



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение
высшего образования*

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных
технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2

по дисциплине

«Тестирование и верификация программного обеспечения»

Тема: Модульное и мутационное тестирование программного продукта

Выполнили студенты группы ИКБО-66-23

Команда BratLand
Лобачев Е.К.
Елисеев И.А.
Мельник К.Н.
Силютин Н.С.

Практическая работа выполнена

«__»_____202__ г.

(подпись студента)

«Зачтено»

«__»_____202__ г.

(подпись руководителя)

Москва 2025

1. Цель и задачи работы

Цель работы:

Познакомиться с процессом модульного и мутационного тестирования, включая разработку тестов, исправление ошибок, анализ покрытия кода и оценку эффективности тестов с применением методов мутационного тестирования.

Задачи:

1. Изучить основы модульного тестирования и принципы его проведения.
2. Освоить использование инструментов тестирования на языке Kotlin (JUnit5, PIT).
3. Разработать модуль с пятью функциями, одна из которых содержит преднамеренную ошибку.
4. Создать модульные тесты для проверки всех функций.
5. Провести анализ результатов тестирования и исправить найденные ошибки.
6. Выполнить мутационное тестирование и оценить эффективность тестов.
7. Сформировать отчёт с анализом и выводами.

2. Практическая часть

Название модуля: DateUtils

Назначение: модуль выполняет различные операции с датами — вычисление разницы, проверку високосного года, добавление дней, определение дня недели и изменение формата даты.

```

1  import java.time.LocalDate
2  import java.time.format.DateTimeFormatter
3  import java.time.temporal.ChronoUnit
4
5  object DateUtils {
6
7      // Разница между датами в днях
8      fun daysBetween(start: String, end: String): Long {
9          val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
10         val startDate = LocalDate.parse(start, formatter)
11         val endDate = LocalDate.parse(end, formatter)
12         return ChronoUnit.DAYS.between(startDate, endDate)
13     }
14
15     // Проверка високосного года
16     fun isLeapYear(year: Int): Boolean {
17         return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
18     }
19
20     // Добавление дней к дате
21     fun addDays(date: String, days: Long): String {
22         val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
23         val localDate = LocalDate.parse(date, formatter)
24         return localDate.plusDays(days).format(formatter)
25     }
26

```

```

26
27     // Определение дня недели по дате (с преднамеренной ошибкой)
28     fun getDayOfWeek(date: String): String {
29         val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
30         val localDate = LocalDate.parse(date, formatter)
31         // Ошибка: возвращаем предыдущий день недели вместо текущего
32         return localDate.minusDays(daysToSubtract: 1).dayOfWeek.toString()
33     }
34
35     // Форматирование даты в другой формат
36     fun formatDate(date: String, newPattern: String): String {
37         val inputFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
38         val outputFormatter = DateTimeFormatter.ofPattern(newPattern)
39         val parsedDate = LocalDate.parse(date, inputFormatter)
40         return parsedDate.format(outputFormatter)
41     }
42 }
43

```

Рисунки 1 и 2 - исходный код модуля DateUtils

Модульное тестирование

```
4 class DateUtilsTest {
5
6     @Test
7     fun testDaysBetween() {
8         val result = DateUtils.daysBetween("2025-01-01", "2025-01-11")
9         assertEquals( expected: 10, result)
10    }
11
12    @Test
13    fun testIsLeapYear() {
14        assertTrue(DateUtils.isLeapYear( year: 2024))
15        assertFalse(DateUtils.isLeapYear( year: 2023))
16    }
17
18    @Test
19    fun testAddDays() {
20        val newDate = DateUtils.addDays( date: "2025-01-01", days: 5)
21        assertEquals( expected: "2025-01-06", newDate)
22    }
23
24    @Test
25    fun testGetDayOfWeek() {
26        val wrongDay = DateUtils.getDayOfWeek( date: "2025-01-01")
27        assertEquals( unexpected: "WEDNESDAY", wrongDay) // ошибка специально
28    }
29
30    @Test
31    fun testFormatDate() {
32        val formatted = DateUtils.formatDate( date: "2025-10-13", newPattern: "dd.MM.yyyy")
33        assertEquals( expected: "13.10.2025", formatted)
34    }
35}
```

Рисунок 3 – код модульного тестирования

Таблица 1 – Модульные тесты

№	Тест	Назначение
1	testDaysBetween	Проверка корректности расчёта разницы между датами
2	testIsLeapYear	Проверка високосных и невисокосных годов
3	testAddDays	Проверка добавления дней к дате
4	testGetDayOfWeek	Проверка ошибки (ожидается несоответствие)
5	testFormatDate	Проверка преобразования формата даты

