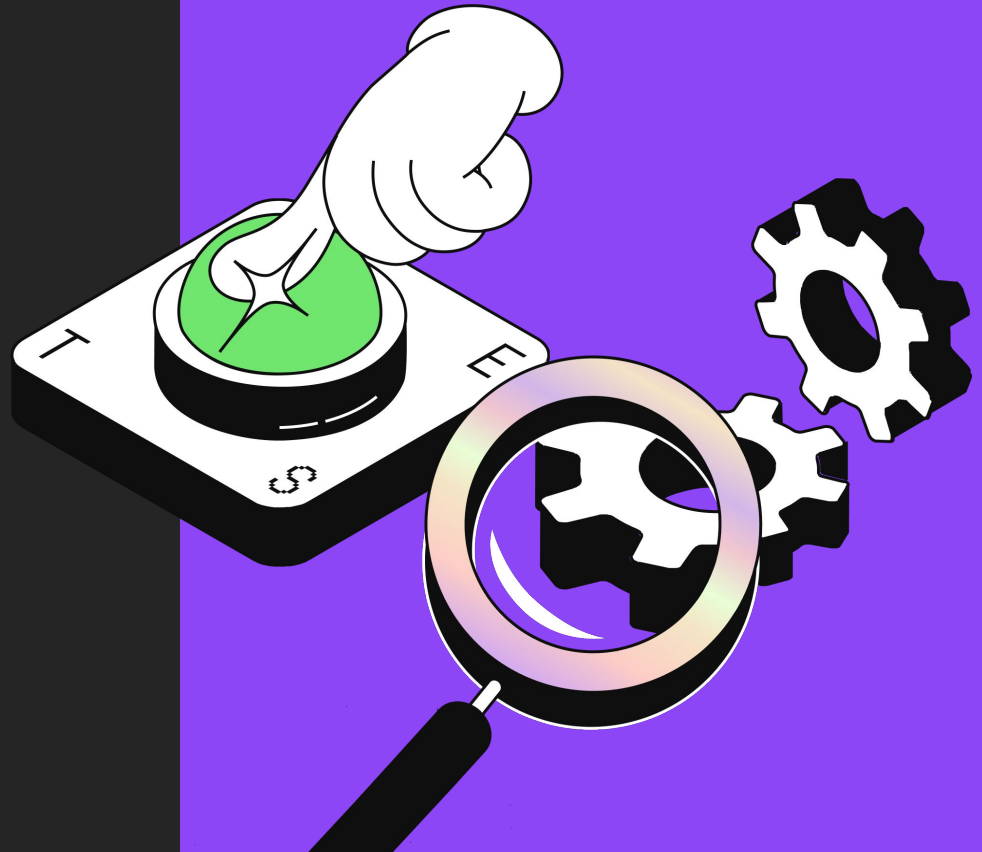
GeekBrains

Введение в юнит-тестирование

Урок 1
Цели и смысл тестирования



Знакомство
и содержание курса

Илья Смотров

Руководитель отдела тестирования

- Руковожу отделом тестирования в компании Аэро в подразделении консалтинга.
- Много лет занимаюсь автоматизацией тестирования и внедрения и выстраивания процессов автотестирования на стороне крупных российских и зарубежных заказчиков.
- Пишу тесты на Java, Java Script, Python, PHP и других языках.
- Спикер на курсе Unit тестирования в Geekbrains

План курса

1. Цели и смысл тестирования
2. Знакомство с тестовыми фреймворками
3. Качество тестов
4. Зависимости в тестах
5. Другие виды тестирования
6. Юнит-тестирование в других языках программирования

Содержание урока

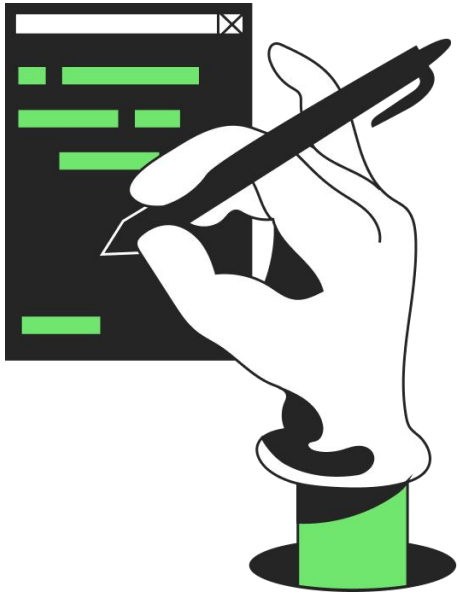
Что будет на уроке сегодня

- 📌 Что такое тестирование
- 📌 Цели тестирования
- 📌 Ошибки
- 📌 Принципы тестирования программного обеспечения
- 📌 Цикл разработки
- 📌 Написание простого теста
- 📌 Assert и AssertJ

Тестирование — это ...

Разберемся с терминами

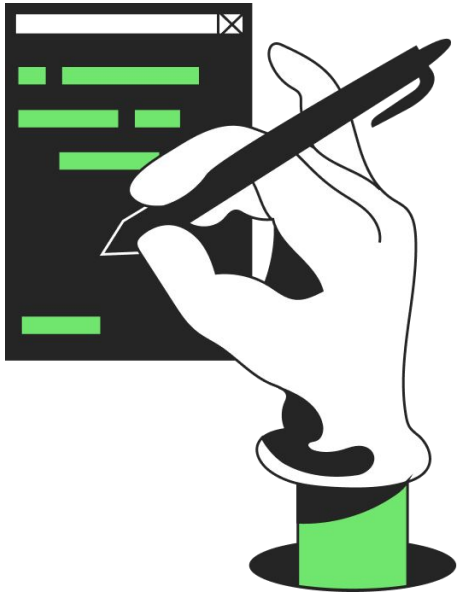
Тестирование — написание кода для проверки другого кода.



Разберемся с терминами

Тестирование — написание кода для проверки другого кода.

