ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ КОСТРОМСКОЙ ОБЛАСТИ

ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ «КОСТРОМСКОЙ ЭНЕРГЕТИЧЕСКИЙ ТЕХНИКУМ ИМ. Ф.В. ЧИЖОВА»

Химико-технологическое отделение

Специальность 09.02.07 «Информационные системы и программирование»

ОП.04 Основы алгоритмизации и программирования

**Реферат**

**"Принцип разработки программного обеспечения TDD"**

Выполнил:

студент 2-1 ИС,

Варнаков Степан Максимович.

Преподаватель

Ищук Денис Николаевич.

Кострома, 2024 г.

Оглавление

[Глоссарий 3](#_Toc187532581)

[Введение 4](#_Toc187532582)

[Зачем нам тестирование? 4](#_Toc187532583)

[Подходы к разработке ПО 4](#_Toc187532584)

[TDD 4](#_Toc187532585)

[BDD 9](#_Toc187532586)

[Web-UI 15](#_Toc187532587)

[Практика 15](#_Toc187532588)

[Список литературы 16](#_Toc187532589)

# Глоссарий

# Введение

Разработка программного обеспечения – это комплексный и сложный процесс со множеством разнообразных задач. Среди них не только обеспечение корректной работы системы, но и соблюдение законодательных требований и стандартов индустрии. При этом не менее важны гибкость и экономическая эффективность.

Чтобы соответствовать всем стандартам постоянно развивающейся производственной среды, разработчикам необходимо полагаться на тестирование программного обеспечения, которое, в свою очередь, является одним из ключевых элементов, обеспечивающих развитие индустрии. В этой статье мы рассмотрим, почему тестирование необходимо. Я познакомлю вас с лучшими практиками, инструментами и платформами тестирования на Java.

## Зачем нам тестирование?

Тестирование помогает быстро обнаружить и исправить ошибки, что имеет решающее значение для обеспечения качества продукта перед его запуском на клиентах. Иными словами, это залог достижения их удовлетворённости и доверия.

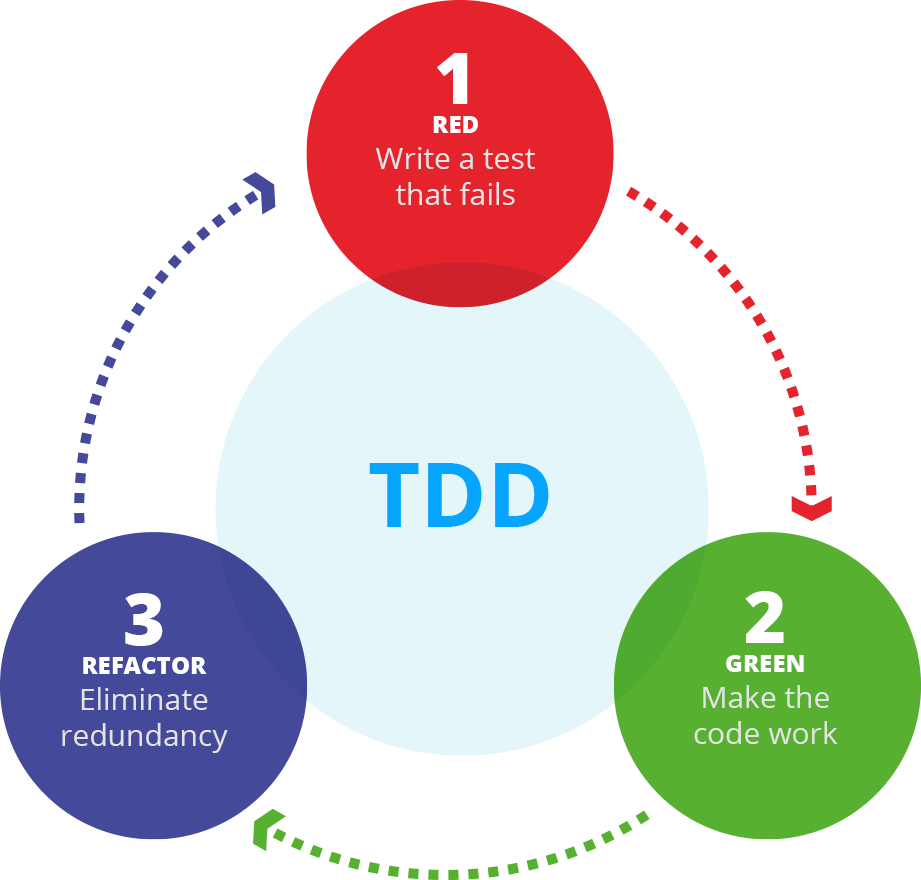
Причины, почему тестирование важно при разработке ПО, включают следующее:

