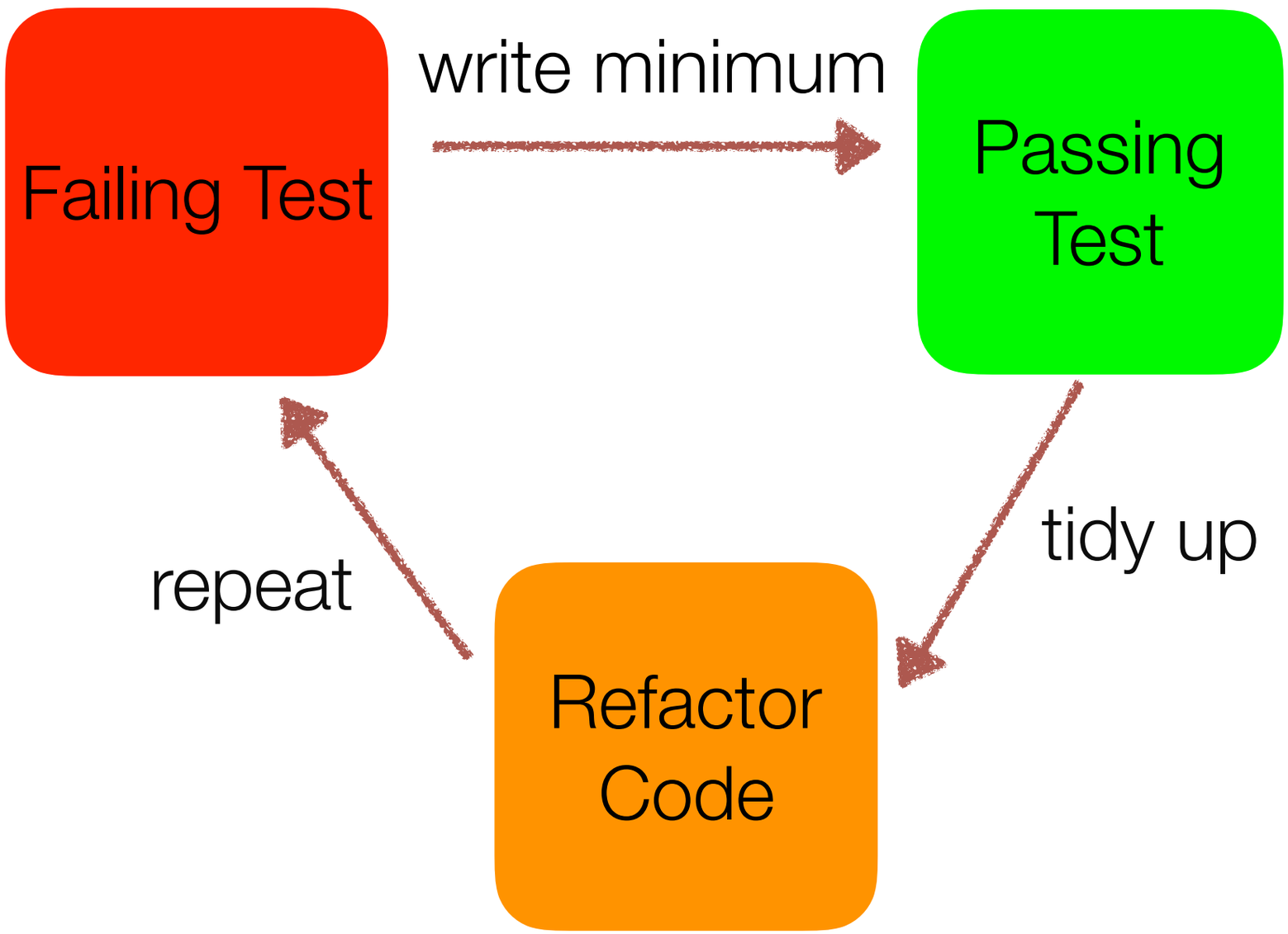
Georgina McFadyen
@gemcfadyen

Introduction

- Software Crafter at 8th Light
- @gemcfadyen

Test Driven Development



Types of TDD

ATDD (Acceptance Test Driven Development)

Inside Out

Bottom Up

Classic School

Top Down

Outside In

BTDD (Behavioural Test Driven Development)

