BDD and Testing

User requirements and testing are tightly coupled
New Concept: **Acceptance Tests**

- **Customer criteria for accepting a milestone**
  - Get paid if pass!

- **Black-box** tests specified with the customer
  - Agreed upon features: User Stories!
  - Performance requirements (speed, scalability)

- *(System tests are like acceptance tests, but are spec’d by the developers for own use)*

- Customer may have no idea how to write good acceptance tests
  - We have to **collaborate to do it well**
Writing Good Tests is Hard

How many tests do we need to write to test the `add` method?

```java
// perform arithmetic “+” of x and y
public int add(int x, int y) {
    return x + y;
}
```

A. One
B. About four
C. About 20
D. About $2^{32} \times 2^{32}$
You can’t test all combinations

- Simply too many tests ($2^{64}$ for measly `add` !)

- If there is one bug in `add`:
  - Most tests will pass
  - Another set of tests will fail, all the same way
  - Most tests, then, are unnecessary (whew!)

- Ideally, have only:
  - 1 test that succeeds (shows most of it works)
  - 1 test that exposes the bug

- But doing that well requires knowing what the bug is (and that there’s only one!)

- ...Or does it?
New Concept: **Cluster tests into similar classes**: one (or a few) tests per class

```java
public int add(int x, int y) {
    return x + y;
}
```

- add(-5, -100) → -105  
  two negatives

- add(5, 100) → 105  
  two positives

- add(100, -5) → 95  
  positive, negative

- add(-100, 5) → -95  
  negative, positive

- add(0, 5) → 5  
  special value: 0

- add(-1, 1) → 0  
  special values: near 0

- add(Integer.MAX_INT, -1) → 2^32 - 2  
  boundary val: max int

These are really **classes of coding errors**, not just classes of values

- Handling neg’s, handling pos’s, handling mixed neg/pos, handling 0…)

- Each is a hypothesis (guess) about how `add` might be coded wrong

- Classes can also be based on outputs, not just inputs
What are the classes for myCity acceptance tests?

- Not so simple
  - `add` is stateless
  - myCity is all about state

- To test “send message to friend on map”:
  - Have to be logged in to retrieve friends
  - User has to have a location
  - Friend has to have a location
  - Need to be close to each other (friend on the map)

- Also, not natural for customer to talk in terms of value classes or even classes of coding errors
  - Different kinds of buddies (logged in, logged out, near, far)
  - Different kinds of locations (San Diego, LA, …huh?)
  - …
The Answer: **Scenarios** (new concept)

- The customer naturally thinks in terms of **situations** that users encounter when using app
  - New user, returning user, forgot password, …
  - Can’t acquire GPS location
  - No buddies close by
  - …

- Each situation is the beginning of a “story”
  - Not a User Story (describes a **feature**)
    - “User Story” is a misnomer
  - A personal story called a **scenario**
  - Progresses step-by-step, driven by state of the world, state of the app, and motivations of the user
Bob was looking for a lunch partner, so he decided to login to myCity. Clicking its icon, he was taken directly to the map; it had remembered his login and password. After a few moments, the map changed to his current location. Good. However, no buddies were shown in the city block shown. He didn’t mind walking a little, so he clicked the “-” on the map to zoom out. Now he could see Alice on the map....

Key elements:
- Bob’s thoughts: motivations, preferences, etc.
- State of the app: remembered login info
- Progression: determines location, updates map
- State of the world: no one nearby

Their combination “drive” the scenario forward
Scenarios are a Powerful Tool

- In **non-technical** language for customer
  - Structure of a story good match for our brains

- Help **uncover requirements**
  - Need to support zoom!

- Implicitly **highlight benefit** of app’s features

- Exhibit common **real-world progressions**
  - **Omit** trillions of **unlikely progressions** (i.e., tests)
  - Identifies **testing classes**!
    - **Buddy-to-message distance**: on map, off map
Behavior-Driven Development (BDD)

- Powered by scenarios
- Two parts, the **User Story** and its **Scenarios**
- Both use **templates** to ensure consistency

**User Story template:**

- As a **role**
- I want **feature**
- So that **benefit**

**Novel parts of BDD User Story are:**

- Template syntax
- Explicit statement of benefit (relieves scenario of job)

**Example User Story:**

As a user
I want to message a friend on the map
So that we can spontaneously meet up
4. **User tracking**: Map continues to track user’s changing location, with option to not track

- **As a user**

- **I want** the map to update with my changing location **unless** I’ve unclicked the track box