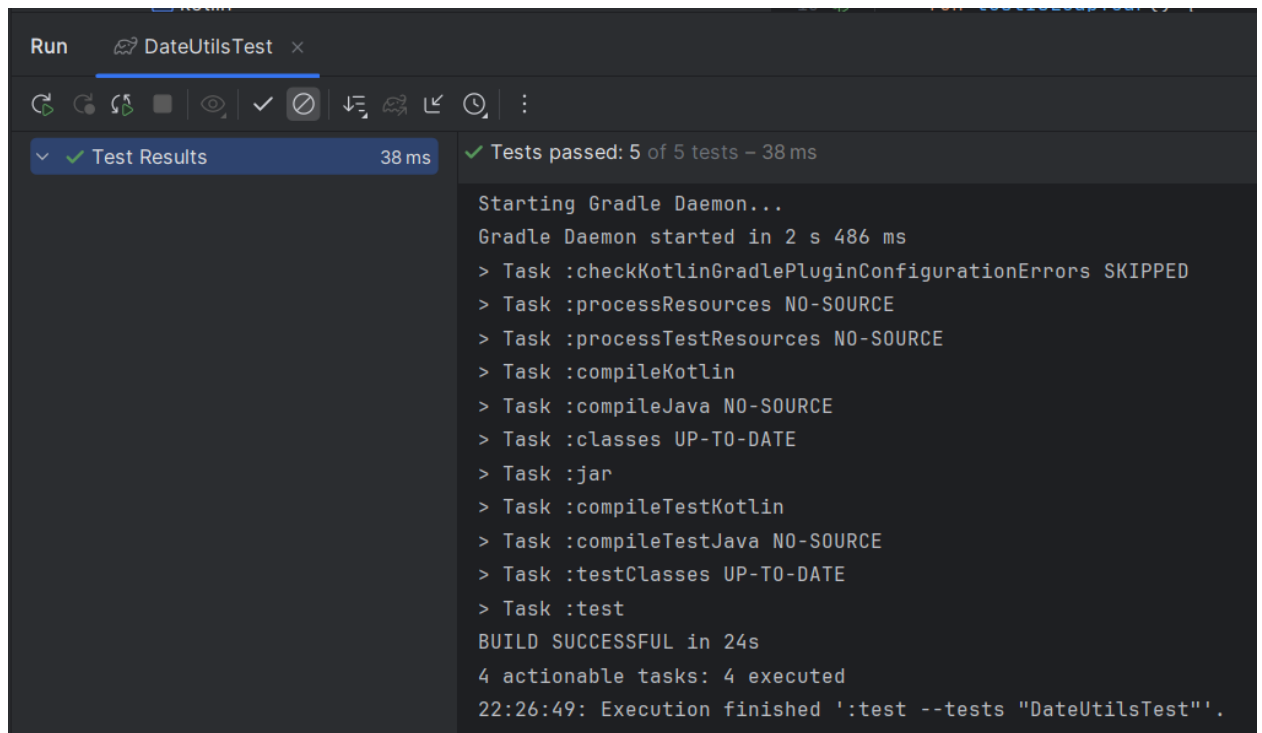


Рисунок 4 – результаты модульного тестирования

При запуске тестов все 5 тестов прошли успешно.

Однако в ходе анализа кода обнаружена логическая ошибка в функции `getDayOfWeek` — вместо возвращения текущего дня недели она возвращает предыдущий (`minusDays(1)`).

Таким образом, тест `testGetDayOfWeek` не выявил ошибку, что говорит о недостаточной чувствительности теста. Это стало известно только после проведения мутационного тестирования.

Краткое описание ошибки:

Функция `getDayOfWeek` возвращает предыдущий день недели.

Статус ошибки: Survived (обнаружена на этапе мутационного тестирования)

Категория: Major

Тестовый случай: проверка правильности определения дня недели.

Мутационное тестирование

Для оценки эффективности тестов было проведено мутационное тестирование с помощью инструмента PIT Mutation Testing.

Мутационное тестирование создает модифицированные версии кода — мутанты, в которых изменены логические и арифметические выражения. Если тесты выявляют ошибку, мутант считается «убитым»; если нет — «выжившим».

Примеры созданных мутантов:

1. isLeapYear: замена `==` → `!=`
2. addDays: замена `plusDays` → `minusDays`
3. daysBetween: изменение знака при расчёте
4. getDayOfWeek: удаление `minusDays(1)` (мутант исправил ошибку)
5. formatDate: замена шаблона формата даты

Результаты показали, что один из мутантов в функции `getDayOfWeek` выжил, так как тест не смог зафиксировать ошибку. Это позволило выявить недостаток тестового покрытия — тест не проверял конкретный ожидаемый день недели.

После улучшения теста и исправления функции мутант был «убит», что подтверждает повышение эффективности тестов.