1. **Простота проверки ожидаемого поведения кода:**самый простой способ убедиться, что ваш код работает правильно, – это протестировать его. Это позволит вам быть уверенным, что, например, определённая часть кода делает то, что должна делать в данных условиях. Важно ничего не сломать во время внесения изменений в уже работающую систему, и подобные тесты могут показать, будут ли ваши изменения работать с остальной частью системы.
2. **Подстраховка для инженеров при командной работе:** использование тестов важно и в коллаборативной среде. Разработчики обычно работают с кодом, который они изначально не писали. Однако, запустив уже имеющиеся тесты, разработчик может отслеживать, ведёт ли код себя так же, как раньше (или по-другому, если он этого ожидает), и избежать неприятных побочных эффектов.
3. **Документация через тесты:** Тестирование может также играть роль дополнительного способа ведения документации, которое отличается от комментариев и традиционной документации тем, что оно не пытается описать природу программы, а скорее демонстрирует, что и в каких сценариях должно происходить.
4. **Дополнительный уровень верификации, уменьшающий зависимость от QA:**тест добавляет ещё один уровень проверки, уменьшая зависимость от команд качества, которые часто играют важную роль в процессе разработки. Умело применяя методы тестирования, разработчики могут решить множество потенциальных проблем ещё до того, как код достигнет контроля качества. Ещё одним преимуществом этого подхода является сокращение циклов итераций, которое позволяет QA тестировать более сложные сценарии системного уровня. С другой стороны, это также помогает компании сэкономить на расходах на тестирование, поскольку на начальных этапах не нужно нанимать больше тестировщиков. (Urikh, 2024)

## Подходы к разработке ПО

### TDD

TDD, сокращение от Test-Driven Development, — это подход к разработке программного обеспечения, при котором тесты пишутся перед написанием кода. Процесс включает в себя три этапа: написание неудачного теста, реализацию кода для прохождения теста, а затем рефакторинг кода для улучшения дизайна. TDD способствует сокращению циклов разработки, помогает обеспечить корректность кода и улучшает ремонтопригодность и охват тестированием.



(TDD, 2025)

**TDD имеет ряд преимуществ, в том числе:**

* **Улучшение качества кода:** написание тестов до написания кода помогает убедиться, что код хорошо продуман и не содержит ошибок.
* **Сокращение времени разработки:** TDD может помочь сократить время разработки, не позволяя разработчикам писать код, который не работает или не соответствует требованиям.
* **Повышение удобства сопровождения:** TDD может помочь сделать код более удобным для сопровождения, упростив его понимание и изменение.

**Циклы TDD**

Цикл TDD можно свести к трём этапам:

#### 1. Красная Фаза

На этапе «красный» разработчик пишет неудачный тест, описывающий желаемое поведение новой функции или исправление ошибки. Тест должен быть написан так, чтобы он не прошёл, если функция или исправление ошибки не реализованы.

Цель красной фазы — заставить разработчика задуматься о структуре кода до того, как он начнёт его писать. Это поможет избежать таких проблем, как дублирование кода и «спагетти-код».

#### 2. Зелёная Фаза

На зелёной фазе разработчик пишет ровно столько кода, чтобы неудачный тест прошёл. Этот код может быть неидеальным, но его должно быть достаточно для прохождения теста.

На зелёной фазе цель состоит в том, чтобы как можно быстрее заставить код работать. Это поможет избежать ситуации, когда вы тратите слишком много времени на проектирование и недостаточно — на реализацию.

#### 3. Этап рефакторинга

На этапе рефакторинга разработчик улучшает код, не меняя его поведение. Это может включать в себя удаление дублирования, переименование переменных или реструктуризацию кода.

Цель этапа рефакторинга — упростить код для понимания и поддержки. Это поможет избежать таких проблем, как ошибки и снижение производительности.

