Storyboarding; Test-Driven Development

Dr. Kosta Damevski

CMSC 355 - Software Engineering: Spec and Design

Fall 2015

School of Engineering | Computer Science

Announcements

- Keep working on forming groups
    - When done, use the web form
- Most of what we are talking about now will be on the project
    - Remaining lectures on Android and VCS

Last time

- Requirements - the Agile way
  - Behavior-Driven Design (BDD)
  - User stories (in Connextra format)
- User stories map into several (testable!) scenarios
  - Can use as acceptance tests

Today (spoiler alert!)

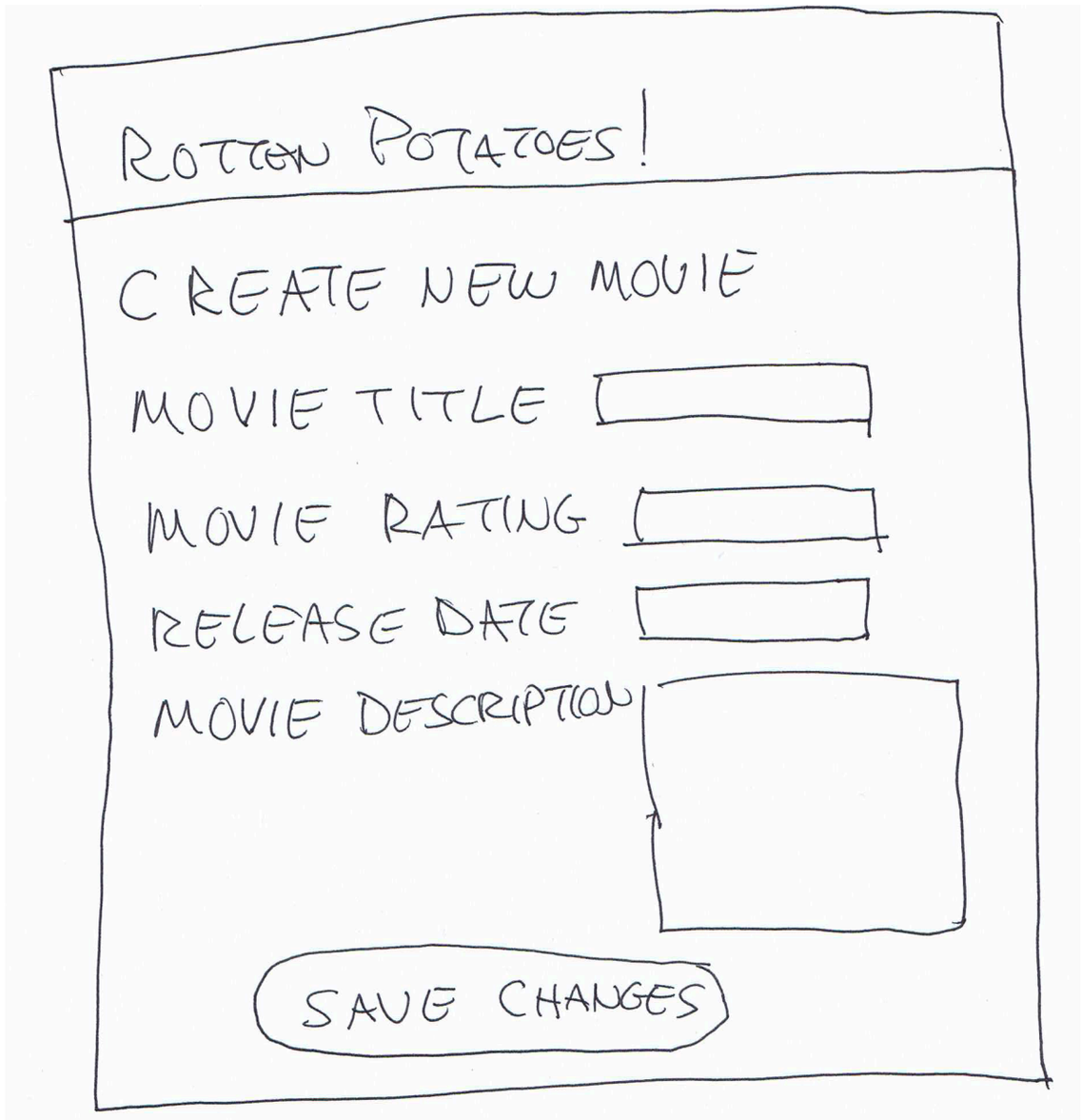
- UI part of the requirements
  - using storyboarding
- Techniques for agile project tracking
  - some of the features of the Pivotal Tracker tool
- Test-Driven Development (TDD)
  - write tests before implementing feature

Building a successful UI

- Android apps often face users
- User stories need User Interface (UI)
    - How to get customer to participate in UI design so is happy when complete?
    - Avoid WISBNWIW* UI?
- UI version of 3x5 cards?
- How can we show interactivity without building a prototype?

