More TDD

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

At the end of Today's Lecture you will be able to:

- Understand the requirements for good tests.
- Understand the basic flow of TDD
- Use TDD in practice

ROAR

Inspiration

Experience is a hard teacher because she gives the test first, the lesson afterward

Idaho State University
Computer Science

Running Example

General Problem Statement

Build a subsystem for an email application
Allow users to use **email templates** to create personalized responses for repeated email messages

Example

Teacher sends and email:

```
Hello <student>,

Please read the syllabus.
```


ROAR

Idaho State University
Computer Science

From Requirements to Tests

Template System as Tasks

- Write a regular expression to identify variables from the template
- Implement a template parser that uses the regex
- Implement a template engine that provides a public API

Template System as Tests

- Template without any variables renders as is
- Template with one variable is rendered with variables replaces by value
- Template with multiple variables is rendered with each variable replaced by an appropriate value

Which approach do you find more natural?

ROAR

What Makes a Good Test?

- A good test is **atomic**
  - Does one and only one thing
  - Keeps things focused
- A good test is **isolated**
  - Does not depend on other tests
  - Does not affect other tests

This is not a complete list, but a start

Programming by Intention

- Given an initial set of tests
  - Pick one
  - Goal: **Most progress** with least effort
- Next, write test code
  - Wait! Code won't compile!
  - Imagine code exists
  - Use most natural expression for call (design the API)
- Benefit of **programming by intention**
  - Focus on what we COULD have
  - Not what we DO have

Evolutionary API design from client perspective

Choosing the First Test

- Some detailed requirements:
  - System replaces variable placeholders like `${firstname}` in template with values provided at runtime
  - Sending template with undefined variables raises error
  - System ignores variables that aren't in the template
- Some corresponding tests:
  - Evaluating template `"Hello, ${name}"` with value `name = Reader` results in `"Hello, Reader"`
  - Evaluating `"${greeting}, ${name}"` with `"Hi"` and `"Reader"` results in `"Hi, Reader"`
  - Evaluating `"Hello, ${name}"` with `"name"` undefined raises `MissingValue`

Writing the first (failing) test

- Evaluating template `"Hello, ${name}"` with value `Reader` results in `"Hello, Reader"`

```
public class TestTemplate {
  @Test
  public void oneVariable() throws Exception {
    Template template = new Template("Hello, ${name}");
    template.set("name", "Reader");
    assertEquals("Hello, Reader", template.evaluate())
  }
}
```

- Design Decisions:
  - Class name: `Template`
  - `Template` API: `set()`, `evaluate()`

Try this on your computer

Idaho State University
Computer Science

Code to Make Compiler Happy

```
public class Template {
  public Template(String templateText) {
  }
  public void set(String variable, String value) {
  }
  public String evaluate() {
    return null;
  }
}
```

- This allows the test to **compile**
- The test **fails**, of course
- Running it should result in a RED bar
- We're at the RED part of RED-GREEN-REFACTOR

Making the First Test Pass

```
public class Template {
  public Template(String templateText) {
  }
  public void set(String variable, String value) {
  }
  public String evaluate() {
    return "Hello, Reader"; // min to make test pass
  }
}
```

- We're looking for the **green bar**
- We know this code **will change later** - That's fine
- 3 dimensions to push out code: **variable**, **value**, **template**

Second Test

- Purpose of 2nd test is to "**drive out**" hard coding of variable's value
- This is called **triangulation**

```
public class TestTemplate {
  @Test
  public void oneVariable() throws Exception {
    Template template = new Template("Hello, ${name}");
    template.set("name", "Reader");
    assertEquals("Hello, Reader", template.evaluate())
  }
  @Test
  public void differentValue() throws Exception {
    Template template = new Template("Hello, ${name}");
    template.set("name", "someone else");
    assertEquals("Hello, someone else", template.evaluate());
```

Making the 2nd Test Pass

- Revised code

```
public class Template {
  private String variableValue;
  public Template(String templateText) {
  }
  public void set(String variable, String value) {
    this.variableValue = value;
  }
  public String evaluate() {
    return "Hello, " + variableValue;
  }
}
```

Third Test

- Note revisions to JUnit test to squeeze out more hard coded values

```
public class TestTemplate {
  @Test
  public void oneVariable() throws Exception {
    Template template = new Template("Hello, ${name}");
    template.set("name", "Reader");
    assertEquals("Hello, Reader", template.evaluate())
  }
  @Test
  public void differentTemplate() throws Exception {
    Template template = new Template("Hi, ${name}");
    template.set("name", "someone else");
    assertEquals("Hi, someone else", template.evaluate());
  }
}
```

Breadth-first, depth-first

- What do do with a “hard” red bar?
- Issue is **what to fake** vs. **what to build**
- “Faking” is an accepted term in TDD that means “**deferring a design decision**”
- **Depth first** means supplying detailed functionality
- **Breadth first** means covering end-to-end functionality (even if part is faked)

Idaho State University
Computer Science

Making the 3rd Test Pass

```
public class Template {
  private String variableValue;
  private String templateText;

  public Template(String templateText) {
    this.templateText = templateText;
  }
  public void set(String variable, String value) {
    this.variableValue = value;
  }
  public String evaluate() {
    return templateText.replaceAll("\\$\\{name\\}", variableValue);
  }
}
```


ROAR

Idaho State University
Computer Science

Test 4: Multiple Variables

- A new test with more than one variable

```
@Test
public void multipleVariables() throws Exception {
  Template template = new Template("${one}, ${two}, ${three}");
  template.set("one", "1");
  template.set("two", "2");
  template.set("three", "3");
  assertEquals("1, 2, 3", template.evaluate());
}
```


ROAR

Making Test 4 Pass

```
public class Template {
  private Map<String, String> variables;
  private String templateText;

  public Template(String templateText) {
    this.variables = new HashMap<>();
    this.templateText = templateText;
  }
  public void set(String variable, String value) {
    this.variables.put(varoab;e, value);
  }
  public String evaluate() {
    String result = templateText;
    for (Entry<String, String> entry : variables.entrySet()) {
      String regex = "\\$\\{" + entry.getKey() + "\\}";
      result = result.replaceAll(regex, entry.getValue());
    }
    return result;
```

Special Test Case

- Special case of a variable that does not exist
  - Variable should simply be ignored
- This test passes for free!

```
@Test
public void unknownVariablesAreIgnored() throws Exception {
  Template template = new Template("Hello, ${name}");
  template.set("name", "Reader");
  template.set("doesnotexist", "Hi");
  assertEquals("Hello, Reader", template.evaluate());
}
```

Are there any questions?