**Инструменты и фреймворки для TDD на Java**

Для разработки на основе тестирования (TDD) на Java доступно несколько инструментов и фреймворков. Вот некоторые из них:

1. **JUnit:** популярная [**среда тестирования Java**](https://www.educba.com/testing-frameworks-for-java/) называется JUnit. Она предлагает утверждения и аннотации для создания и запуска модульных тестов. С помощью JUnit можно легко создавать тестовые сценарии, группировать тесты в наборы и тестировать различные сценарии.
2. **Mockito** — это фреймворк для имитации, который позволяет создавать фиктивные объекты для тестирования. Он помогает изолировать тестируемый код, имитируя поведение зависимостей. Mockito упрощает написание модульных тестов, предоставляя методы для определения фиктивного поведения и проверки взаимодействий.
3. **По сравнению с JUnit, TestNG** — это ещё одна платформа для тестирования, предоставляющая более широкие функциональные возможности. Поддерживает тестирование на основе данных, параллельное выполнение тестов и настраиваемые параметры. TestNG предлагает утверждения и аннотации, аналогичные тем, что есть в JUnit, для написания тестов.
4. **Hamcrest:** Hamcrest — это библиотека, предоставляющая набор сопоставлений для написания выразительных и понятных утверждений в тестах. Она позволяет создавать собственные сопоставления и предоставляет различные встроенные сопоставления для разных типов данных.
5. **PowerMock:** PowerMock — это фреймворк, расширяющий возможности имитации и позволяющий имитировать статические методы, финальные классы и закрытые методы. Он работает в сочетании с фреймворками имитации, такими как Mockito или EasyMock, и предоставляет дополнительные функции для тестирования кода с более сложными зависимостями.

(Gupta, 2023)

**Пример:**

|  |
| --- |
| Тест |
| import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.\*;  public class RectangleTest {  @Test void testAreaCalculation() {  Rectangle rectangle = new Rectangle(5, 10); *// create a rectangle object*  assertEquals(50, rectangle.getArea()); *// assert that the area is correct*  }  @Test void testAreaWithZeroWidth() {  Rectangle rectangle = new Rectangle(0, 10);  assertEquals(0, rectangle.getArea());  }  @Test void testAreaWithZeroHeight() {  Rectangle rectangle = new Rectangle(5,0);  assertEquals(0, rectangle.getArea());  }  @Test void testAreaWithNegativeWidth() {  assertThrows(IllegalArgumentException.class, ()-> new Rectangle(-5,10));  }  @Test void testAreaWithNegativeHeight() {  assertThrows(IllegalArgumentException.class, ()-> new Rectangle(5,-10));  }  } |
| Код |
| public class Rectangle {  private final int width;  private final int height;  public Rectangle(int width, int height) {  if (width < 0 || height < 0) {  throw new IllegalArgumentException("Width and height must be non-negative");  }  this.width = width;  this.height = height;  }  public int getArea() {  return width \* height;  }  } |

