Intro to BDD

Admin

- Ok – so
  - Quiz (exam in 2 weeks – not the whole class)
  - Presentations
  - Then lets get back into it.
  - Not going to get completely back on track till next week.

What is BDD

- BDD (Behavior Driven Development) began as a better TDD (Test Driven Development)
  - TDD had a tendency to focus on the small unit tests
  - Losing the forest for the trees.
  - BDD initially to fix that.

The old way of developing Software

BDD in a picture

BDD trying to avoid

- BDD Book:
  - Two primary reasons for software failure rates
    - “Not building the software right”
    - “not building the right software”
  - BDD an attempt to fix both of those.
  - Examples:
    - Healthcare.gov circa 2013
      - Postmortum not yet public
    - Queensland health department payroll software.
      - Delivered 2010 barely functional (required 2008)
      - Postmotem: $400+ million to make, $800+ million to fix

Example TDD tests

- Here is a proposed TDD test
  - “public class BankAccountTest {
  - @Test
  - public void testTransfer() {...}
  -
  - @Test
  - public void testDeposit() {...}
  - }”
  -

- Excerpt From: John Ferguson Smart. “BDD in Action: Behavior-Driven Development for the whole software lifecycle.” iBooks.

Example BDD test

- So what would a BDD equivalent look like?
  - “public class WhenTransferringInternationalFunds {
  - @Test
  - public void should_transfer_funds_to_a_local_account() {...}
  - @Test
  - public void should_transfer_funds_to_a_different_bank() {...}
  - ...
  - @Test
  - public void should_deduct_fees_as_a_separate_transaction() {...}
  - ...
  - }”
  - Excerpt From: John Ferguson Smart. “BDD in Action: Behavior-Driven Development for the whole software lifecycle.” iBooks.

BDD's Cousins

- Several methodologies developed similarly and at nearly the same time as BDD
  - Specification by Example
  - Acceptance-Test-Driven Development (ATDD)
  - Acceptance-Test-Driven Planning
  -
  -

BDD guiding principles

- Some rules of thumb
  - Focus on features customers need
    - Not one that might be neat soon.
  - Customers and devs work together to figure out which those features are
  - Embrace uncertainty
    - Waterfall is dead – you won't know everything before you start
  - Use real concrete examples to show features
  - “Don't write automated tests, write executable specifications”

Gherkin

- Gherkin:
  - a structured Natural Language format for specifying/testing a feature
  - Given..When..Then
  - Can use and/or to string together conditions
  - These structured text files are then read in by bdd frameworks
  - Available for nearly every language that matters
    - Java: jbehave
    - Python behave
    - Various others: cucumber

Gherkin example

Feature: Fight or flight (from python behave framework tutorial)

- In order to increase the ninja survival rate,
- As a ninja commander
- I want my ninjas to decide whether to take on an
- opponent based on their skill levels
-
- Scenario: Weaker opponent
- Given the ninja has a third level black-belt
- When attacked by a samurai
- Then the ninja should engage the opponent
-
- Scenario: Stronger opponent
- Given the ninja has a third level black-belt
- When attacked by Chuck Norris
- Then the ninja should run for his life

Living Documentation

- How is your software documented?

Living Documentation

- How is your software documented?
  - I know a fellow at IBM whose job it is to go work with their developers and write technical documentation
    - Existing projects
    - New projects?
  - What if there was a better way?

Living Documentation

- Living Documentation
  - Developed from automated acceptance criteria
  - Full circle, from code back to formalized natural language
  - Advantages?

Living Documentation

- Living Documentation
  - Developed from automated acceptance criteria
  - Full circle, from code back to formalized natural language
  - Advantages?
    - Docs are never out of date
    - Others?

Docs and Maintenance

- Software maintenance
  - 40-80% of software cost
  - Sometimes two groups of devs
    - Initial devs
    - Maintenance team
      - Has to learn the software from scratch.

BDD

- Advantages:
  - Reducing wasted effort/reduced cost
    - You don't spend lots of time building software no one wants
  - Easier changes
    - Change specs-> change tests
  - Faster releases
    - Easier changes lead to faster, safer releases

BDD

- Hurdles
  - The customer/business reps have to be involved all the time
  - Really need some sort of agile development process
  - No silo-ing the devs away
  - Tests need to be well written or it all falls apart