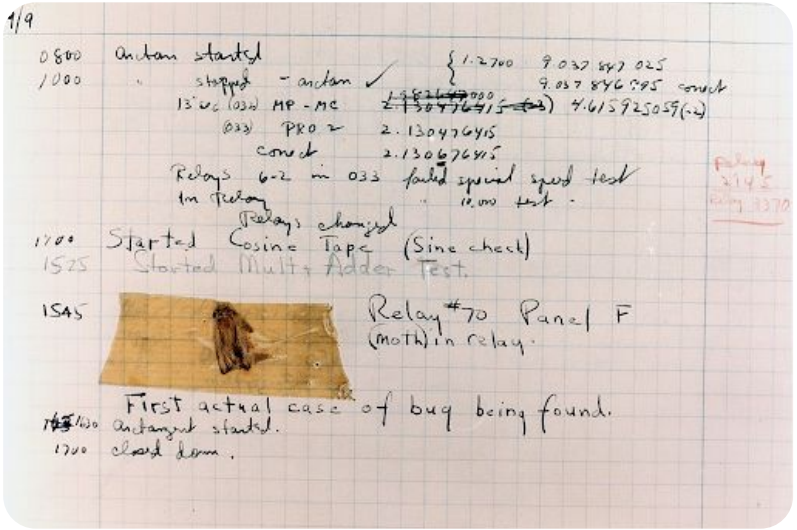
Тестирование — **процесс исследования,** испытания программного продукта, имеющий своей целью проверку соответствия между **реальным поведением программы и её ожидаемым поведением** на конечном наборе тестов, выбранных определенным образом (ISO/IEC TR 19759:2005)



Ошибки, виды ошибок

Баг

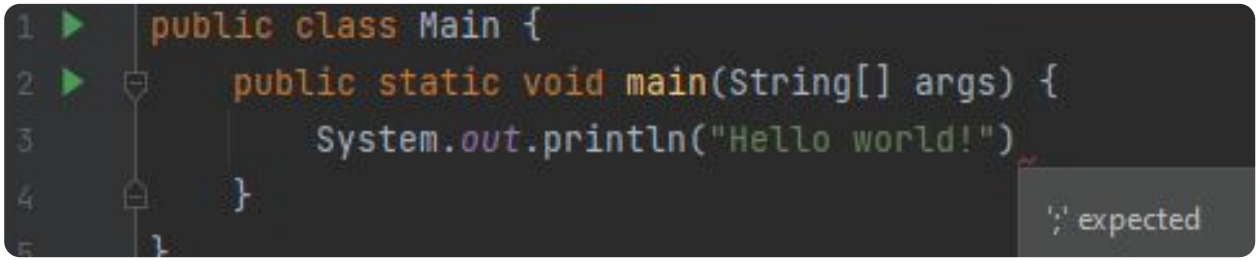
Программная ошибка (жарг. баг) — означает ошибку в программе или в системе, из-за которой программа выдает неожиданное поведение и, как следствие, результат.



Основные виды ошибок

Синтаксические

- ✔ Легко обнаружить и исправить на этапе компиляции (опечатки, скобки).
- ✔ Невозможна дальнейшая компиляция — ошибки устраняются сразу после обнаружения.



Основные виды ошибок

Логические ошибки

- ✔ Наиболее опасный вид ошибок: не обнаруживается на этапе компиляции, не прерывает выполнение программы (неправильный алгоритм, ошибки с использованием переменных).
- ✔ Специальные практики и строгие процедуры отладки, тестирование.

```java
public class Main {
    //  Типы ошибок
    public static void main(String[] args) {
        compareNumbers(2, 2); // Вызывается метод сравнения двух чисел
    }

    private static void compareNumbers(int a, int b) {
        if (a > b) {
            System.out.printf("%d more than %d", a,b);
        }
        if (a <= b) { // Допущена ошибка - знак <= вместо <
            System.out.printf("%d less than %d",a,b);
        }
    }
}
```

Основные виды ошибок

Ошибки выполнения

- ✔ Не обнаруживаются на этапе компиляции, но выявляются во время выполнения (Не оптимизированный алгоритм, исключения).
- ✔ Оптимизация, отладка, тестирование.

```
public static void main(String[] args) {
    int a = 10, b = 0;
    System.out.printf("Result: %d", a/b);
}
```

Вопрос

Цель тестирования в том чтобы проверить ...?

Напишите ваш ответ в комментариях к видеолекции.
Время на размышление - 1 минута.

Ответ

Цель тестирования в том, чтобы проверить соответствие между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов

Цена ошибки

Маринер-1. Авария на старте

Наиболее популярная версия причины потери связи с аппаратом — ошибка в ручном переводе математического символа в спецификации программы, а точнее — потерянная черта над символом.

Протон-М. Потеря спутников

Аварии, повлекшие потерю нескольких российских и иностранных спутников. Ошибки в формуле расчёта дозы заправки топлива.

Knight Capital. $440млн за 45 мин.

Компания потеряла почти полмиллиарда долларов меньше чем за час.
Причина — непроверенный софт для автоматизации торгов на бирже.

MISRA

MISRA (Motor Industry Software Reliability Association) — рекомендации по разработке электронных систем, связанных с безопасностью, встроенных систем управления, приложений с интенсивным использованием программного обеспечения и автономного программного обеспечения.

Кривая Боэма – рост затрат на поиск и устранение причин дефектов

Вопрос

Как вы думаете, какие еще преимущества дает тестирование, кроме уменьшения количества ошибок?

Напишите ваш ответ в комментариях к видеолекции.
Время на размышление - 1 минута.