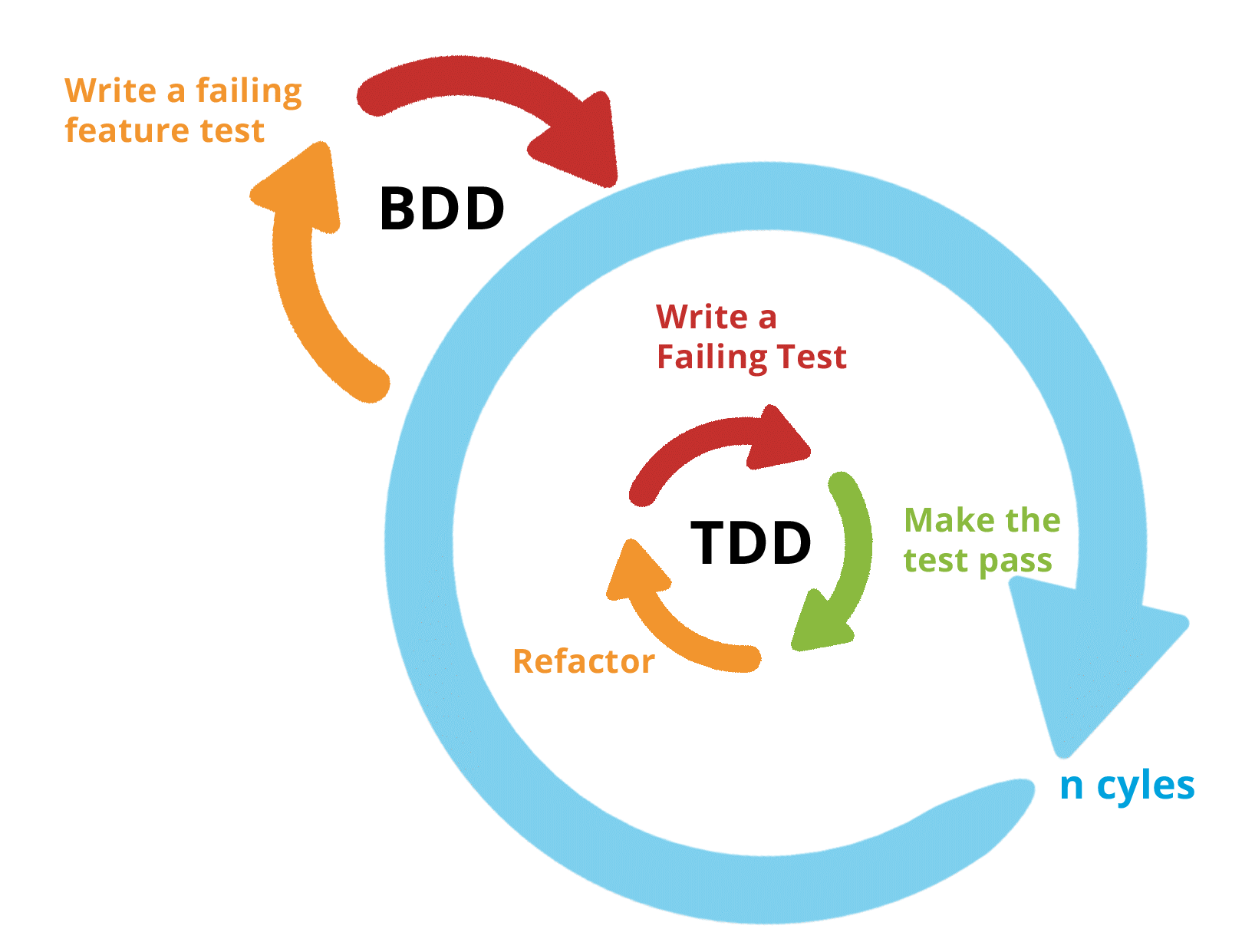
(OpenAI, 2024)

### BDD

Behavior Driven Development (BDD) — это подход к разработке программного обеспечения, который вырос из TDD. В то время как TDD сосредоточен на тестировании отдельных модулей кода, BDD расширяет этот подход, сосредоточиваясь на поведении системы в целом.

(Яндекс нейро, 2024)

При написании BDD тестов реализуются степы, или шаги — небольшие функции, которые выполняют одно определённое действие юзера. BDD использует естественный язык и конкретные примеры для описания ожидаемого поведения системы. Это помогает улучшить коммуникацию между разработчиками, тестировщиками и непрограммистами, такими как менеджеры проектов или стейкхолдеры.



(BDD, 2022)

**Принципы BDD**

BDD основывается на нескольких ключевых принципах:

1. Описание поведения: вместо того чтобы сосредоточиваться на технических деталях реализации, BDD фокусируется на описании ожидаемого поведения системы с точки зрения пользователя или стейкхолдера. Это помогает убедиться, что разрабатываемые функции действительно соответствуют потребностям пользователей.
2. Использование естественного языка: BDD использует естественный язык для описания сценариев тестирования, что делает их понятными не только для разработчиков и тестировщиков, но и для менеджеров проектов, бизнес-аналитиков и других участников команды.
3. Сотрудничество и коммуникация: BDD подчеркивает важность сотрудничества и общения между всеми участниками команды. Это помогает обеспечить общее понимание целей и требований проекта.

Примеры: BDD использует конкретные примеры для описания ожидаемого поведения. Это помогает уточнить требования и обеспечивает ясность в отношении того, что должна делать система.

**Плюсы:**

* Повышение качества коммуникации: BDD использует естественный язык для описания ожидаемого поведения, что делает его понятным для всех участников команды, включая непрограммистов.
* Сосредоточение на бизнес-целях: BDD помогает команде сосредоточиться на достижении конкретных бизнес-целей, а не просто на написании кода.
* Поддержка автоматизации тестирования: BDD поддерживает автоматизацию тестирования, что позволяет быстро и эффективно проверять поведение системы.

**Минусы:**

* Сложность внедрения: Внедрение BDD может потребовать значительных изменений в процессах разработки и тестирования, что может быть сложно для некоторых команд.
* Требуется обучение: для эффективного использования BDD команде может потребоваться обучение, особенно для понимания и написания сценариев на естественном языке.
* Риск неправильного понимания: если сценарии BDD написаны неправильно или нечетко, это может привести к неправильному пониманию требований или ожидаемого поведения системы.

