Beginning TDD

Isaac Griffith

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

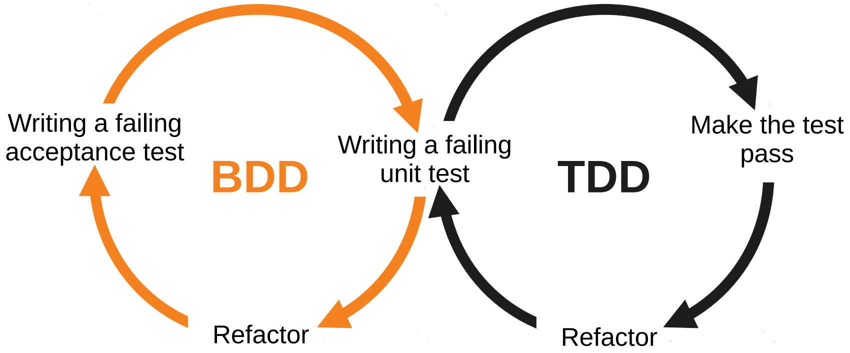
Outcomes

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

Inspiration

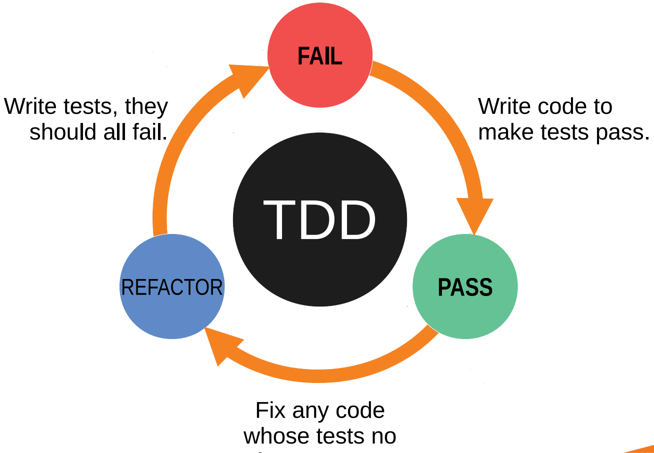
Experience is a hard teacher because she gives the test first, the lesson afterward


Writing a failing acceptance test
BDD
Writing a failing unit test
TDD
Make the test pass
Refactor
Refactor

Test Driven Development

Build It Right: TDD

- Test-Code-Refactor: The heart-beat
  - The rule: **Only write code to fix a failing test**
  - Test-driven development cycle (**red-green-refactor**)


FAIL
Write code to make tests pass.
PASS
Fix any code whose tests no
REFACTOR
Write tests, they should all fail.
TDD

Idaho State University
Computer Science

Build It Right: TDD

- First, we write a test
- This really amounts to design by example
  - We make decisions about how the **Application Programmer Interface (API)** works
    - Class name, method names, return results, etc.
    - This is essentially the user interface
  - We're thinking hard about how code is used
  - We're taking a client perspective
  - We're working at a very small scale
- Example for a stack:

```
stack = ...;
stack.push(x);
y = stack.pop();
assertEquals(x, y);
```

Start with one concrete client interaction

Build It Right: TDD

- Then we write **just enough** code
  - We don’t write more code
  - All we want is to make the test pass
    - It should be a very small step
    - Implementation probably not optimal
    - We don’t care (yet)

Goal: Make code base (just) pass test suite

Build It Right: TDD

- And then we refactor
- TDD without refactoring just makes ugly code
  - Maintenance debt
- We have numerous transformations to address this
- Developing with small steps
  - The code always runs!
    - Changes are small enough to fit in our heads
    - Time-frame is minutes to (maybe) hours
  - Evolutionary design
    - Anticipated vs. unanticipated changes
    - Many “anticipated changes” turn out to be unnecessary

New ways to apply standard lessons

Build It Right: TDD

- Keeping code healthy with refactoring

Definition

Refactoring
A disciplined technique for restructuring an existing body of code, and altering its internal structure without changing its external behavior.

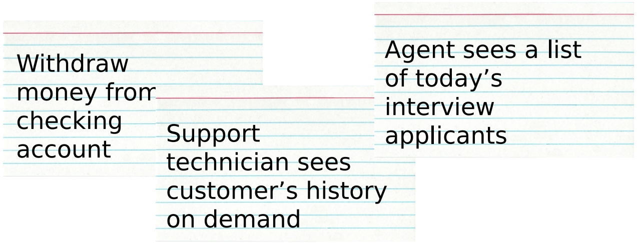
- Refactoring is disciplined
  - Wait for a problem before solving it
- Refactorings are transformations
  - Many refactorings are simply applications of patterns
- Refactorings alter internal structure
- Refactorings preserve behavior

Focus is on current code, not future code

User Stories

Definition

A **User Story** is a few sentences that capture what a user will do with the software



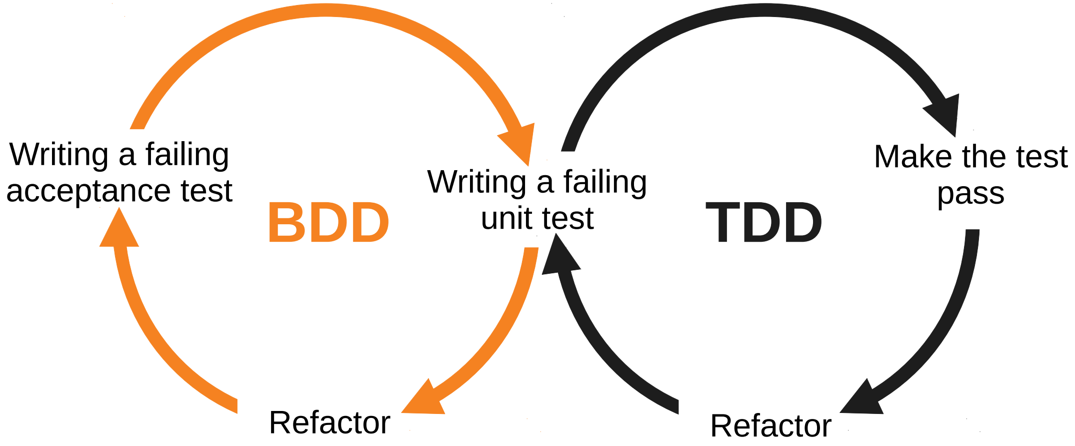
Withdraw money from checking account

Support technician sees customer's history on demand

Agent sees a list of today's interview applicants

- In the language of the **end user**
- Usually small in scale with **few details**
- **Not** archived

Acceptance Tests in Agile Methods


Writing a failing acceptance test
BDD
Writing a failing unit test
TDD
Make the test pass
Refactor
Refactor

The Testing Shortfall

- Do **TDD tests** (acceptance or otherwise) test the software well?
  - Do the tests achieve good **coverage** on the code?
  - Do the tests find most of the **faults**?
  - If the software passes, should management feel confident the software is **reliable**?

NO!

Why Not?

- Most agile tests focus on “happy paths”
  - What should happen under normal use
- They often miss things like
  - **Confused**-user paths
  - **Creative**-user paths
  - **Malicious**-user paths

The agile methods literature does not give much guidance

Take Small Steps

- More companies are putting **testing first**
- This can dramatically **decrease cost** and **increase quality**
- A different view of "**correctness**"
  - Restricted but practical
- Embraces **evolutionary design**
- TDD is definitely **not** test automation
  - Test automation is a **prerequisite** to TDD
- **TDD tests** aren't enough

Are there any questions?