Acceptance TDD Explained

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:

- Able to write User Stories
- Develop Acceptance Tests
- Understanding the process
- Understand Acceptance TDD as a team activity
- Understand the Benefits of acceptance TDD
- Provide a brief overview of available tools
- Understand and use Test Doubles

Inspiration

“In the spacecraft business no design can survive the review process without first answering the question—how are we going to test this thing?” – Glen Alleman

Introduction to User Stories

- Format of a story
  - Free form
  - Or structured: As a (**role**) I want (**functionality**) so that (**benefit**)
  - Often written on index cards
- Card, conversation, confirmation (CCC)
- Power of storytelling
  - User view of **what** is needed, but not **how** it is provided
- A user story **represents** a requirement, and creates a **promise** to communicate with the customer later

Storytelling reveals meaning without defining it – Hannah Arendt

Example User Stories

- Support technician sees customer’s history on-screen at the start of a call
- Application authenticates with the HTTP proxy server
- The system prevents user from running multiple instances of the application

We State **what**, NOT **how**

Enabling value: A user story is valuable because it enables engineers to add functionality.

Acceptance Tests

- Create tests based on user stories
- Properties of user stories
  - Owned by customer
  - Written together with customer, developer, and tester
  - Focus on the **what**, not the **how**
  - Expressed in language of the problem domain—user's vocabulary
  - Concise, precise, and unambiguous

In-Class Exercise

Discussion

Do the following User Stories:

- Support technician sees customer's history on-screen at the start of a call
- Application authenticates with the HTTP proxy server
- The system prevents user from running multiple instances of the application

Satisfy the following properties:

- Focus on the **what**, not the **how**
- Expressed in language of the problem domain–user's vocabulary
- Concise, precise, and unambiguous

Acceptance Tests–Example Tests

- Support technician sees customer's history on-screen at the start of a call
- Tests:
  - Simulate a call with Fred's account number and verify that Fred's info can be read from the screen
  - Verify that the system displays a valid error message for a non-existing account number
  - Omit the account number in the incoming call completely and verify that the system displays the text "no account number provided" on the screen

What vs. How

❶ Go to the “new transaction” screen, fill in the required details, and save the entry; verify that the transaction shows up on the list

❷ Select the “delete” checkbox for the newly created entry, click “delete all marked transactions,” and verify that they’re gone

❸ Create multiple transactions, check several of them and delete; verify that all selected transactions were indeed deleted

In-Class Discussion:

What is wrong with these tests?

What vs. How

User Story:

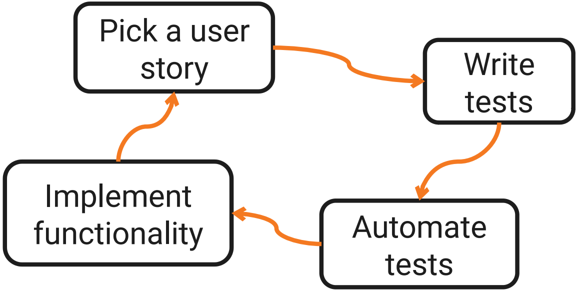
- Support technician sees customer’s history on-screen at the start of a call

- Tests:
  - ❶ Simulate a call with Fred’s account number and verify that Fred’s info can be read from the screen
  - ❷ Verify that the system displays a valid error message for a non-existing account number
  - ❸ Omit the account number in the incoming call completely and verify that the system displays the text “no account number provided” on the screen

- Too Detailed
  - Trimmed Versions
    - ❶ Valid account number
    - ❷ Non-existing account number
    - ❸ No account number provided

Understanding the Process

- The ATDD cycle
  1. Pick a story
  2. Write tests for the story
  3. Automate the tests
  4. Implement the functionality



A process with feedback

ATDD Process Step 1

- The ATDD Cycle
  1. Pick a story (which story?)
     - Most important
     - Business value
     - Technical risk
     - Amount of programming
  2. Write tests for the story
  3. Automate the tests
  4. Implement the functionality

ATDD Process Step 2

- The ATDD Cycle
    1. Pick a story
    2. Write tests for the story
        - Involve the customer
        - Iterate
        - Keep abstract as long as possible
        - Get ahead of refactoring
    3. Automate the tests
    4. Implement the functionality

