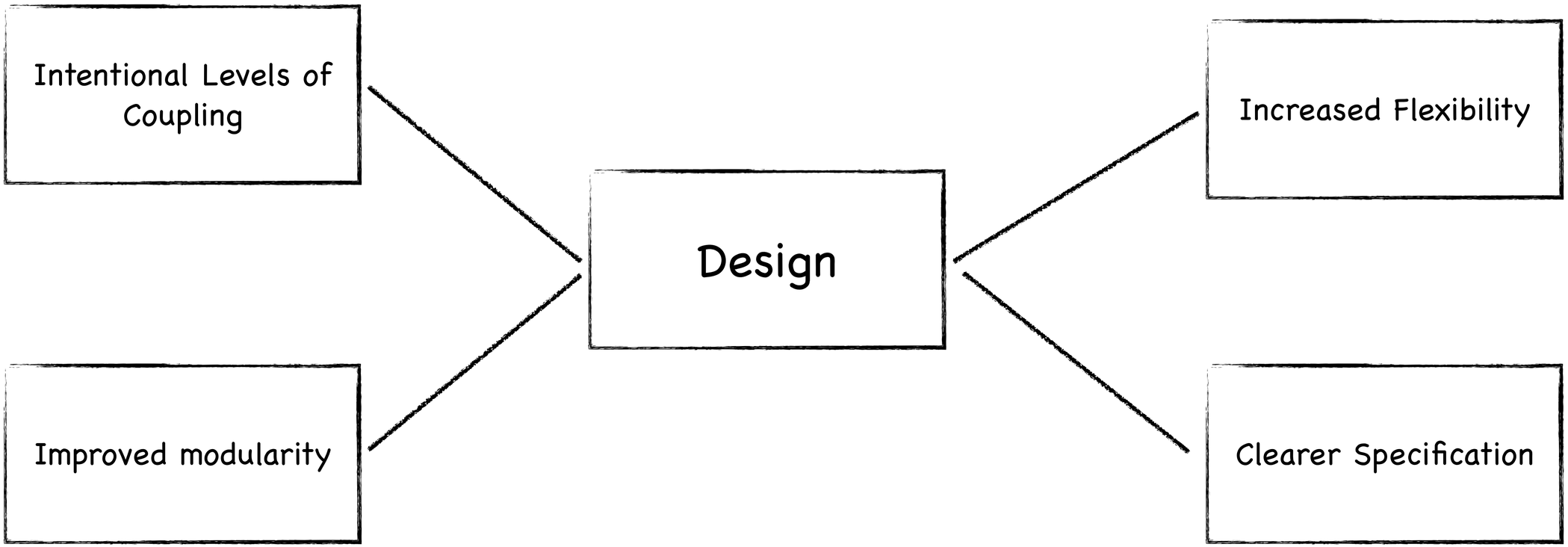
Advent of TDD Introduction

Engineering Excellence

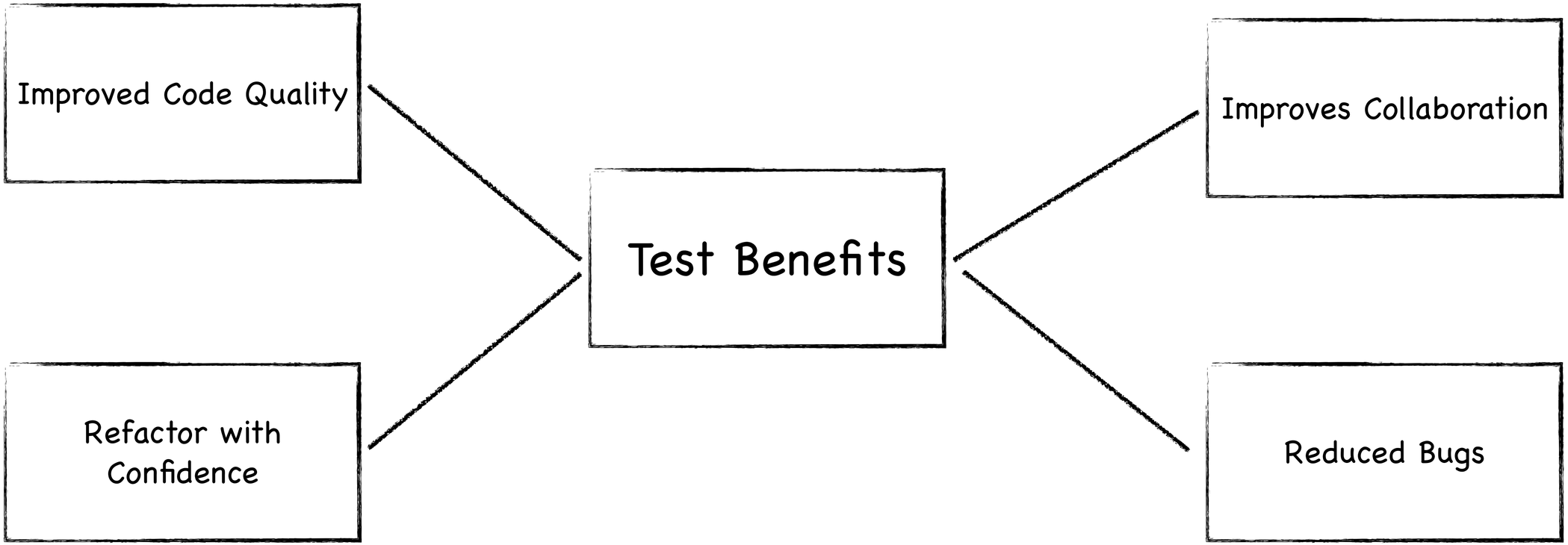
Agenda

- Benefits of Test Driven Development
- The Basics
- Styles of TDD
- Unit Tests
- Spikes
- Mocking (in Java)
- Workshop Outline

Benefits of Test Driven Development



Benefits of Test Driven Development



The Basics

- **Red**
  - Write the test
  - Fix compilation issues (auto generate)
  - Prove the test fails
- **Green**
  - Make the test pass
- **Refactor**
  - Improve the subject under test
  - Improve the test

Styles of TDD

- Outside-In
  - What does the caller need, what is the contract for interaction?
  - Start there and work in
- Inside-Out
  - Start with the lower levels
  - Start there and work outward

Unit Tests

- An isolated subject under test - e.g. a single `public method` in a `class`
- Produce repeatable, isolated and fast results
- Contain `assertions` to validate behaviour
- Supported by a Test Framework e.g. JUnit
- Start with `edge` and `exception cases`

Spikes

- Test Driven Development is hard
- Working on two things: The Test, The Subject Under Test
- Too much discovery is an overload
- A spike can help
  - Create a new branch
  - Experiment without tests
  - Hack something together
- Return to test informed and with the first test idea

Mocking

- Used to isolate dependencies and come in different styles
- **Dummy** - Not used in the test
- **Stubs** - Return pre-defined values **when** something happens
- **Spy** - Check or **verify** a mock was called with specific values
- **Fake** - Could be a hand crafted mock, useful in integration tests
- Mocking is required to isolate and effectively write unit tests
- Mocks define intentional seams and interactions between code

Begin with Edge Cases - Red

```
