

Вопросы

- 1) С помощью ИС «Санта-Клаус» детям дарят подарки за хорошее поведение. Разработать модель требований, Use-Case модель и доменную модель.
- 2) Модель разработки Disciplined Agile Delivery. «+» и «-» в сравнении с другими моделями
- 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)

1)

Модель требований:

- 1) Система должна иметь базу данных детей и их достижений.
- 2) Система должна предоставлять родителям возможность зарегистрировать своих детей в системе.
- 3) Система должна хранить информацию о поведении каждого ребенка.
- 4) Система должна иметь каталог подарков и их описание.
- 5) Система должна иметь возможность отправлять уведомление родителям о достижениях и подарках для их детей.
- 6) Система должна иметь возможность автоматически распределять подарки в соответствии с достижениями ребенка.

Use-case модель:

Регистрация ребенка:

Родитель заполняет форму регистрации ребенка.

Система добавляет ребенка в базу данных.

Добавление достижения ребенка:

Родитель добавляет достижение ребенка.

Система обновляет информацию о поведении ребенка в базе данных.

Получение подарка:

Ребенок, достигший определенных результатов, получает уведомление о подарке.

Родитель получает уведомление о подарке для своего ребенка.

Просмотр достижений:

Родитель может просмотреть достижения своего ребенка в системе.

Доменная модель:

Ребенок: имя, фамилия, возраст, достижения.

Родитель: имя, фамилия, контактные данные.

Достижение: описание, число баллов.

Подарок: название, описание, количество баллов.

2)

Disciplined Agile Delivery (DAD) - это Agile методология, основанная на принципах Agile Manifesto и на опыте применения Agile подходов в различных проектах. DAD рассматривает не только процесс разработки, но и весь жизненный цикл проекта, включая планирование, архитектуру, тестирование и развертывание.

Плюсы модели DAD:

- Предоставляет гибкость и адаптивность для различных типов проектов и организаций
- Хорошо подходит для больших проектов.
- Учитывает весь жизненный цикл проекта, включая планирование, архитектуру, тестирование и развертывание.
- Позволяет совмещать Agile и не-Agile подходы в зависимости от требований проекта.
- Предлагает много инструментов и практик для улучшения качества продукта.

Минусы модели DAD:

- У DAD есть высокий уровень сложности из-за большого количества различных процессов, практик и инструментов.
- Эта модель требует высокого уровня компетенции команды для правильной реализации всех процессов и практик.

3)

Начальное состояние	Ввод пользователя	Вывод системы	Конечное состояние
Готов (Количество добрых дел ребенка = 12)	12	Назначен подарок	Ожидание утверждения от Санта-Клауса
Ожидание утверждения от Санта-Клауса	Санта-Клаус дает подтверждение	Подтверждение Санта-Клауса	Готов

Начальное состояние	Ввод пользователя	Вывод системы	Конечное состояние
Готов (Количество добрых дел ребенка = 0)	0	Назначен уголек	Ожидание утверждения от Санта-Клауса
Ожидание утверждения от Санта-Клауса	Санта-Клаус дает подтверждение	Подтверждение Санта-Клауса	Готов


Начальное состояние	Ввод пользователя	Вывод системы	Конечное состояние
Готов (Количество добрых дел ребенка = 9)	9	Назначен уголек	Ожидание утверждения от Санта-Клауса
Ожидание утверждения от Санта-Клауса	Санта-Клаус отклоняет	Отклонение Санта-Клауса	Готов

Начальное состояние	Ввод пользователя	Вывод системы	Конечное состояние
Готов (Количество добрых дел ребенка = -3)	-3	Неверный ввод данных	Готов (Начальное состояние)
Готов (Количество добрых дел ребенка = «десять»)	«десять»	Неверный ввод данных	Готов (Начальное состояние)

Вариант 1:

1 вариант

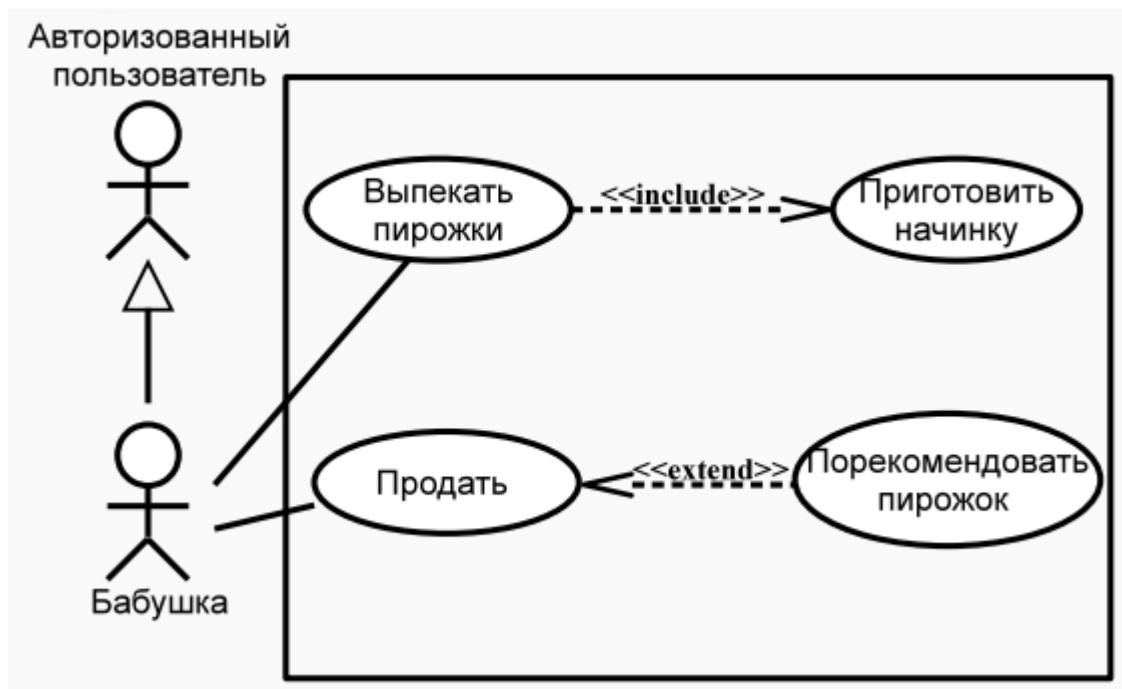
1. Рисунок. Use-case диаграмма
2. Что предложил Ройс в своей модели
3. Разработать тестовой покрытие (рисунок)
4. Ресурсные риски
5. Экспозиция рисков
6. git pull. Что выполняет?
7. Команды git (рисунок)
8. Классификация жизненных ситуаций цикла ПО
9. Gradle скрипт для сборки
10. Анализ эквивалентности
11. Отказоустойчивость системы
12. Скорость работы программы (Пирамида памяти)



Ответы

1. (7 6) Рисунок
2. (1 0.5) Предварительный дизайн. Документирование. «Do it twice». Тестирование. Подключение пользователя
3. (7 6) Таблица и рисунок с покрытием
4. (1 0.5) 1 и 3
5. (2 1) 2,3,5
6. (1 0.5) Скачать и применить к своему локальному репозиторию последнюю версию удаленного репозитория
7. (2 1) git reset HEAD D
git add C
git commit -m "Commit message"
8. (1 0.5) Mistake
9. (3 2) Текст скрипта сборки
10. (1 0.5) 4
11. (1 0.5) 420
12. (2 1) 378%

1) Пример Use-case диаграммы



2)

Первый шаг: Предварительный дизайн программы. В нем дизайнеру предлагается спроектировать и создать модели обработки данных, и разработать документ: обзор будущей программы. Благодаря этому дизайнер может убедиться, что требуемые характеристики могут быть реализованы. Выполняется между программными требованиями и анализом.

Второй шаг: документирование дизайна: требования к системе, спецификация предварительного дизайна, спецификация дизайна интерфейсов, финальные спецификации дизайна системы, план тестирования, инструкция по использованию.

Третий шаг: 'do it twice', тестовая разработка параллельно основному процессу и использование в качестве пилота для подтверждения или опровержения основных спецификаций ПО

Четвертый шаг: планирование, контроль и мониторинг тестирования. Тестирование - последняя точка, где может быть выбрана альтернатива. При планировании тестирования предлагается исключить дизайнера системы из процесса тестирования, провести "визуальную инспекцию" — повторный просмотр кода другим лицом, которое, не проводя глубокий анализ, отметит визуально заметные дефекты, протестировать каждый логический путь внутри программы (несмотря на то, что это труднореализуемо). После исправления большинства простых ошибок провести проверку (checkout) программы в необходимом тестовом окружении.

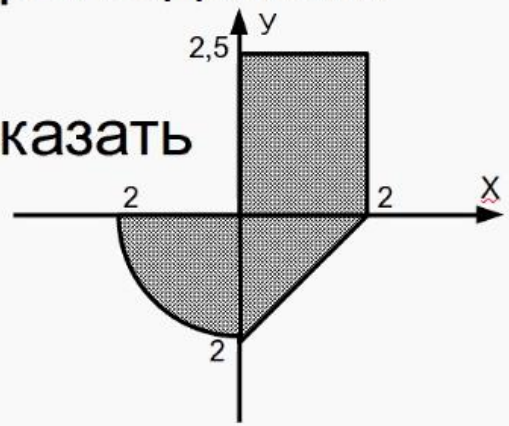
Пятый шаг: подключения пользователя на ранних этапах. В своей модели Ройс представил три точки, где необходим опыт, оценка и подтверждение пользователем — предварительный, критический и финальный просмотр.