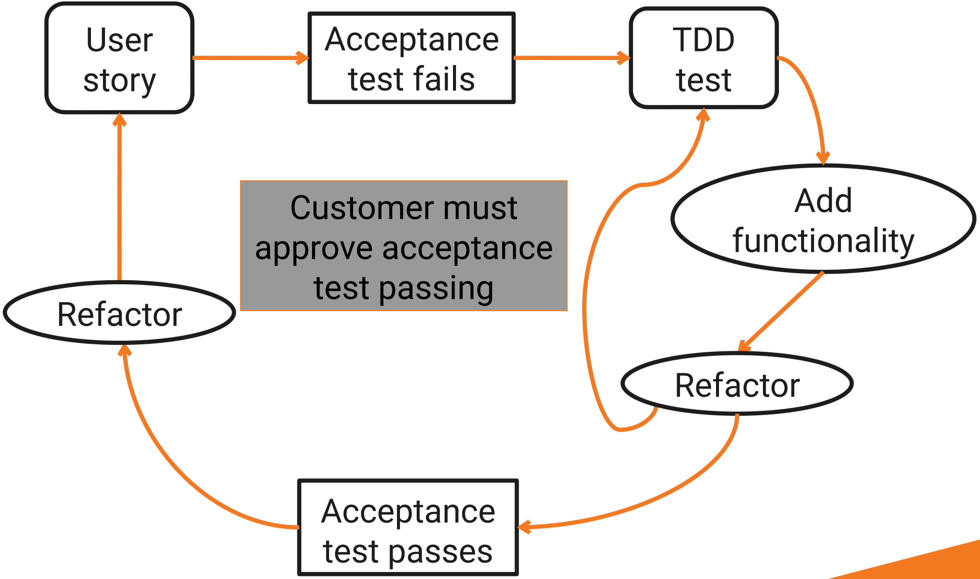
ATDD Process Step 3

- The ATDD Cycle
  1. Pick a story
  2. Write tests for the story
  3. Automate the Tests
     - Start with a table format
     - Translate to implementation
     - Postpone use of tools – tools steal focus from topic
  4. Implement the functionality

ATDD Process Step 4

- The ATDD Cycle
  1. Pick a story
  2. Write tests for the story
  3. Automate the tests
  4. Implement the functionality
     - Each ATDD test leads to multiple small tests

Acceptance Test in Agile



ATDD as a Team Activity

- Defining the customer role
  - Representative of end users
  - Possibly several people
- Characteristics of customer role
  - Shared interest in success
  - Authority to make decisions
  - Ability to understand implications
  - Ability to explain domain

Key is to verify against target domain

Acceptance Testing Team

- Who writes tests with the customer?
  - Tester?
  - Developer?
  - Requirements expert?
  - Everybody?
- How many testers do we need?
  - One or two developers per tester
  - Tester is a role, not a job title
  - All developers should be testers

More contributors is better

Benefits of ATDD

- Definition of “done”
  - Customer must agree it’s done
  - Knowing where we are
  - Knowing when to stop
  - Test criteria satisfied
- Cooperative work
- Trust and commitment
- Specification by example
  - This is a big one!
- Filling the gap
  - Unit tests are not the same as acceptance tests

Both unit and acceptance tests needed

What are We Testing, Exactly?

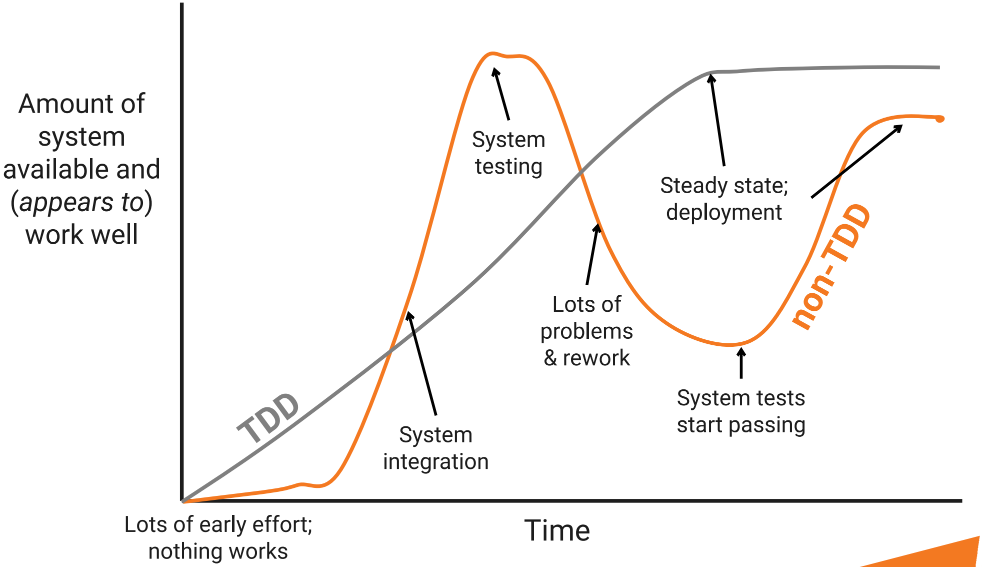
- Should we test against the UI?
  - Do whatever is easier long term
  - UIs are often in the way
  - Good tools can automate tests through or around the UI
  - Performance might matter
- Should we stub our system?
  - Sufficiently close to the real thing
  - Sometimes stubs are necessary
- Should we test business logic directly?
  - Of course—it's what the customer cares about

Tests are like votes—they need to run early and often

Test Double “Rules”

- What’s a “Test Double”?
- Should you **edit** code to control program behavior at test time?
  - NO! The change in behavior should be dynamic. Why?
- A **seam** is a special variable that can be set from inside a test.
  - The seam controls behavior and is internal to the component under test
- An **enabling point** is a location where it is possible to set a seam to the desired value
  - sometimes called controlling the seam
  - also usually in the component under test
  - should **not** be part of the public API. Why not?
- A test **exploits the seam** by using the enabling point
- The terminology sounds borrowed from the security domain. Why is that?

Summary



Lots of early effort; nothing works

System integration

System testing

Lots of problems & rework

System tests start passing

Steady state; deployment

Are there any questions?

ROAR