Mockist Approach

London School

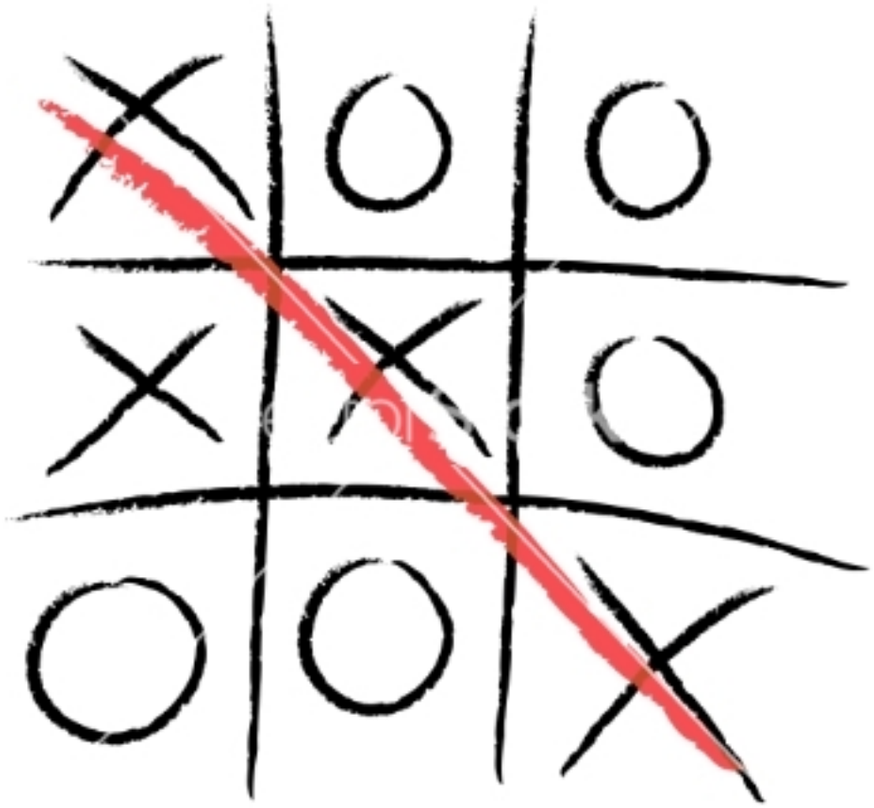
Inside Out

Inside Out

- Focus on one entity (module/class) at a time
- Integrate later
- Parallelise work

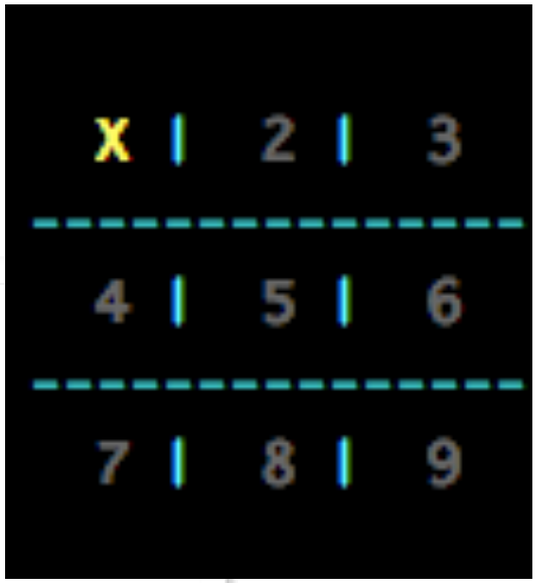
Inside Out Example

- Identify individual entities - Board, UserInterface, Game
- Build out each one at a time



Inside Out Example

- Board
  - Needs to be updated with players move




```
public class BoardTest {

    @Test
    public void updateBoardWithUsersMove() {
        Board board = new Board("- - - " +
                                "- - - " +
                                "- - -");

        Board updatedBoard = board.update(1, X);

        assertThat(updatedBoard.symbolAt(1), is(X));
    }
}
```