3)



Вопрос №3 (7 мин)

- Разработать тестовое покрытие с использованием анализа эквивалентности для определения соответствия фигуры заданной
- Оформить в виде таблицы и показать точки на фигуре.



- Сделать таблицу в виде X, Y, result
- Внести туда точки, которые проверяли бы попадание и не попадание в каждой области (не забывать про пустую область и проверять граничные случаи)
- Нанести на фигуру

4)



Вопрос №4 (1 мин)

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

⊕₂

- 1- Ресурсный
- 2- Бизнес-риск
- 3- Ресурсный
- 4- Бизнес-риск
- 5- Политический

Ответ: 1, 3

5)



Вопрос №5 (2 мин)

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

Номер риска	Название	Вероятность, %	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

Введём понятие экспозиции риска (Risk Exposure), которая определяется произведением вероятности наступления риска и величины денежных потерь.

$$1- 0.1 * 20000 = 2000$$

$$2- 0.25 * 100000 = 25000$$

$$3- 0.2 * 100000 = 20000$$

$$4- 0.15 * 10000 = 1500$$

$$5- 0.25 * 20000 = 5000$$

Ответ: 2 3 5

6)



Вопрос №6 (1 мин.)

- Какие действия выполняет команда `"git pull"` ?

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

7)



Вопрос №7 (2 мин.)

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

```
$ git status
```

```
Changes to be committed:
```

```
modified:   A
```

```
new file:   D
```

```
new file:   E
```

```
Untracked files:
```

```
C
```

Ответ:

```
git reset HEAD D
```

```
git add C
```

```
git commit -m "I love mispi"
```



Вопрос №8 (1 мин.)

- 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).
- Failure — неисправность, отказ, сбой (внешнее проявление дефекта).
- Error — невозможность выполнить задачу вследствие отказа.
- BUG — используется неформально. Может обозначать: дефект, отказ, невозможность выполнить задачу.
 - Что-то другое или ничего не обозначать.

Ответ: Mistake



Вопрос №9 (3 мин.)

- Приведите скрип сборки для Gradle, который устанавливает следующую последовательность целей сборки COMP, RESOURCE, LIB, BUILD, DEPLOY. Цели печатают на констоли 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>
```

Maven:

```
<project>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-antrun-plugin</artifactId>
        <version>1.8</version>
        <executions>
          <execution>
            <id>print-targets</id>
            <phase>compile</phase>
            <goals>
              <goal>run</goal>
            </goals>
            <configuration>
              <tasks>
                <echo message="COMP" />
              </tasks>
            </configuration>
          </execution>
          <execution>
            <id>print-targets</id>
            <phase>compile</phase>
            <goals>
              <goal>run</goal>
            </goals>
            <configuration>
              <tasks>
                <echo message="RESOURCE" />
              </tasks>
            </configuration>
          </execution>
          <execution>
            <id>print-targets</id>
            <phase>compile</phase>
            <goals>
              <goal>run</goal>
            </goals>
            <configuration>
              <tasks>
                <echo message="LIB" />
              </tasks>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

```

        </execution>
        <execution>
            <id>print-targets</id>
            <phase>compile</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
                <tasks>
                    <echo message="BUILD" />
                </tasks>
            </configuration>
        </execution>
        <execution>
            <id>print-targets</id>
            <phase>compile</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
                <tasks>
                    <echo message="DEPLOY" />
                </tasks>
            </configuration>
        </execution>
    </executions>
</plugin>
</plugins>
</build>
</project>

```



ИТМО ВТ

Вопрос №10 (1 мин.)

- 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 500 000 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



Вопрос №11 (1 мин.)

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

Ответ: $365 * 24 * 60 * 0.08\% (0.0008) = 420.48 = 420$



Вопрос №12 (2 мин.)

- 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.

Было – $1000 + 10 * 100 + 90 * 20 + 4 * 50\,000$ нс (50 мкс) = 203800


Стало – $1000 + 10 * 100 + 90 * 20 + 1 * 50\,000$ нс (50 мкс) = 53800

Различается на 150 000нс => $203800 / 53800 * 100 = 378\%$

Ответ: 378%

2 вариант

1. Доменная модель
2. Чему уделялось внимание в спиральной модели
3. Разработать тестовое покрытие (рисунок)
- 4.
5. Экспозиция рисков
6. git rebase
- 7.
- 8.
9. makefile. Написать скрипт сборки
- 10.



Ответы

1. (7 6) Рисунок
2. (1 0.5) Риски
3. (7 6) Таблица и рисунок с покрытием
4. (1 0.5) 3,5
5. (2 1) 1,2,5
6. (1 0.5) Установить в качестве базового заданный коммит
7. (2 1) A C E
8. (1 0.5) 2,5
9. (3 2) Текст скрипта сборки
10. (1 0.5) release
11. (1 0.5) Failure
12. (2 1) 3