*What-I-Said-But-Not-What-I-Want

UI Sketches

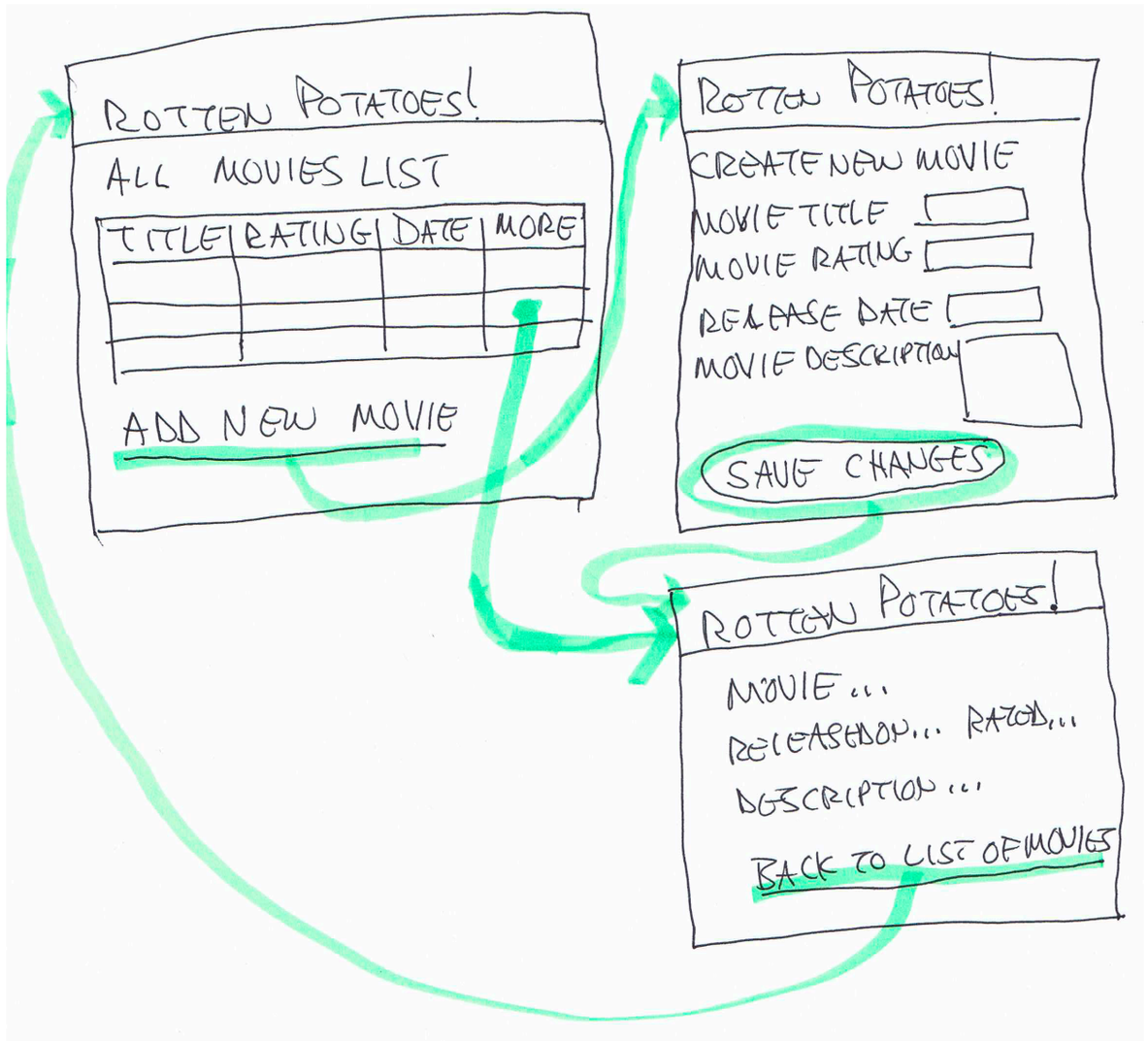


(From Engineering Software as a Service © by Armando Fox and David Patterson, used with permission)

Storyboards

- Need to show how UI changes based on user actions
    - HCI => “storyboards”
- Like scenes in a movie, but not linear

Example storyboard



(From Engineering Software as a Service © by Armando Fox and David Patterson, used with permission)

Ideas behind storyboarding

- Tedious to do sketches and storyboards,
  but easier than producing Android App UIs! and…
    - Less intimidating to nontechnical stakeholders
    - More likely to suggest changes to UI if not code behind it
    - More likely to focus on interaction rather than colors, fonts, …
- We can worry about making it look nice later, when putting together the actual app

Questions?

