 50 \cdot 10^3 = 53800$

$$\frac{203800 - 53800}{203800} \cdot 100 = 73\%$$

$$\frac{53800}{203800} \cdot 100 = 378\%$$

## 3 Вариант - 2

### 3.1 Доменная модель

### 3.2 Чему уделялось внимание в спиральной модели

Применялась для разработки больших программных продуктов. После постановки целей и выявления альтернативных решений, производился анализ рисков для минимизации числа действий для разработки ПО. На каждой итерации модели перед созданием прототипа происходит сверка с картой рисков. (При этом анализировались не только технические риски, но и политические и тд) Так же в данной модели все изменения, которые вносятся в проект, являются неотъемлемой частью разработки. Любые действия с ними зависят от тех рисков, которые они несут для процесса разработки.

## 4 Термины

### 4.1 Модель требований FURPS+

Подробное описание того что должно быть реализовано системой, но при этом не должно описывать, как его нужно реализовать. Включает в себя функциональные требования, нефункциональные.

Функциональные определяют, что система должна делать: наборы функциональных требований (FR + номер), возможности ПО (CAP + номер), требования к безопасности (SEC + номер). Наборы - набор свойств продукты необходимый для выполнения конкретной деятельности (сис должна обеспечивать ввод, модификацию и удаление данных о клиенте). Требования к безопасности - метод аутентификации, список ролей, шифрование, хранение данных в защищённых источниках (сис должна обеспечивать двухфакторную аутентификацию пользователей с помощью имени пользователя и пароля и подтверждения с помощью СМС.).

Нефункциональные:

Usability - учёт особенностей пользователя (быстрота ответа в интервале), эстетические требования (цвет, расстояния между элементами), согласованность пользовательского интерфейса, согласованность пользовательского интерфейса, требования к справочной подсистеме, требования к пользовательской документации, требования к учебным материалам.

Reliability - частота и обработка заказов, способность системы восстанавливать продуктивное функционирование, предсказуемость поведения, точность, среднее время между отказами. Требования к надёжности, которые предназначены для способности ПО безотказно функционировать. В требованиях обычно указывается допустимое число отказов и сбоев за определённый промежуток времени. Способность системы восстанавливать продуктивное функционирование в течение заданного времени. Требованием является accuracy - точность, например, проведения вычислений. Коэффициент готовности системы — отношение времени исправной работы к сумме времён исправной работы и вынужденных простоев объекта, взятых за один и тот же календарный срок.

Performance - скорость решения задач, эффективность, готовность системы к решению задач, пропускная способность, время отклика, время восстановления, использование системных ресурсов. Требованием является скорость решения вычислительных задач. Также скорость важна в длительных инженерных расчетах, когда необходимо выполнить, например, моделирование за разумное для человека время. Требования к эффективности фиксируют процент времени, которое тратится на выполнение полезных задач, по отношению к времени на выполнение общесистемных. Требованием к производительности является готовность (availability) быстро начать выполнение задачи. Какой объём данных или запросов система может обработать за единицу времени. Для большой реактивности придется пожертвовать пропускной способностью.

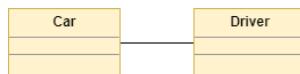
Supportability - способность системы к расширению и масштабированию и выполнению большего объема обработки данных. Адаптируемость под конкретные задачи, поддерживаемость. Требования к совместимости позволяют использовать различные операционные системы, версии продуктов, браузеров и пр. совместно с разработанным ПО. Отдельно выделяются системные требования и минимальные требования к установке системы, например, объём ОЗУ, количество и частота процессоров и пр.

## 4.2 Доменная модель

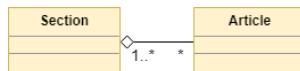
Доменная модель — это концептуальная модель предметной области, которая отображает ключевые сущности, их атрибуты и взаимосвязи между ними, а также основные правила бизнес-логики. Эта модель помогает разработчикам и всем участникам проекта лучше понять структуру и функционирование системы, на которой они работают. Основной сущностью является класс, он включает только атрибуты без типов данных и без методов. Первая часть содержит имя класса, вторая содержит атрибуты.

