Why do SW projects fail?

Failing projects:

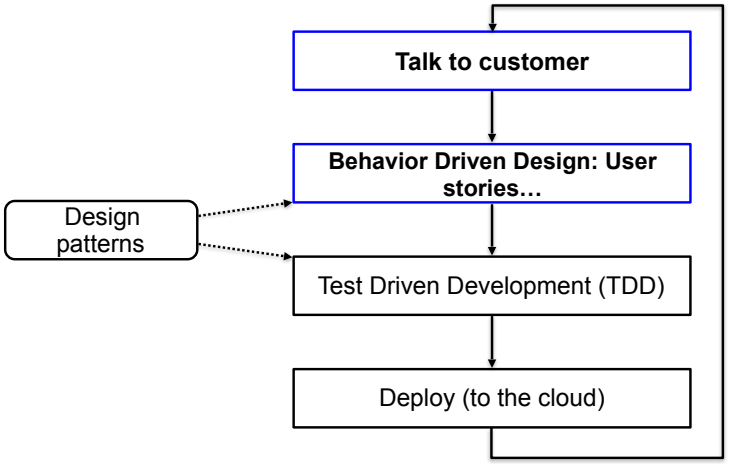
- Don't do what customers want
- Are late
- Over budget
- Hard to maintain and evolve
- All of the above

How does Agile try to avoid failure?

Recall: Agile Lifecycle

- Work closely and continuously with stakeholders to develop requirements, tests
  - Users, customers, developers, maintenance programmers, operators, project managers, …
- Maintain a working prototype while deploying new features every *1-2 week iteration*
- Check in with stakeholders on what's next, to validate building right thing (vs. verify)

*DD in our Agile iterations

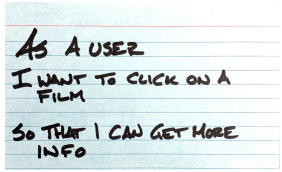


Behavior-Driven Design (BDD)

- BDD is a conversation about app behavior *before and during development* to reduce miscommunication
  - Recall "Individuals and interactions over processes and tools" in Agile manifesto
- Requirements written down as *user stories*
  - Lightweight descriptions of how application is used
- BDD concentrates on *behavior* vs. *implementation* of application
  - Test Driven Development (TDD) focuses on implementation

User Stories

- 1-3 sentences in everyday language
  Fits on an index card
  Written by or with the customer
- Often in "Connextra" format:
  Feature name
  As a [kind of stakeholder],
  I want to [some task],
  So that [some result or benefit].
  (all 3 phrases are needed, but can be in any order)



User stories will ultimately become work items in our product backlog (our team's prioritized "to-do list")

S.M.A.R.T. user stories

- **S**pecific
- **M**easurable (with specific, implies testable)
- **A**chievable (ideally implement in 1 iteration)
- **R**elevant (discover "business" value or kill)
- **T**ime-boxed (know when to split/stop)

As a user,
I want to click on a film,
so that I get more information.

As a user,
I want to click on a film to get plot details,
so that I can see if I will like the film

Student advice: Stories vs. Layers

- "Dividing work by stories helps all team members understand app & be more confident when changing it"
- "Tracker helped us prioritize features and estimate difficulty"
- "We divided by layers [front-end vs. back-end vs. JavaScript, etc.] and it was hard to coordinate getting features to work"
- "It was hard to estimate if work was divided fairly…not sure if our ability to estimate difficulty improved over time or not"

Adapted from Berkeley CS169

The customer wants "login with Facebook" integrated into their site. Nobody on your team is familiar with how to do this. You should:

A. Break up the story into very small user stories, to be on the safe side about how long each chunk takes.
B. Do a spike on Facebook integration, then propose one or more stories to implement.
C. Apologize to the customer that they can't have this functionality

Epic > User Stories > Scenarios

User Stories are expanded into scenarios

Scenarios are formal but not code.

- Creates a “meeting point” between developers and customers.

With Gherkin syntax, we turn scenarios into automated acceptance tests:

- *Given [a context],*
- *When [an event happens],*
- *Then [an outcome should occur]*

cucumber

Testing scenario example

```
Given I open the url 'http://the/test/url'
When I click on the element 'Jurassic World'
Then I expect that the element 'img[src="http://the/
poster"]' is visible
```

BDD is all about conversation

“Having conversations is more important than capturing conversations is more important than automating conversations”

Liz Keough

Given what you have learned about BDD, which of the following is the most accurate?

A. BDD is designed to support validation (build the right thing) and verification (build it right)
B. The best user stories include information about implementation choices
C. User stories have no counterpart in plan-and-document processes
D. Functionality should only be featured in a single user story for a single stakeholder

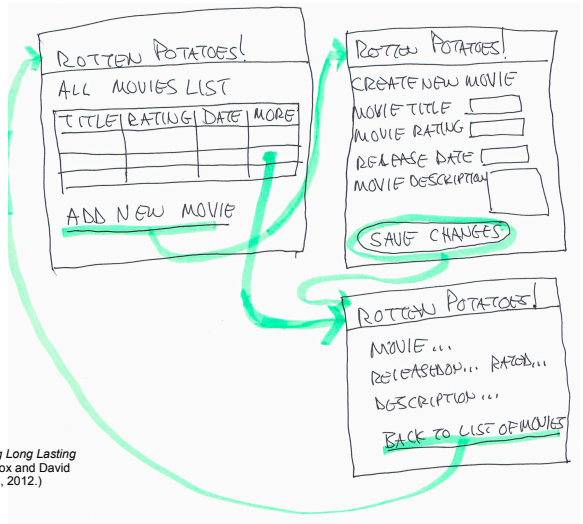
Building Successful UI

Our apps often faces users, thus needs UI

- How to get customer to participate in the UI design so they are happy with results?
  - Avoid WISBNWIW* UI
  - UI version of User Story index cards?
- How to get feedback cheaply?

* What-I-Said-But-Not-What-I-Wanted

Lo-fi Storyboards



(Figure 4.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

Lo-Fi to React, HTML and CSS

Sketches and storyboards are tedious, *but easier than code!* And…

- Less intimidating to non-technical stakeholders
- More likely to suggest changes to UI if not code behind it
- More likely to focus on *interaction* rather than colors, fonts, …

What you think is cool may not be what your users (customers) think is valuable.

Student Advice: BDD & Lo-Fi Prototyping

- "Lo-fi and storyboards really helpful in working with customer"
- "Frequent customer feedback is essential"
- "What we thought would be cool is not what customer cared about"
- "We did hi-fi protoypes, and invested a lot of time only to realize customer didn't like it"
- "Never realized how challenging to get from customer description to technical plan"

Adapted from Berkeley CS169