(rmksrv, 2024)

**Инструменты и фреймворки для TDD на Java**

**Cucumber** — это инструмент разработки, управляемой поведением (BDD), который позволяет писать тесты в понятном человеку формате. Он использует синтаксис «Дано — Когда — Тогда» для описания сценариев и определения шагов для их сопоставления с исполняемым кодом. Cucumber способствует сотрудничеству между разработчиками, тестировщиками и заинтересованными сторонами.

Файл функций (.feature) описывает желаемое поведение в понятном человеку формате с использованием синтаксиса Gherkin.

(Поваров, 2022)

|  |
| --- |
| Feature: Rectangle Area Calculation  Scenario: Calculating the area of a rectangle  Given a rectangle with width 5 and height 10  When I calculate the area  Then the area should be 50  Scenario: Calculating area with zero width  Given a rectangle with width 0 and height 10  When I calculate the area  Then the area should be 0  Scenario: Calculating area with zero height  Given a rectangle with width 5 and height 0  When I calculate the area  Then the area should be 0  Scenario Outline: Calculating area with various dimensions  Given a rectangle with width <width> and height <height>  When I calculate the area  Then the area should be <area>  Examples:  | width | height | area |  | 2 | 3 | 6 |  | 7 | 0 | 0 |  | 0 | 4 | 0 |  | -1 | 5 | 0 | #Error handling scenario |

(OpenAI, 2024)

Методы Java реализуют шаги, описанные в файле функций. Они взаимодействуют с нашим Rectangle классом. Для тестов мы будем использовать JUnit 5.

|  |
| --- |
| Тест |
| import io.cucumber.java.en.Given;  import io.cucumber.java.en.When;  import io.cucumber.java.en.Then;  import org.junit.jupiter.api.Assertions;  public class RectangleSteps {  private Rectangle rectangle;  private int area;  @Given("a rectangle with width {int} and height {int}")  public void aRectangleWithWidthAndHeight(int width, int height) {  try{  rectangle = new Rectangle(width, height);  } catch (IllegalArgumentException e){  *//Handle exception appropriately, perhaps logging or modifying the test to expect an exception.*  System.out.println("Exception handled in step definition: "+ e.getMessage());  area = 0; *//Set area to 0 in case of exception*  }  }  @When("I calculate the area")  public void iCalculateTheArea() {  area = rectangle.getArea();  }  @Then("the area should be {int}")  public void theAreaShouldBe(int expectedArea) {  Assertions.assertEquals(expectedArea, area);  }  } |
| Код |
| public class Rectangle {  private final int width;  private final int height;  public Rectangle(int width, int height) {  if (width < 0 || height < 0) {  throw new IllegalArgumentException("Width and height must be non-negative");  }  this.width = width;  this.height = height;  }  public int getArea() {  return width \* height;  }  public int getPerimeter() {  return 2\*(width+height);  }  } |

(OpenAI, 2024)

### Web-UI

WebDriver API очень простой. Команды примитивные — только те, которые вы можете формировать у себя с клавиатуры и мышки. Вы можете на что-то кликнуть или ввести какой-то текст. Все. Сложность в использовании WebDriver — это найти на странице того, по кому надо кликнуть или куда ввести. И для этого используются локаторы.

(Urikh, 2024)

# Практика

# Список литературы

1. BDD, Ш. И. (2022). Получено из https://masters.donntu.ru/2019/fknt/shevelov/diss/index.html
2. Gupta, L. (29 11 2023 г.). *TDD Java*. Получено из EDUCBA: https://www.educba.com/tdd-java/
3. OpenAI. (2024). *ChatGPT*. Получено из ChatGPT 4o: https://chatgptchatapp.com
4. rmksrv. (18 4 2024 г.). *Введение в BDD*. Получено из auriga: https://habr.com/ru/companies/auriga/articles/808527/
5. TDD. (2025). *TDD*. Получено из https://paitowarna.serv00.net/benefits-of-test-driven-development-in-devops-environments-blog-/
6. Urikh, G. (02 05 2024 г.). *Тестирование в Java: лучшие практики, инструменты и рекомендации для разработчиков*. Получено из proglib: https://proglib.io/p/testirovanie-v-java-luchshie-praktiki-instrumenty-i-rekomendacii-dlya-razrabotchikov-2024-05-02
7. Поваров, Е. (10 3 2022 г.). *BDD тестирование*. Получено из smartiqa: https://smartiqa.ru/blog/bdd\_gherkin\_cucumber