Inside Out Example

- Board
  - Identify winning combinations

```
X | 2 | O
-------------
X | 5 | O
-------------
X | 8 | 9
```

```java
@Test
public void hasWinningColumn() {
    Board board = new Board("X - - " +
                            "X - - " +
                            "X - -");

    boolean hasWinningLine = board.hasWinningLine();

    assertThat(hasWinningLine, is(true));
}
```

Inside Out Example

- UserInterface (Prompt)

```
Please enter move:
```

```java
public class PromptTest {

    @Test
    public void promptsForNextMove() {
        Writer writer = new StringWriter();
        Prompt prompt = new Prompt(writer);

        prompt.askForNextMove();

        assertThat(writer.toString(), is("Please enter move:"));
    }
}
```

Inside Out Example

- Game - All pieces must fit together

```
public class GameTest {

    @Test
    public void gameIsOverWhenWinningMoveMade() {
      Writer writer = new StringWriter();
      Reader reader = new StringReader("6\n");
      Prompt prompt = new Prompt(reader, writer);

      Game game = new Game(prompt, new Board("X - - " +
                                             "X O O " +
                                             "- - -"));

      Board updatedBoard = game.play();

      assertThat(writer.toString(), containsString("Please enter move:"));
      assertThat(writer.toString(), containsString("Player X won!"));
      assertThat(updatedBoard.symbolAt(6), is(X));
    }
}
```

Inside Out Example

- Update Board
  - Identify winning symbol

```
@Test
public void hasCorrectWinningSymbol() {
    Board board = new Board("X - - " +
                            "X - - " +
                            "X - -");

    PlayerSymbol winningSymbol = board.getWinningSymbol();

    assertThat(winningSymbol, is(PlayerSymbol.X));
}
```

Inside Out Example

- Only need to identify one entity to get started
- Inner details emerge
- May end up exposing more or less behaviour than required

Outside In

Outside In

- Defined route through the system from the start
- Knowledge of how different parts of the system interact
- Often leverage testing doubles, mocks
- Start with a failing high level acceptance test

Outside In Example

- Given the winning move is taken, the game should display a winning message.

```
 X | X | X
--------------
 O | 5 | O
--------------
 7 | 8 | 9

Congratulations - X has won
```

```java
public class GameTest {

    @Test
    public void gameShouldEndIfThereIsAWinningRowInTheGrid() {
        Board board = new Board("X X - " +
                                "O - O " +
                                "- - -");

        PromptSpy promptSpy = new PromptSpy("2");
        Game game = new Game(promptSpy, board);

        game.play();

        assertThat(promptSpy.hasAnnouncedWinner(), is(true));
    }
```

Outside In Example

- Identified the need for several entities

```
public class Game {

    private Prompt prompt;

    public Game(Prompt prompt, Board board) {
        this.prompt = prompt;
    }

    public void play() {
        prompt.displaysWinningMessageFor(X);
    }
}
```

Outside In Example

```java
public class GameTest {

    @Test
    public void gameShouldEndIfThereIsAWinningRowInTheGrid() {
        Board board = new Board("X X - " +
                                "O - O " +
                                "- - -");

        PromptSpy promptSpy = new PromptSpy("2");
        Game game = new Game(promptSpy, board);

        game.play();

        assertThat(promptSpy.hasAnnouncedWinn
    }
```

```java
class PromptSpy implements Prompt {
    boolean hasDisplayedWinningMessage = false;

    public PromptSpy(String playersMove) {
    }

    public void displaysWinningMessage() {
        hasDisplayedWinningMessage = true;
    }

    public boolean hasAnnouncedWinner() {
        return hasDisplayedWinningMessage;
    }
}
```