Типы отношений:

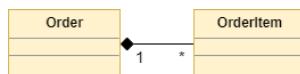
Ассоциация - это основные отношения между двумя сущностями. Эти сущности могут существовать независимо друг от друга. Мы рисуем его как простую сплошную линию. Первая сущность имеет ссылку на другую, а другая - на первую. Мы можем изменить это поведение, добавив простую стрелку, определяющую направление отношений. Только экземпляр, из которого стрелки хранят ссылку на другую сущность в этих случаях.



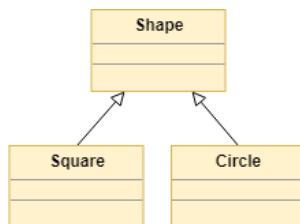
Агрегация - связь между классом и его частями. Бриллиант рисуется в классе, представляющем целое (например, раздел article - статья).



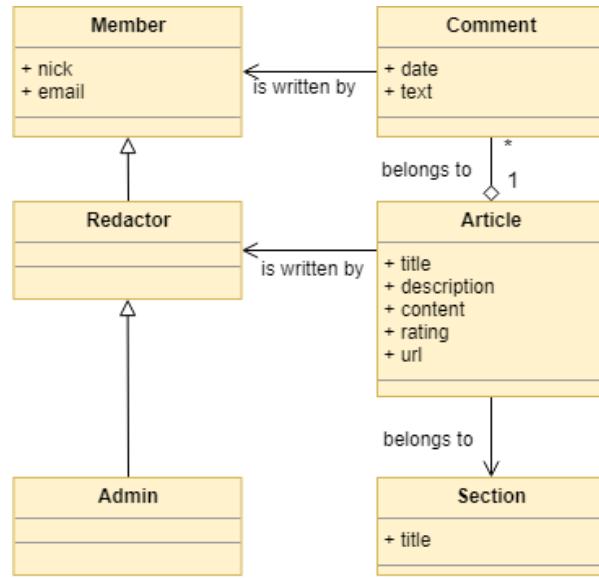
Композиция - тоже агрегирование, но часть не имеет смысла без главной сущности, при удалении сущности, удаляется и его часть



Обобщение - наследование. Одна сущность наследует свойства и поведение от другой сущности.



Пример:



© ICT.social

### 4.3 Риски

- Прямые и непрямые - можем контролировать или управлять или нет. Прямыми рисками можно в явном виде управлять: воздействовать на них, уменьшать их вероятность, реагировать на них. Непрямые риски возникают по внешним причинам и не поддаются управлению, можно только принять на себя их последствия. Отсюда существуют различные методы работы с прямыми и непрямыми рисками.
- Ресурсные - организационные, финансовые, люди, время. связаны с недостатком у компании времени, людей, денег, оборудования. Ути риски имеют отношение к особенностям и специфике конкретной организации, разрабатывающей проект. Данные риски являются управляемыми, так как возможно изменение выделяемых ресурсов.
- Бизнес-риски - конкуренция, подрядчики, убыточность решения. появляются из-за взаимодействия с другими организациями и рынком в целом. Они определены конкуренцией, взаимодействием с подрядчиками или потенциальной убыточностью решения, т.е. отсутствием в дальнейшем прибыли от реализации и внедрения ПО. Управление бизнес- рисками сложно поддается управлению со стороны разработчиков.
- Технические риски - границы проекта, технологические, внешние зависимости. находятся в пределах компетенции разработчиков и поэтому являются ими управляемыми. Примеры технических рисков - отсутствие у разработчиков компетенции в применяемых технологиях.
- Форс-мажор. В отличие от политических рисков, повлиять на форс-мажор нельзя, и предугадать его тоже. К таким рискам относятся стихийные бедствия, изменение законодательства и т.п.
- Политические риски - связаны с изменением сфер влияния внутри компании-заказчика. Например, с появлением нового менеджера могут измениться условия сделки. Это возможно предвидеть, но не всегда. Противодействием может быть упреждающая реакция на возможные изменения в компании (например, установление контактов с вероятным новым менеджером).