Ответ

1

Проверка соответствия требованиям

Компаниям необходимо гарантировать работоспособность и качество предоставляемых сервисов для получения прибыли, поэтому тестирование это обязательный этап в современной разработке

2

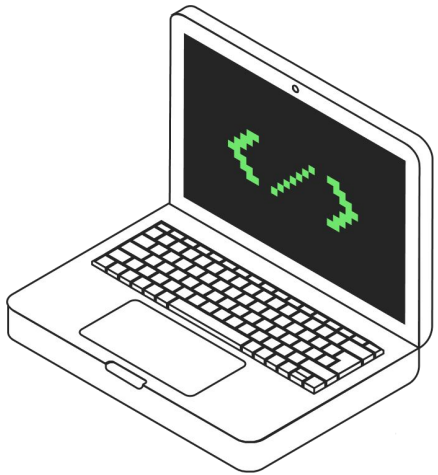
Обнаружение проблем на более ранних этапах разработки и предотвращение повышения стоимости продукта (поддержание стабильного качества продукта)

Строковой метод, заменяющий фигурные скобки на переменные

Принципы тестирования программного обеспечения

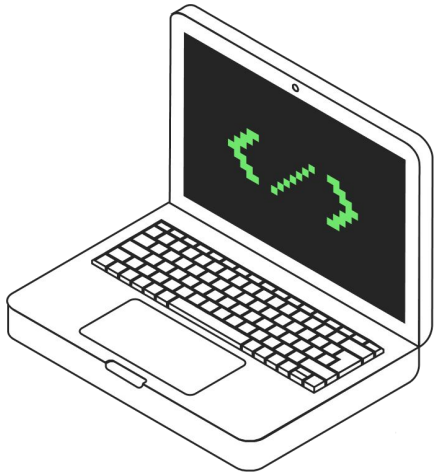
Семь принципов тестирования программного обеспечения

💡 Тестирование показывает наличие дефектов



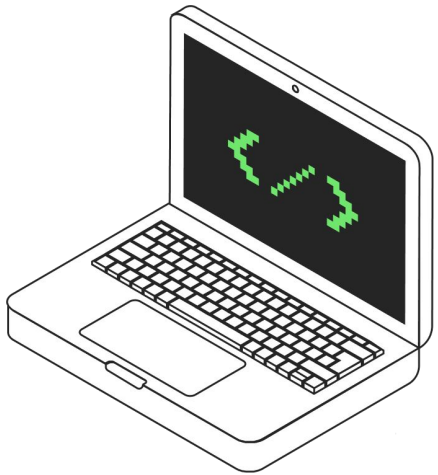
Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно



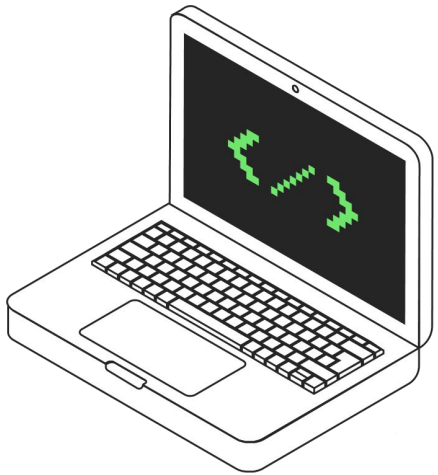
Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование



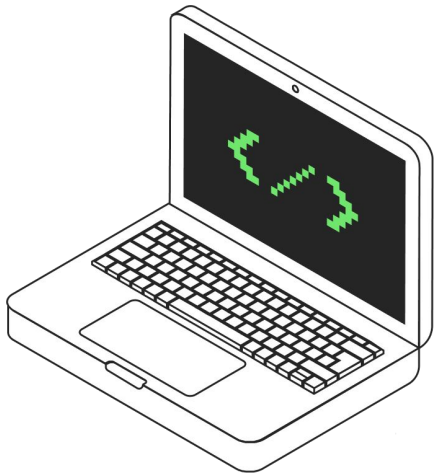
Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов



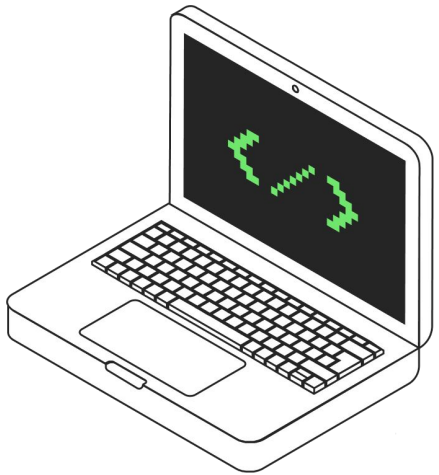
Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов
- 💡 Парадокс пестицидов



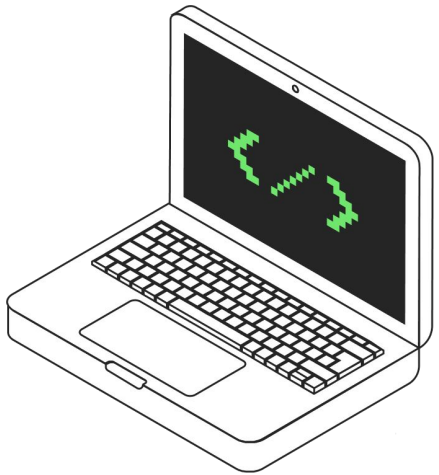
Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов
- 💡 Парадокс пестицидов
- 💡 Тестирование зависит от контекста



Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов
- 💡 Парадокс пестицидов
- 💡 Тестирование зависит от контекста
- 💡 Заблуждение об отсутствии ошибок



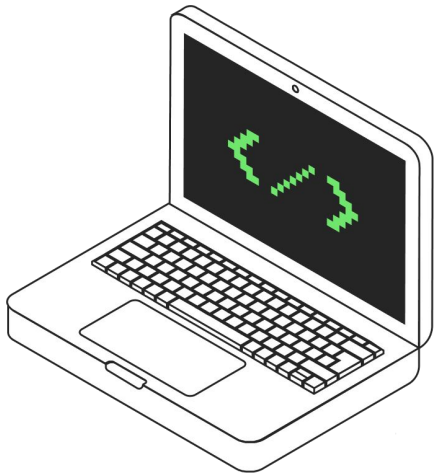
Вопрос

В тестировании ПО принято проанализировать продукт
или новую функцию после чего сфокусировать усилия в тестировании
на более рисковые и приоритетные случаи и участки нашего продукта.
Как вы думаете с каким признаком это связано?