public class TestTodoListShould {
    private TodoList todoList;

    @BeforeEach
    public void setup() {
        todoList = new TodoList();
    }

    @Test
    void throw_an_exception_when_todo_is_empty() {
        assertThrowsExactly(RuntimeException.class, () -> todoList.saveTask(""));
    }
}
```

Begin with Edge Cases - Green

```
public class TodoList {
    public void saveTask(String task) {
        if(task == null || task.isEmpty()) {
            throw new RuntimeException("Yikes - You're busy doing nothing again");
        }
    }
}
```

Save Task - Red

```java
@ExtendWith(MockitoExtension.class)
public class TestTodoListShould {
    private TodoList todoList;

    @Test
    void throw_an_exception_when_todo_is_empty(@Mock TaskStore taskStore) {
        todoList = new TodoList(taskStore);
        assertThrowsExactly(RuntimeException.class, () -> todoList.saveTask(""));
    }

    @Test
    void store_a_valid_task_and_return_an_id(@Mock TaskStore taskStore) {
        todoList = new TodoList(taskStore);
        when(taskStore.persistTask("Workshop")).thenReturn(1);
        assertThat(todoList.saveTask("Workshop"), equalTo(1));
```

Save Task - Green

```
public class TodoList {
    private final TaskStore taskStore;

    public TodoList(TaskStore taskStore) {
        this.taskStore = taskStore;
    }

    public int saveTask(String task) {
        if(task == null || task.isEmpty()) {
            throw new RuntimeException("Yikes - You're busy doing nothing again");
        }
        return taskStore.persistTask(task);
    }
}
```

Workshop Outline

- https://github.com/jpgough/advent-of-tdd
- Exercise 1: Unit testing Elfs
- Exercise 2: Spiking a Java complexity
- Exercise 3: Mocking and testing File Input (You may soon need this most days)
- Exercise 4: Second Star - new requirements