### 4.4 Git rules

#### 4.4.1 Определение

Git - децентрализованная система. У каждого разработчика есть локальная копия репозитория, в котором хранится всё о проекте целиком и доступна вся история изменений, как от самого разработчика, так и история изменений других разработчиков. Каждый локальный репозиторий имеет свой URL, по которому он доступен, а другие разработчики могут обращаться к нему при помощи алиасов. Благодаря такому подходу можно гибко подбирать вариант работы:

централизованный рабочий процесс (главная ветка и с ней синхронизируются все разработчики);

рабочий процесс с менеджером по интеграции (каждый разработчик меняет свой канонический репозиторий, открытый только для чтения, в локальной копии и потом запрашивает менеджера загрузить его в канонический репозиторий);

рабочий процесс с диктатором и лейтенантами (лейтенанты - интеграционные менеджеры, которые отвечают за отдельный части репозитория. У всех лейтенантов есть ещё один интеграционный менеджер - доброжелательный диктатор, репозиторий которого выступает как эталонный, откуда все участники проекта получают изменения)

#### 4.4.2 Commands

- git pull - скачать и применить к своему локальному репозиторию последнюю версию удаленного репозитория
- git clone - клонировать удаленный репозиторий в новую директорию на локальной машине
- git init - создать новый пустой репозиторий Git или реинициализировать существующий
- git add - добавляет изменённые файлы к последующему коммиту, помещая их в Stage Area.
- git commit - фиксирует изменения в текущей ветке. Опция -t задает сообщение коммита, которое будет показываться пользователю
- git push - отправить коммиты из локального репозитория в удаленный репозиторий
- git status - показывает статус текущих состояний файлов в файловой системе и информацию о ветви, в которой производится редактирование.
- git log - показывает журнал коммитов, а опция -graph выводит в графическом виде ветви, в которых производились данные изменения.
- git branch — показывает ветви.
- git checkout — переключает разработчика между ветвями.
- git merge — объединяет несколько ветвей в текущую ветвь.
- git diff — покажет все изменения, которые были сделаны относительно последних зафиксированных изменений.
- git reset — отменить изменения, вернув файлы в рабочем каталоге к состоянию последнего коммита
- git fetch — скачать изменения из удаленного репозитория, но не применять их к локальному репозиторию
- git rebase — перенести или перепроиграть коммиты на другую базовую точку
- git cherry-pick — применить изменения из указанного коммита в текущую ветку
- git reset --hard HEAD сбросит все изменения, которые были сделаны в текущем локальном репозитории.

### 4.5 SVN rules

#### 4.5.1 Определение

SVN - централизованная система контроля версий. Каждая фиксация изменений - ревизия. Организация файлов: есть каталоги проектов, внутри которых размещены директории, предназначенные для организации работы с разными версиями проекта. В каталоге trunk происходит основной процесс разработки. Для хранения истории релизов можно копировать содержимое trunk в дополнительные каталоги trunk и tags.

- trunk - основная ветвь разработки
- branches - хранение модификаций продукта (Releases - значимые версии продукта, features - выполнение работ над версиями проекта без влияния на trunk, vendor - версии с модификациями сторонних библиотек)
- tag - функционально целостные изменения.

#### 4.5.2 Основной цикл разработчика

- svn checkout - первоначальное создание рабочей копии
- svn update - обновить рабочую копию
- svn (add, delete, copy, move, mkdir) - изменения в проекте
- svn diff, svn status - просмотр изменений
- svn revert - откат изменений
- svn update - для загрузки изменений
- svn commit - фиксация изменений
- svn merge - подгружает изменения из веток в рабочую копию, учитывая изменение структуры.