Напишите ваш ответ в комментариях к видеолекции.
Время на размышление - 1 минута.

Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов
- 💡 Парадокс пестицидов
- 💡 Тестирование зависит от контекста
- 💡 Заблуждение об отсутствии ошибок



Ответ

2 принцип. Исчерпывающее тестирование невозможно Все случаи просто не могут быть протестированы, так как это заняло бы у нас очень много времени и в конце концов не стоило бы нам таких усилий

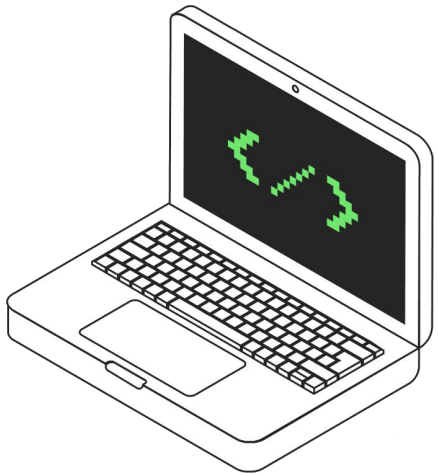
Вопрос

Перед тем как разрабатывать и тестировать тот или иной продукт, следует выяснить всю специфику, возможные конфликты в спецификации, невозможность новой фитчи. Как вы думаете с каким признаком это связано?

Напишите ваш ответ в комментариях к видеолекции.
Время на размышление - 1 минута.

Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов
- 💡 Парадокс пестицидов
- 💡 Тестирование зависит от контекста
- 💡 Заблуждение об отсутствии ошибок



Ответ

3 принцип. Раннее тестирование. Чем раньше найдутся баги тем дешевле их исправить! Помните про кривую Боэма.

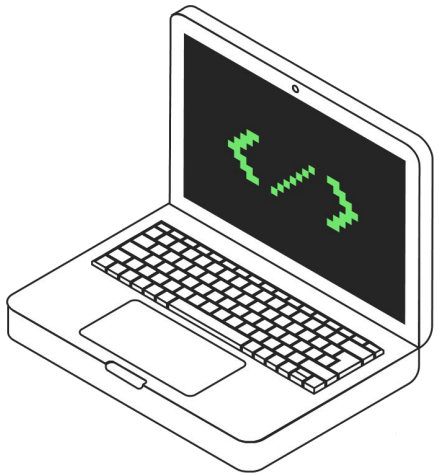
Вопрос

Иногда тестируя и выискивая баги, мы забываем посмотреть с другой стороны и спросить а нужно ли это пользователю. Какой признак говорит об этом?

Напишите ваш ответ в комментариях к видеолекции.
Время на размышление - 1 минута.

Семь принципов тестирования программного обеспечения

- 💡 Тестирование показывает наличие дефектов
- 💡 Исчерпывающее тестирование невозможно
- 💡 Раннее тестирование
- 💡 Кластеризация дефектов
- 💡 Парадокс пестицидов
- 💡 Тестирование зависит от контекста
- 💡 Заблуждение об отсутствии ошибок