- on UI requirements/design and storyboarding

Productivity of agile teams

- Don’t we want to avoid major planning effort in Agile?
    - If so, how estimate time without a plan?
- Can User Stories be used to measure progress on project?
- What should a tool do to help measure progress for Agile?

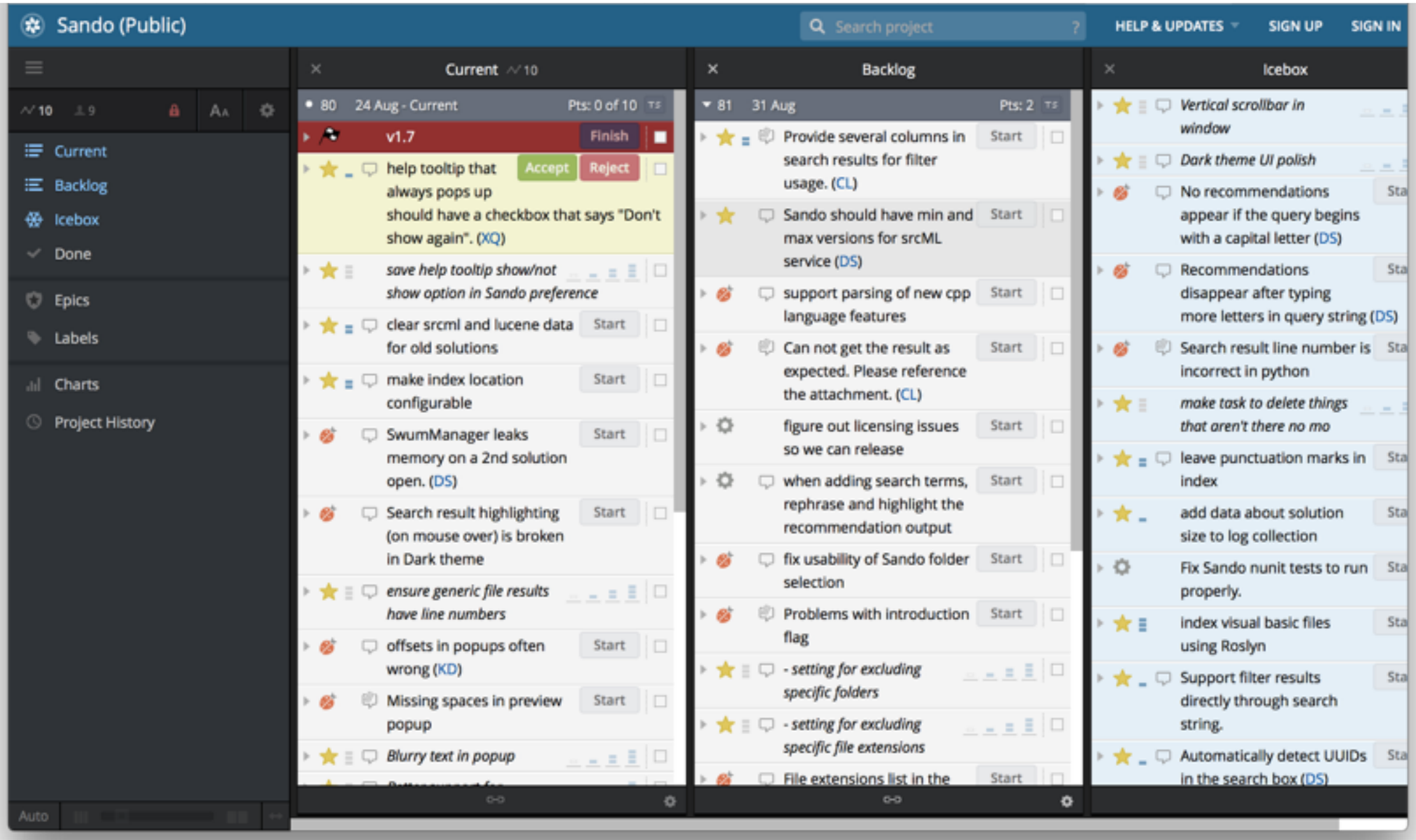
Measuring productivity

- A measure of team productivity: calculate avg no. stories / week?
    - But some stories much harder than others
        - Rate each user story in advance on a simple integer scale
- 1 for straightforward stories, 2 for medium stories, 3 for very complex stories
- **Velocity:** avg number of points / week