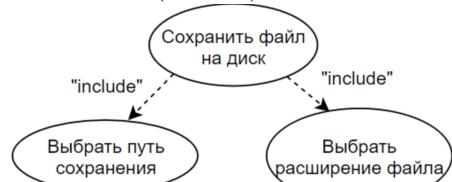
#### 4.6 Классификация проблемных ситуаций

- Mistake (Error) — ошибка, просчёт (человека).
- Fault — дефект, изъян (ПО в результате ошибки). Неверный шаг в алгоритме. Следствие ошибки.
- Failure — неисправность, отказ, сбой (внешнее проявление дефекта). Крах, падение программы.
- Error — невозможность выполнить задачу с использованием программы.
- BUG — используется неформально (они не подходят друг к другу). Может обозначать: дефект, отказ, невозможность выполнить задачу.

#### 4.7 Use-Case

РАМКА БЕЗ АКТОРОВ!

Включение (Include) — обязательная, неотъемлемая связь между use-кейсами.



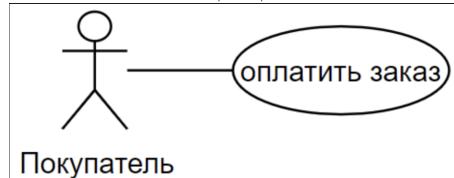
Расширение (Extend) — необязательное отношение. Если use-кейс не является обязательным, то актор может выбирать.



Отношение обобщения



## Отношение ассоциации



## 4.8 Методологии разработки

### 4.8.1 Водопадная (Каскадная)

Стандартные шаги разработки

- Определение системных требований
- Определение требований в ПО
- Анализ требований
- Проектирование программы
- Разработка кода
- Тестирование
- Введение ПО в эксплуатацию

Есть понятие итерации между фазами разработки, есть возможность отката к предыдущей фазе. На фазе тестирования можно обнаружить, что итоговые характеристики отличаются от заданных изначально, поэтому нужно менять либо требования, либо дизайн системы.

### 4.8.2 Методология Ройса

Ройса предложил разбиение на следующие шаги:

- Предварительный дизайн - занимаются только дизайнеры, цель: спроектировать, определить и создать модели обработки данных, даже если они будут требовать переделок. Разработать док - обзор будущей системы
- Документирование дизайна, которое включает: требования к системе, спецификация предварительного дизайна, спецификация дизайна интерфейсов, финальные спецификации дизайна системы, план тестирования, инструкция по использованию.
- Just do it nike tw - Предлагается запустить тестовую разработку параллельно основному процессу и взять это за каскадёра с сокращённым временем разработки, который позволит подтвердить основные характеристики ПО или найти ошибки.
- Тестирование - наиболее рискованная и дорогостоящая фаза, а также последний шанс для выбора альтернатив. Также при планировании тестирования исключается дизайнер и осмотр всех логических путей проводит другое лицо для инспекции кода. После исправления ошибок нужно провести checkout (тестирование)
- Подключение пользователя - это должно произойти на ранних этапах перед финальной поставкой продукта. Необходимо 3 точки: опыт, оценка и подтверждение пользователем - предварительный, критический и финальный просмотр.

### 4.8.3 Традиционная V-chart model 1977

Так как в то время уделялось внимание качеству ПО, то Ройсом была предложена идея. В основе V-chart model лежит такая же последовательность шагов как и в водопадной модели, но каждому уровню разработки положен свой уровень тестирования. Модульное, интеграционное и системное тестирование проводятся последовательно на основе критериев, которые заданы соответствующими уровнями разработки. Последний этап - приёмочное тестирование (проверка продукта на соответствие функциональным требованиям). Для тестирования нужно эталонное поведение, которое должно быть задано вне кода данной системы. Есть разделение на статическое и динамическое тестирование. Динамическое - компьютерное исполнение тестов, а статическое (может проверяться на ранней стадии проекта, что позволяет выявлять грубые ошибки в проекте на начальной стадии) проверяет артефакты без компьютерного исполнения.