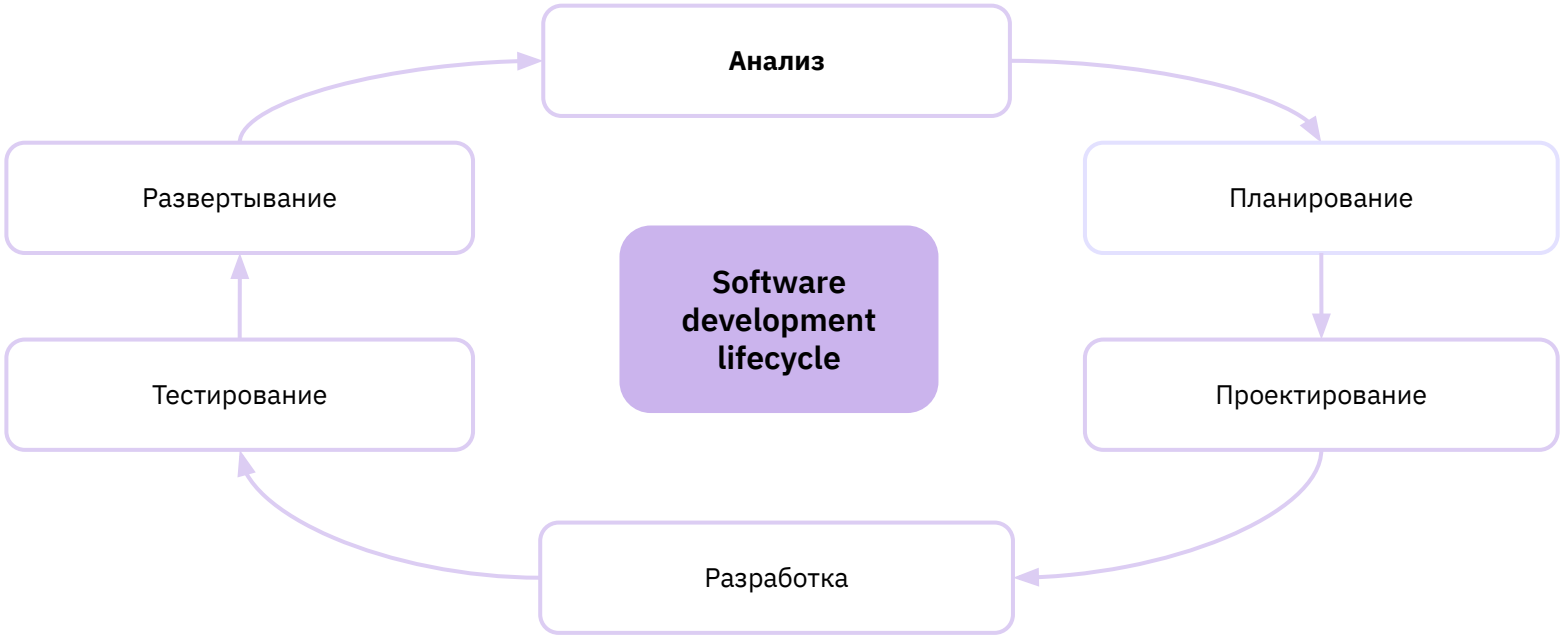


Ответ

Это 7 принцип. Заблуждение об отсутствии ошибок.
Если предложенный функционал не нужен пользователю, то какой бы качественный наш продукт не был — это уже не так важно.

Цикл разработки

Цикл разработки. SDLC



Цикл разработки. SDLC

📌 **Анализ**

На этом этапе формируются бизнес-требования к продукту, учитываются пожелания и потребности пользователей, оцениваются потенциальные риски и возможности. К работе привлекаются специалисты по продукту.

💡 **В результате всех этих усилий должны появиться ответы на вопросы:** «Какие проблемы должно решать ПО?», «Что именно (какой продукт) необходимо сделать?» и «Как именно это должно работать?».

Цикл разработки. SDLC

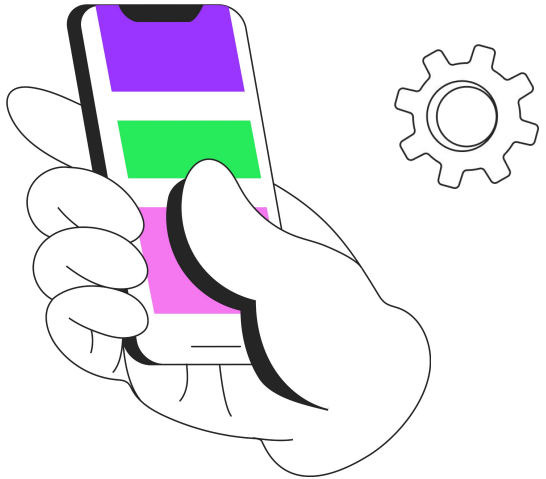
📌 **Планирование**

На этапе планирования руководители проекта оценивают условия проекта. Это включает в себя расчет затрат на рабочую силу и материалы, составление графика с указанием целевых показателей, а также создание команд проекта и структуры руководства.

Цикл разработки. SDLC

📌 **Проектирование**

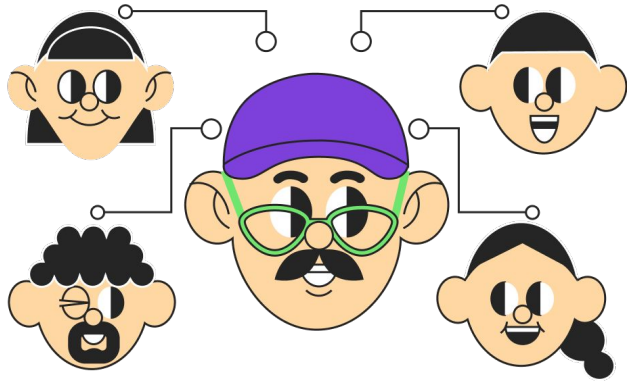
На этапе проектирования моделируется то, как будет работать программное приложение. На этом этапе обсуждается архитектура и другие тонкости системы. Также создание прототипа может быть частью этапа проектирования. Прототип демонстрирует основную идею о том, как приложение выглядит и работает



Цикл разработки. SDLC

📌 **Разработка**

Это фактическое написание программы. Небольшой проект может быть написан одним разработчиком, в то время как большой проект может быть разделен и работать несколькими командами.



Цикл разработки. SDLC

📌 **Тестирование**

Очень важно протестировать приложение, прежде чем сделать его доступным для пользователей. Большая часть тестирования может быть автоматизирована, например, тестирование безопасности. Другое тестирование может быть выполнено только в определенной среде. Тестирование должно гарантировать, что каждая функция работает правильно. Различные части приложения также должны быть протестированы на бесперебойную совместную работу — интеграционные тесты, чтобы уменьшить любые зависания или задержки в обработке.

После этого продукт можно внедрять и интегрировать со сторонним программным обеспечением. Процесс разработки на этом не заканчивается — он продолжается, пока не будут внесены доработки

Цикл разработки. SDLC

📌 **Развертывание**

На этапе развертывания приложение становится доступным для пользователей. Многие компании предпочитают автоматизировать этап развертывания.

Вопрос

Расставьте по порядку этапы согласно SDLC, используя цифры (1-Анализ):

📈 Анализ

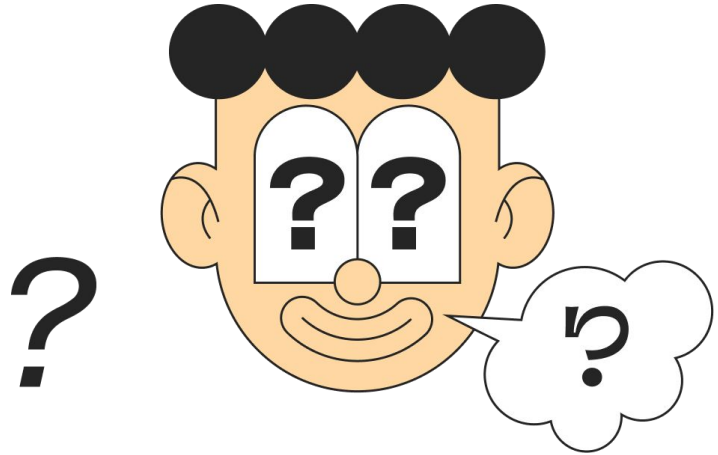
(1) Проектирование

(2) Планирование

(3) Разработка

(4) Развертывание

(5) Тестирование



Ответ

Расставьте по порядку этапы согласно SDLC, используя цифры (1-Анализ):