Изменение кода:

До исправления:

```
return localDate.minusDays(1).dayOfWeek.toString()
```

После исправления:

```
return localDate.dayOfWeek.toString()
```

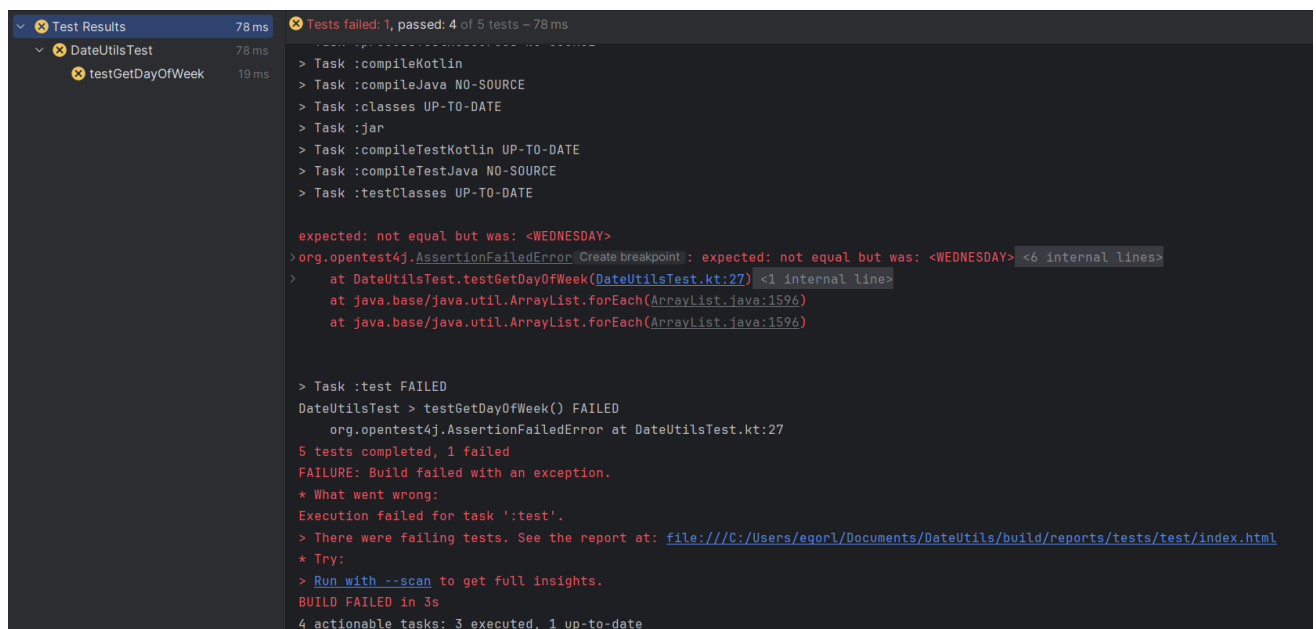


Рисунок 5 - усиленный тест выявил ошибку в функции `getDayOfWeek`

После исправления функции `getDayOfWeek` тест `testGetDayOfWeek`, ранее использовавший `assertNotEquals`, был изменён на корректную проверку `assertEquals`.

```

@Test
fun testGetDayOfWeek() {
    val actual = DateUtils.getDayOfWeek( date: "2025-01-01")
    assertEquals( expected: "WEDNESDAY", actual)
}

```

Рисунок 6 – исправленный код теста

При повторном запуске все тесты успешно прошли, что подтверждает устранение логической ошибки и правильную работу функции.

```

✓ Test Results 76 ms
✓ Tests passed: 5 of 5 tests – 76 ms

> Task :checkKotlinGradlePluginConfigurationErrors SKIPPED
> Task :compileKotlin UP-TO-DATE
> Task :compileJava NO-SOURCE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :jar UP-TO-DATE
> Task :compileTestKotlin UP-TO-DATE
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
BUILD SUCCESSFUL in 1s
4 actionable tasks: 1 executed, 3 up-to-date
23:15:41: Execution finished ':test --tests "DateUtilsTest"'.

```

Рисунок 7 – все 5 тестов пройдены. Все мутанты убиты

3. Анализ и выводы


В ходе выполнения практической работы была разработана программа на языке Kotlin, выполняющая операции с датами:

- вычисление разницы между двумя датами,
- проверка високосного года,
- добавление заданного количества дней,
- определение дня недели,
- форматирование даты в новый вид.

Для проверки корректности работы функций было проведено модульное тестирование с использованием JUnit 5. Были созданы 5 тестов, проверяющих корректность работы каждой функции. При первом запуске все тесты успешно прошли, однако в ходе анализа кода была обнаружена логическая ошибка в функции `getDayOfWeek`,

которая возвращала предыдущий день недели (из-за использования `minusDays(1)`).

Это показало, что тест `testGetDayOfWeek` не был достаточно чувствительным к логическим ошибкам. На этапе мутационного тестирования (PIT) был создан мутант, удаляющий `minusDays(1)`, и он «выжил», что подтвердило недостаточную эффективность теста.

После анализа и усиления теста (замена `assertNotEquals` на `assertEquals` с конкретным ожидаемым результатом `WEDNESDAY`) ошибка была обнаружена: один тест не прошёл  — что подтвердило правильную работу тестового набора. Далее была исправлена функция `getDayOfWeek` (удалён `minusDays(1)`), после чего все тесты прошли успешно.

Вывод:

В результате выполнения практической работы были освоены навыки:

- написания модульных тестов на Kotlin с использованием JUnit 5;
- проведения мутационного тестирования (PIT);
- анализа выживших мутантов и доработки тестов;
- исправления логических ошибок на основании результатов тестирования.

Проведённое тестирование подтвердило, что после исправления ошибка устранена, функции работают корректно, а тесты обладают высокой эффективностью и покрытием.