- **so that** I can continue to see friends who are closeby (for a spontaneous meet up) **or** who will be closeby soon

Note: took a few tries to come to agreement; customer and devs made different assumptions as to benefit of requirement, which really meant different requirements.
More important are BDD scenarios
- Expose/reduce ambiguity/assumptions
- Tilted toward use in testing
- Leaves out the rationale (U.S. benefit fills in here)

Scenario template:
**Given** context
**When** event
**Then** outcome

- Flexible
  - **When-Then** clauses repeated for longer progression
  - **And** is used for multiple **Given, When, or Then's**
BDD: Scenarios II

There are multiple scenarios per user story, driven by:
- Number of unique progressions into the feature
- Number of unique progressions out of the feature

Given I am logged in
And a friend is shown on the map
When I click the friend’s name
Then a text box pops up
When I click the cancel button
Then the text box disappears

Large number of progressions in/out
- Temptation is to enumerate cross product of valid in/out’s
- Use test classes concept to manage tradeoff
  - Need a case for messaging Alice and another for messaging Joe???
  - detail (more scenarios) vs. conciseness (fewer scenarios)
- Also consider if already covered by other User Stories
  - zooming out in the “show friends” User Story

Scenario is “covered” by (at least one) matching acceptance test
- E.g., need tests for sending message and canceling message
Practice: Write your own BDD Scenarios

- As a user

- I want the map to update with my changing location unless I’ve unclicked the track box

- so that I can continue to see friends who are close by (for a spontaneous meet up) or who will be close by soon

- Given that the map is updating When I click do-not-track Then stop updating the map

- Given that the map is updating When I move 100m to the left Then my icon on map uses 100m to the left

- Given the map is not updating When I move 100m to the left Then the map does not update

- Notes: left out dragging map for the “show buddies” U.S.; like the motion details here because it’s testable
Testing: two kinds required for project

- **BDD-style tests**, written in JUnit (or BDD’s JBehave)
  - Written in code; Tasks under a User Story
  - At least one for each BDD Scenario
  - Ideally written before actual code (but **not required**)
  - Run before Git push (e.g., like continuous integration)
  - Run frequently as regression tests (to see if broke anything)

- **System/acceptance tests**
  - Written in English
  - Combine a long progression of BDD User Story Scenarios
    - Finishing one scenario enables the “next” scenario
    - Centering you on map enables showing buddies around you
  - Still check if inputs produce expected outputs (**input/output pairs**)
  - Performed by user (like a demo) or a UI testing tool (Robotium)
  - Done at end of Iterations and Milestone (at least)
How many BBD JUnit Tests for “Msg”?

A. 2
B. 4
C. 8
D. 16
E. $2^8 + 2^6$

Given I am logged in
And a friend is shown on the map
When I click the friend’s name
Then a text box pops up
When I type a message
Then the send button is enabled
When I click send
Then the message is sent to the friend

Somewhere between 4 and 8:
• 1 each for the two scenarios (2)
• 1 each for their cross product (2)
  - no typing + send → no send?
  - typing + cancel → are you sure?
• Someone suggested as clicking the target (user name) as a case (2)
  - click target → get text box
  - miss target → don’t get box
Should we add matching scenarios for these? Clarify size of name target? Not clear (concise vs. detailed)
“Interaction” Design via Scenarios

- U.I. design is now called *Interaction Design*
  - Goal is to design a useful, simple, fun user experience

- BDD scenarios are great interaction design tool
  - One goal: minimize user clicks between steps
  - Draw screen for each major step (by hand is best)
  - Copy identical screens shown in other scenarios

- Can “Demo” to customer
- Generates important design debates
- *Should messaging be on separate tab?*
For Project

- Do Stories and Scenarios just like in article
  - I’ve simplified here due to space
  - Need the Name, for example

- You have to write (and run ;) two kinds of tests
  - Scenario-based system tests (planning phase)
  - BDD scenario tests in JUnit (code & delivery phase)

- Draw screens for interaction design by hand
  - Take pictures of them to put into digital form
  - But PowerPoint OK (stunts experimentation)
Take-Aways from Class Today

- We can’t test every input/output
- But can rigorously test most *classes* of input/output
  - Complex, and involves tradeoffs
- **BDD Scenarios** define likely, testable progressions
- BDD Scenarios & **User Story “benefit” clause** sharpen actual requirements
- BDD’s **templates** ensure compliance & consistency
- **Testing** is considered from the *earliest stages*
- **Interaction design** removes another layer of customer-developer **assumptions**
  - Aided by BDD Scenarios