📈 Анализ

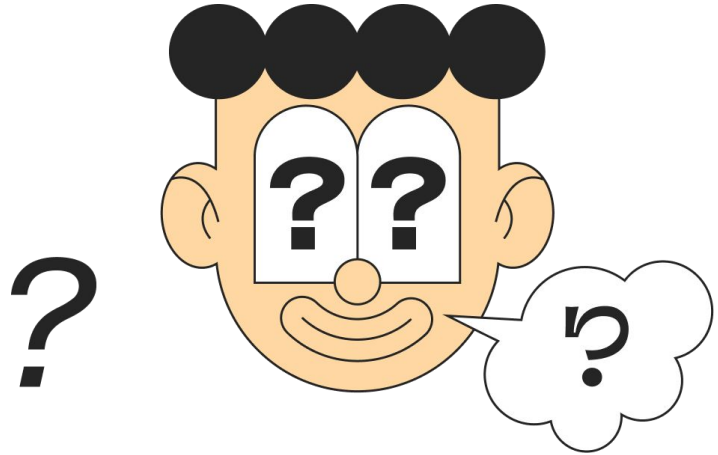
(2) Планирование

(1) Проектирование

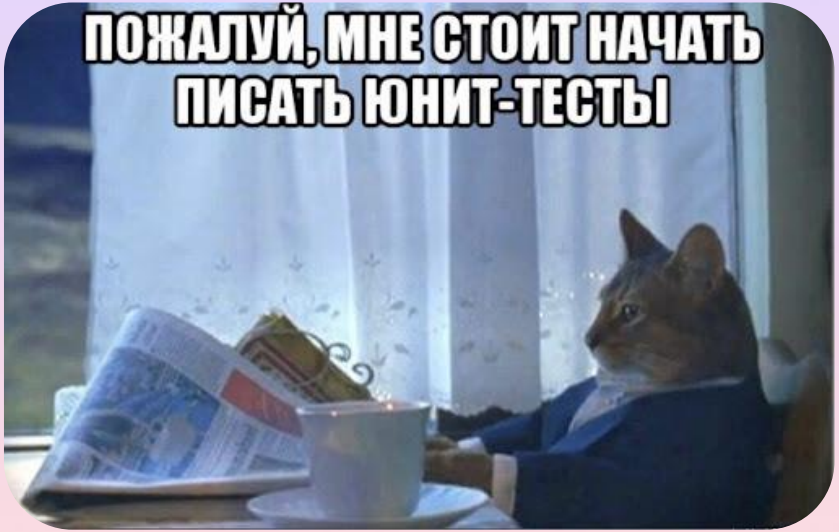
(3) Разработка

(5) Тестирование

(4) Развертывание



Написание простого теста



Написание тестов

Перед нами стоит задача написать простой консольный калькулятор на Java и протестировать его.

Это будет программа, которая принимает на вход числа и действие, которое нужно с ними сделать и выводит результат вычисления в консоль.

```
public static int calculation(int firstOperand, int secondOperand, char operator) {
   int result = 0;

   switch (operator) {
       case '+':
           result = firstOperand + secondOperand;
           break;
       case '-':
           result = firstOperand - secondOperand;
           break;
       case '*':
           result = firstOperand * secondOperand;
           break;
       case '/':
           result = firstOperand / secondOperand;
           break;
   }
   return result;
}
```

```
public static void main(String[] args) {
// Ручное тестирование выдает ожидаемый результат. Можно проверить метод таким образом, но эти проверки занимают много времени и плохо масштабируется
      System.out.printf("Результат операции: %s \n", calculation(2,2,'+'));                    //
Результат операции: 4
      System.out.printf("Результат операции: %s \n", calculation(2,1,'-'));  // Результат операции: 1
      System.out.printf("Результат операции: %s \n", calculation(2,3,'*'));  // Результат операции: 6
      System.out.printf("Результат операции: %s \n", calculation(8,2,'/'));  // Результат операции: 4
   }
```