Velocity

- Can’t compare velocity across teams
- Doesn’t matter if velocity is 5 or 10 points per iteration
    - As long as team consistent
- Idea is to improve self-evaluation and suggest number of iterations for feature set

Pivotal Tracker



Features of Pivotal Tracker

- Prioritize user stories by where place them in Current, Backlog, Icebox panels
    - When completed, move to Done panel
- Developers push Finish, send to Product Owne
    - Owner tries story and then Accepts or Rejects
- Can add logical Release points, so can figure out when a Release will really happen
- Remaining points/Velocity

Pivotal Tracker: features vs. chores

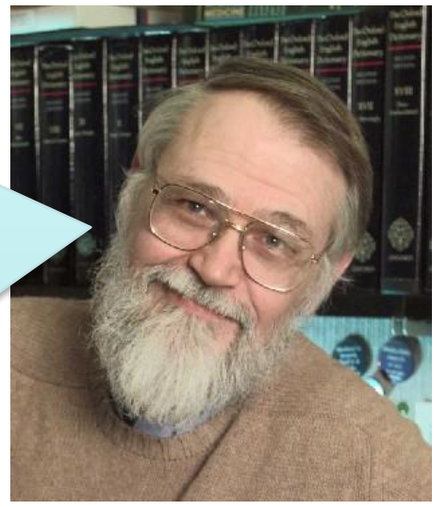
- Features
  - User stories that provide verifiable business value to customer
  - e.g. “Add agree box to checkout page”
    - Worth points & therefore must be estimated
- Chores & Bugs
  - User Stories that are necessary, but provide no direct, obvious value to customer
  - “The agree box should be grayed out when clicked”
  - “Clean up code (refactor) in the Payments subsystem”
  - No points

Questions?

- on measuring productivity in agile projects

Testing is important

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.



Testing can never demonstrate the absence of errors in software, only their presence

Survey Finds 58% of Software Bugs Result from Test Infrastructure and Process, Not Design Defects

Developers Prefer Taxes to Dealing with Software Testing

Sunnyvale, Calif. — June 2, 2010 Electric Cloud®, the leading provider of software production management (SPM) solutions, today released the results of a survey conducted in partnership with Osterman Research showing that the majority of software bugs are attributed to poor testing procedures or infrastructure limitations rather than design problems. Additionally, the software test process is generally considered an unpleasant process, with software development professionals rating the use of their companies' test systems more painful than preparing taxes.

Fifty-eight percent of respondents pointed to problems in the testing process or infrastructure as the cause of their last major bug found in delivered or deployed software, not design defects.

Specifically, the survey found:

- Completely automated software testing environments are still rare, with just 12 percent of software development organizations using fully automated test systems. Almost 10 percent reported that all testing was done manually.

Testing today

- Before
  - developers finish code, some ad-hoc testing
  - “toss over the wall to Quality Assurance [QA]”
    - QA staff manually poke at software
- Today/Agile
  - testing is part of every Agile iteration
  - developers test their own code
  - testing tools & processes highly automated
  - QA/testing group improves testability & tools

BDD+TDD: the big picture

- Behavior-driven design (BDD)
    - develop user stories (the features you wish you had) to describe how app will work
    - user stories become acceptance tests
- Test-driven development (TDD)
    - step definitions for new story,may require new code to be written
    - TDD says: write unit & functional tests for that code first, before the code itself
    - that is: write tests for the code you wish you had

Test-First development

- Think about one thing the code should do
  - Capture that thought in a test, which fails
  - Write the simplest possible code that lets the test pass
    - Aim for “always have working code”
- Refactor
  - The code, the test, related tests, etc.

Typical TDD workflow

- Write a test(s)
    - e.g. testing that clicking the “Cancel” button returns to the previous activity
- Run the test
    - skip this step, if you want to
    - the test should fail = “red” light
- Write the code
    - implementing the “Cancel” button functionality
- Run the test, again
    - It should pass

Questions

- on TDD?

Summary

- A few more agile principles, for:
    - Test-Driven Development
    - Productivity measurement
    - UI requirements engineering
- TDD is one of the important parts of Extreme Programming (XP)
    - XP is flavor of Agile we are using this semester

References

- *“Engineering Software as a Service” by Armando Fox and David Patterson (2nd Edition)*
- *“Software Engineering” by Ian Sommerville (10th Edition)*

Questions