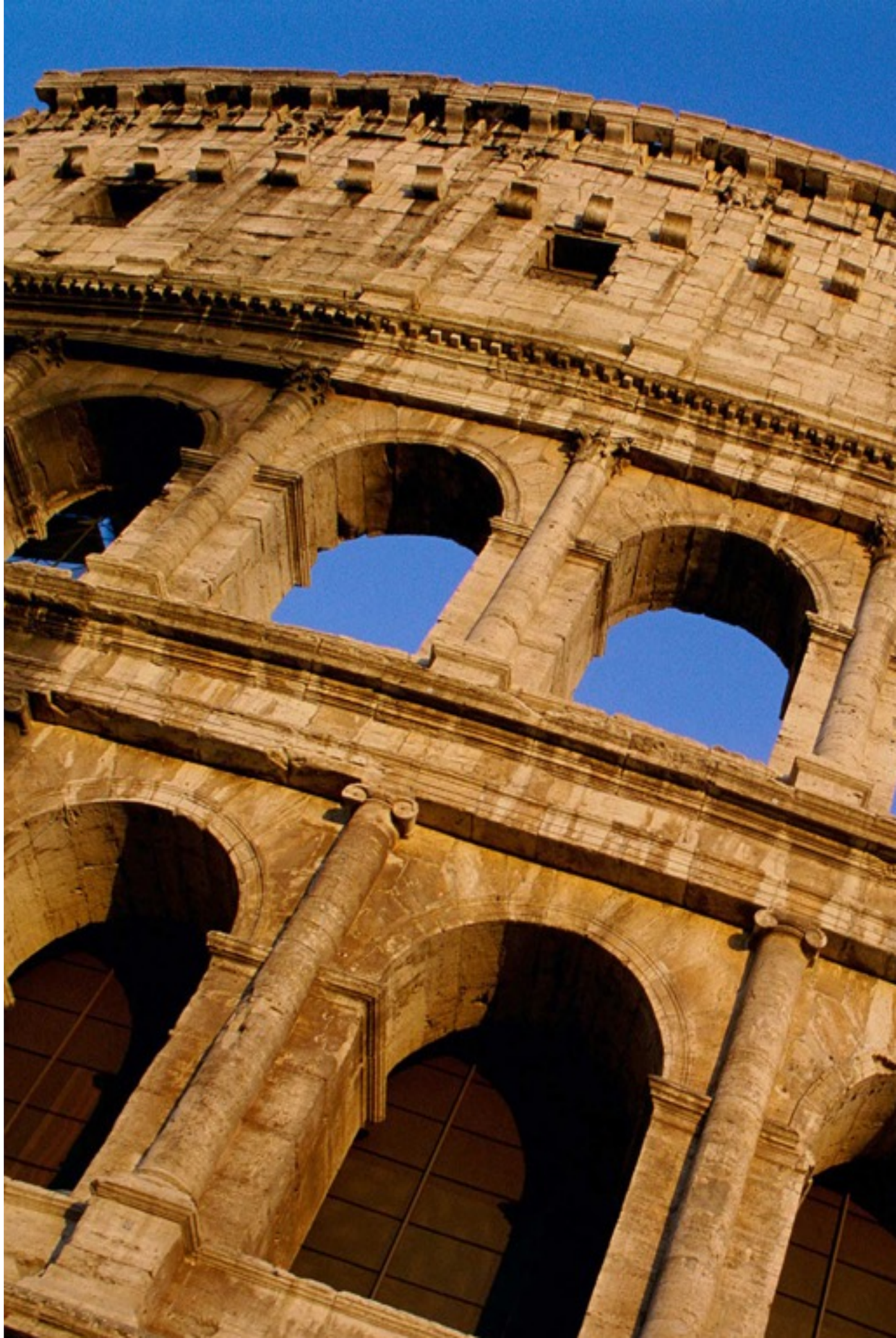
Outside In Example

```java
@Test
  public void playersTakeTurnsUntilTheGameIsDrawn() {
    String seriesOfMoves = "1\n2\n5\n3\n6\n4\n7\n9\n8\n";
    PromptSpy promptSpy = new PromptSpy(seriesOfMoves);

    Board board = new Board("- - - " +
                            "- - - " +
                            "- - -");
    Game game = new Game(promptSpy, board);

    game.play();

    assertThat(promptSpy.hasAnnouncedDraw(), is(true));
    assertThat(promptSpy.numberOfTimesPlayersPromptedForMove(), is(9));
  }
```

Outside In TDD

- Knowledge of how entities communicate
- Interactions rather than internal details
- Tests can be tightly tied to implementation details

System Design



Customer Feedback

Language Proficiency

Test Doubles

Conclusion

- Both approaches form part of your toolkit
- Inside Out
  - Can be easier to get started
  - Methodical
- Outside In
  - Demonstrable route through system early on
  - Driven from user requested scenarios

Resources

- https://8thlight.com/blog/georgina-mcfadyen/