Написание тестов. Пограничные случаи

Ошибки в пограничных случаях — самая частая причина логических ошибок в программах. Программисты всегда забывают что-нибудь учесть. Такие ошибки часто проявляются не сразу, и могут долгое время не приводить к видимым проблемам. Программа продолжает работать, но в какой-то момент обнаруживается, что в результатах есть ошибки

```
 public static void main(String[] args) {
       // Ручное тестирование выдает ожидаемый результат. Можно проверить метод таким образом, но эти
проверки занимают много времени и плохо масштабируется
       System.out.printf("Результат операции: %s \n", calculation(2,2,'+'));  // Результат операции: 4
       System.out.printf("Результат операции: %s \n", calculation(2,1,'-'));  // Результат операции: 1
       System.out.printf("Результат операции: %s \n", calculation(2,3,'*'));  // Результат операции: 6
       System.out.printf("Результат операции: %s \n", calculation(8,2,'/'));  // Результат операции: 4
     //Пограничные случаи. Знак подчеркивания _ вместо минуса
       System.out.printf("Результат операции: %s \n", calculation(8, 6, '_'));  // Результат операции: 0 ???

   }
```

```
    public static void main(String[] args) {
        // Ручное тестирование выдает ожидаемый результат. Можно проверить метод таким образом, но
эти проверки занимают много времени и плохо масштабируется
        System.out.printf("Результат операции: %s \n", calculation(2,2,'+'));  // Результат операции:
4
        System.out.printf("Результат операции: %s \n", calculation(2,1,'-'));  // Результат операции:
1
        System.out.printf("Результат операции: %s \n", calculation(2,3,'*'));  // Результат операции:
6
        System.out.printf("Результат операции: %s \n", calculation(8,2,'/'));  // Результат операции:
4


        //Пограничные случаи. Знак подчеркивания _ вместо минуса
        System.out.printf("Результат операции: %s \n", calculation(8, 6, '_'));

        //Пограничные случаи. Нулевой аргумент
        //System.out.printf("Результат операции: %s \n", calculation(8, 0, '/'));

        //Пограничные случаи. Большие числа
        System.out.printf("Результат операции: %s \n", calculation(12345678910, 1, '*'));
        System.out.printf("Результат операции: %s \n", calculation(1234567891, 10, '*')); // Результат
```

```java
public class CalculatorTest {
   public static void main(String[] args) {
            // Проверка базового функционала с целыми числами:
            if (8 != Calculator.calculation(2, 6, '+')) {
                throw new AssertionError("Ошибка в методе");
            }
            if (0 != Calculator.calculation(2, 2, '-')) {
                throw new AssertionError("Ошибка в методе");
            }
            if (14 != Calculator.calculation(2, 7, '*')) {
                throw new AssertionError("Ошибка в методе");
            }

            if (2 != Calculator.calculation(100, 50, '/')) {
                throw new AssertionError("Ошибка в методе");
            }
   }
}
```

```java
public class Calculator {
   public static void main(String[] args) { ... }

   // Вариант calculation с исправленными ошибками, выявленными ручным тестированием)
   public static int calculation(int firstOperand, int secondOperand, char operator) {
       int result;

       switch (operator) {
           case '+':
               result = firstOperand + secondOperand;
               break;
           case '-':
               result = firstOperand - secondOperand;
               break;
           case '*':
               result = firstOperand * secondOperand;
               break;
           case '/':
               if (secondOperand != 0) {
                   result = firstOperand / secondOperand;
                   break;
               } else {
                   throw new ArithmeticException("Division by zero is not possible");
               }
           default:
               throw new IllegalStateException("Unexpected value operator: " + operator);
       }
       return result;
   }
```

```
public class CalculatorTest {
   public static void main(String[] args) {
          // Проверка базового функционала с целыми числами:
                  ...
          //      Случаи с неправильными аргументами
          //    аргумент operator типа char, должен вызывать исключение, если он получает не
базовые символы (+-*/)
            try {
                Calculator.calculation(8, 4, '_');
            } catch (IllegalStateException e) {
                if (!e.getMessage().equals("Unexpected value operator: _")) {
                    throw new AssertionError("Ошибка в методе");
                }
            }
   }
}
```

Утверждения (Assert)

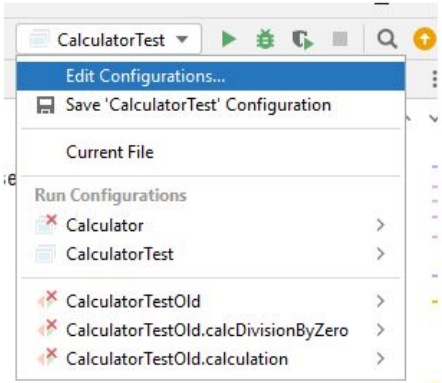
Утверждения (Assert) – это встроенный в java механизм проверки правильности предположений. Он в основном используется для тестирования во время разработки. Утверждения реализуются с помощью оператора assert и java.lang.Class AssertionError. В коде это выражается следующим образом:

```
assert booleanExpression;
```

```
public class CalculatorTest {
   public static void main(String[] args) {
          // Проверка базового функционала с целыми числами:
                  ...
          //      Случаи с неправильными аргументами
                  ...

// Проверка базового функционала с целыми числами, с использованием утверждений (assert):
                assert 8 == Calculator.calculation(2, 6, '+');
                assert 0 == Calculator.calculation(2, 2, '-');
                assert 14 == Calculator.calculation(2, 7, '*');
                assert 2 == Calculator.calculation(100, 90, '/'); // Пример с ошибкой, для
демонстрации работы assert
}
```

Как запустить программу с assert:



Вопрос

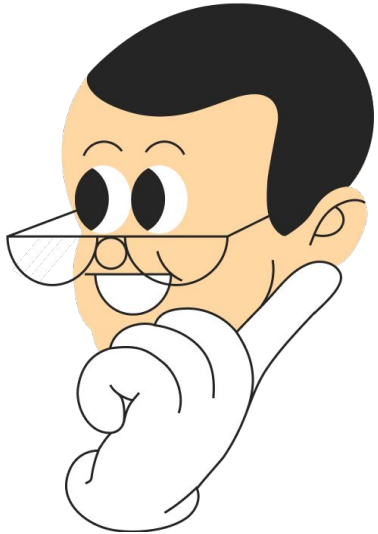
Когда следует использовать исключения, а когда утверждения?

Напишите ваш ответ в комментариях к видеолекции.
Время на размышление - 1 минута.

Когда следует использовать исключения, а когда утверждения?

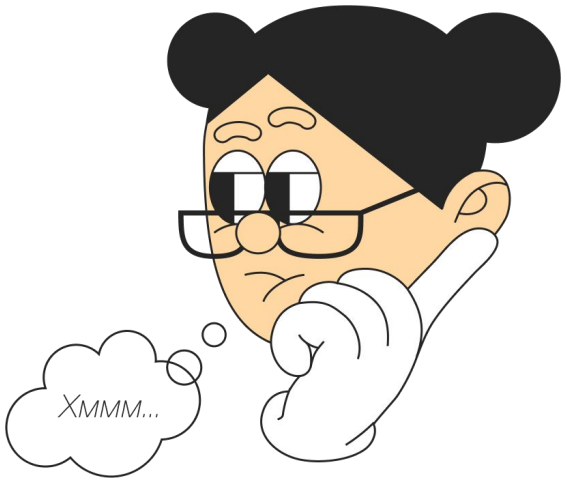
Когда проверка может быть полезна, нужно использовать assert'ы, но нужно учитывать, что они могут быть удалены на этапе компиляции либо во время исполнения программы, поэтому они не должны изменять выполнение программы. Если в результате удаления assert'а поведение программы может измениться, то это явный признак неправильного использования assert'а!