#### **4.8.4 Многопроходная модель**

Учитывая сложность разработки проекта в один проход, было предложено разбить продукт по разным техническим и функциональным требованиям, а в последствии разрабатывать, реализовывать и интегрировать воедино в несколько проходов в виде отдельных сборок. (инкременты) Такой подход снижает стоимость изменений требований заказчика. Разработка - более управляема. Можно использовать уже частично разработанную систему. Современные Agile опираются на итеративный подход. Прогресс виднее заказчику. Минусы: т.к. система может устаревать, то нужно обновлять её, но для инкрементной модели нужна неизменяемая архитектура и API (так как тут разработка ведётся параллельно несколькими командами), которые определяются в самом начале построения проекта (поэтому всё тяжело и дорого). Документы по сборкам сложно поддерживать из-за быстрой скорости изменений. Ещё сложно заключать контракты на разработку, так как там нужно фиксировать часы и суммы на разработку этапов.

#### **4.8.5 Модель прототипирования**

В данной модели лежит идея разделения процесса разработки проекта на прототипы, каждый из которых уточняет архитектуру, опираясь на функциональные требования. В начале система похожа на каркас, куда будет наращиваться проект. Относится к эволюционному построению ПО. Сначала планируется вся итерация, где проводится анализ текущих требований и подходов их реализации. Создаётся бд и интерфейс для пользователя, разрабатывается функционал и проверяется вместе с пользователем системы. Проводятся тесты. Если пользователю всё нравится, то происходит переход к конечной версии разработки ПО. Если пользователь не доволен, то разрабатывается ещё один прототип с новым интерфейсом и с бд. Прототипы создаются пока пользователь не протестирует все характеристики. В современных методах разработки променяются мокапы для создания простеньких визуальных моделей. Или применяют UX - модели, которые более сложные, но могут дать понять пользователю удобство пользования.

#### **4.8.6 Spiral model**

Применялась для разработки больших программных продуктов. Каждый виток спирали представляет собой одну фазу разработки очередной версии ПО. Сначала производится постановка целей, ограничений для каждой итерации. После постановки целей и выявления альтернативных решений, производился анализ рисков для минимизации числа действий для разработки ПО. На каждой итерации модели перед созданием прототипа происходит сверка с картой рисков. (При этом анализировались не только технические риски, но и политические и тд). После анализа рисков происходит разработка и валидация ПО. В конце последней итерации повторяется часть V-образной модели, где происходит тестирование разработанного ПО. На последней части каждого витка происходит планирование следующей итерации. В зависимости от фазы планированию подлежат действия по разработке требований и жизненного цикла разработки, собственно сама разработка, а также действия по интеграции разработанных частей в единое целое и проведение тестирования. Так же в данной модели все изменения, которые вносятся в проект, являются неотъемлемой частью разработки. Любые действия с ними зависят от тех рисков, которые они несут для процесса разработки.

#### **4.8.7 RAD методология**

Методология базируется на идеи увеличения участия пользователя в процессе разработки, а также идеи об автоматизации процессов. Она состояла в том, что сначала производится проектирование, а затем при помощи средств автоматизации собирается ПО. При этом пользователь принимает непосредственное участие в процессе разработки при помощи средств автоматизации (например создавать простые функции и интерфейсы). Минусы: разработка может усложняться за счёт автоматизации и внедрения пользователя, также некоторые интерфейсы могут быть не сильно продуманы. Плюсы: реализуется бизнес-функция "Нужно разработать уже вчера".

#### **4.8.8 \*UP методологии**

Процесс представляет собой разработку в виде инкрементно-эволюционного подхода. Процесс разработки разбит на фазы. Также присутствуют дисциплины - набор правил и указаний (необходимы для определённых задач, таких как тестирование или определение требований). Все дисциплины сделаны для того чтобы организовать разработчиков, дать им подходы, чтобы каждая роль процесса разработки могла выполнять необходимые, требуемые от неё действия. Всего фаз 4: Inception (начало), Elaboration (проектирование), Construction (создание продукта), Transition (внедрение на стороне заказчика) В рамках итеративного подхода фазы могут делиться на итерации.