Исключения же можно писать там где после обнаружения бага, может потребоваться логика дальнейшей обработки ошибок (try-catch).



Недостатки assert

- ✘ Assert'ы, по умолчанию выключены
- ✘ В проверяемых assert'ом функциях может быть только булево выражение, это ограничивает использование утверждений, приходится изменять код для проверок с assert'ом



Разница между фреймворком и библиотекой

Фреймворк

Набор взаимосвязанных классов и методов, которые позволяют создавать приложения, и которые можно использовать в других приложениях.

JUnit — фреймворк для модульного тестирования программного обеспечения на языке Java.

Библиотека

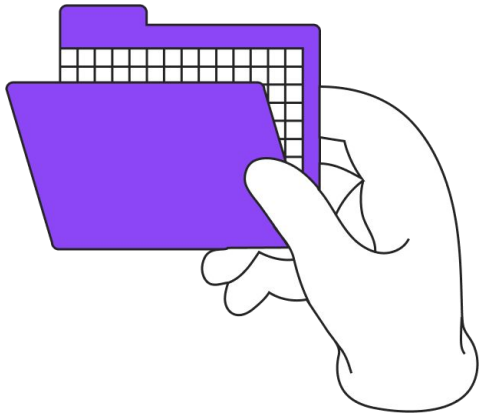
Это просто набор классов. Например, классы, которые вы написали сами.

AssertJ — библиотека для написания более гибких и удобочитаемых утверждений.

Библиотека AssertJ

AssertJ — это библиотека Java, которая предоставляет богатый набор утверждений и действительно полезных сообщений об ошибках, улучшает читаемость тестового кода и разработана так, чтобы ее было очень просто использовать в вашей любимой IDE.

https://assertj.github.io/doc/



```
import static org.assertj.core.api.Assertions.*;

public class CalculatorTest {
   public static void main(String[] args) {
        ...

       // Проверка базового функционала с целыми числами, с использованием утверждений
AssertJ:
       assertThat(Calculator.calculation(2, 6, '+')).isEqualTo(9);
       assertThat(Calculator.calculation(2, 2, '-')).isEqualTo(0);
       assertThat(Calculator.calculation(2, 7, '*')).isEqualTo(14);
       assertThat(Calculator.calculation(100, 90, '/')).isEqualTo(2); // Пример с ошибкой,
для демонстрации работы assertJ


   }
}
```

```
import static org.assertj.core.api.Assertions.*;

public class CalculatorTest {
   public static void main(String[] args) {
        ...

       // Проверка ожидаемого исключения
       assertThatThrownBy( () -> Calculator.calculation(8, 4, '_')
).isInstanceOf(IllegalStateException.class);


   }
}
```

Библиотека AssertJ. Возможности

```
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);
```

```
assertThat(fellowshipOfTheRing).hasSize(9) .contains(frodo, sam)
.doesNotContain(sauron);
```

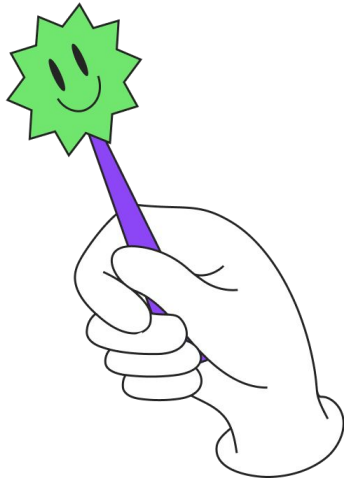
```
assertThat(fellowshipOfTheRing).filteredOn(character ->
character.getName().contains("o")) .containsOnly(aragorn, frodo,
legolas, boromir);
```

```
assertThat(frodo.getAge()).as("check %s's age",
frodo.getName()).isEqualTo(33);
```

Итоги занятия

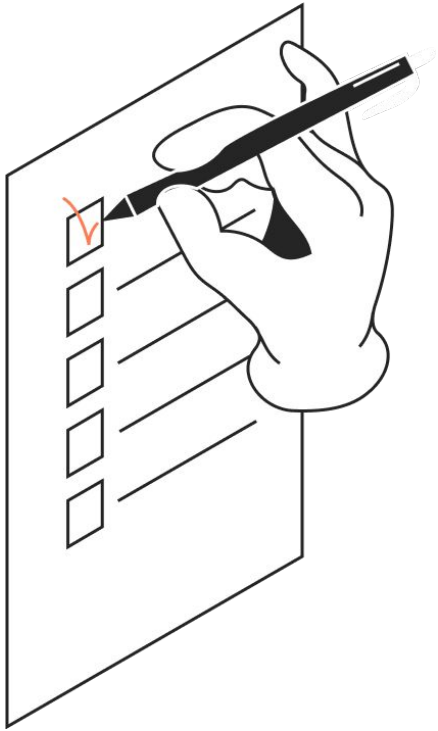
Что было на уроке сегодня

- 📌 Узнали что такое тестирование.
- 📌 Разобрались в том что такое ошибки и узнали какие они бывают.
- 📌 Узнали о принципах тестирования программного обеспечения.
- 📌 Рассмотрели как устроен цикл разработки ПО.
- 📌 Познакомились с утверждениями и библиотекой AssertJ.



Задание

1. Придумайте и опишите (можно в псевдокоде) функцию извлечения корня и необходимые проверки для него используя граничные случаи
2. Письменно ответьте: Как будет выглядеть проверка для случая деления на ноль? (с использованием AssertJ)
3. Письменно ответьте: Сравните одну и ту же проверку с использованием условий, ассертов, AssertJ в каком случае стандартное сообщение об ошибке будет более информативным?
4. (Устно) Протестируйте системный калькулятор на вашем ПК на пограничные условия, как он реагирует? Как на нем обрабатываться деление на ноль?



Спасибо за внимание