Introduction to Behavior-Driven Design and User Stories

(Engineering Software as a Service § 7.1)

David Patterson

© 2013 David Patterson & David Patterson
Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License

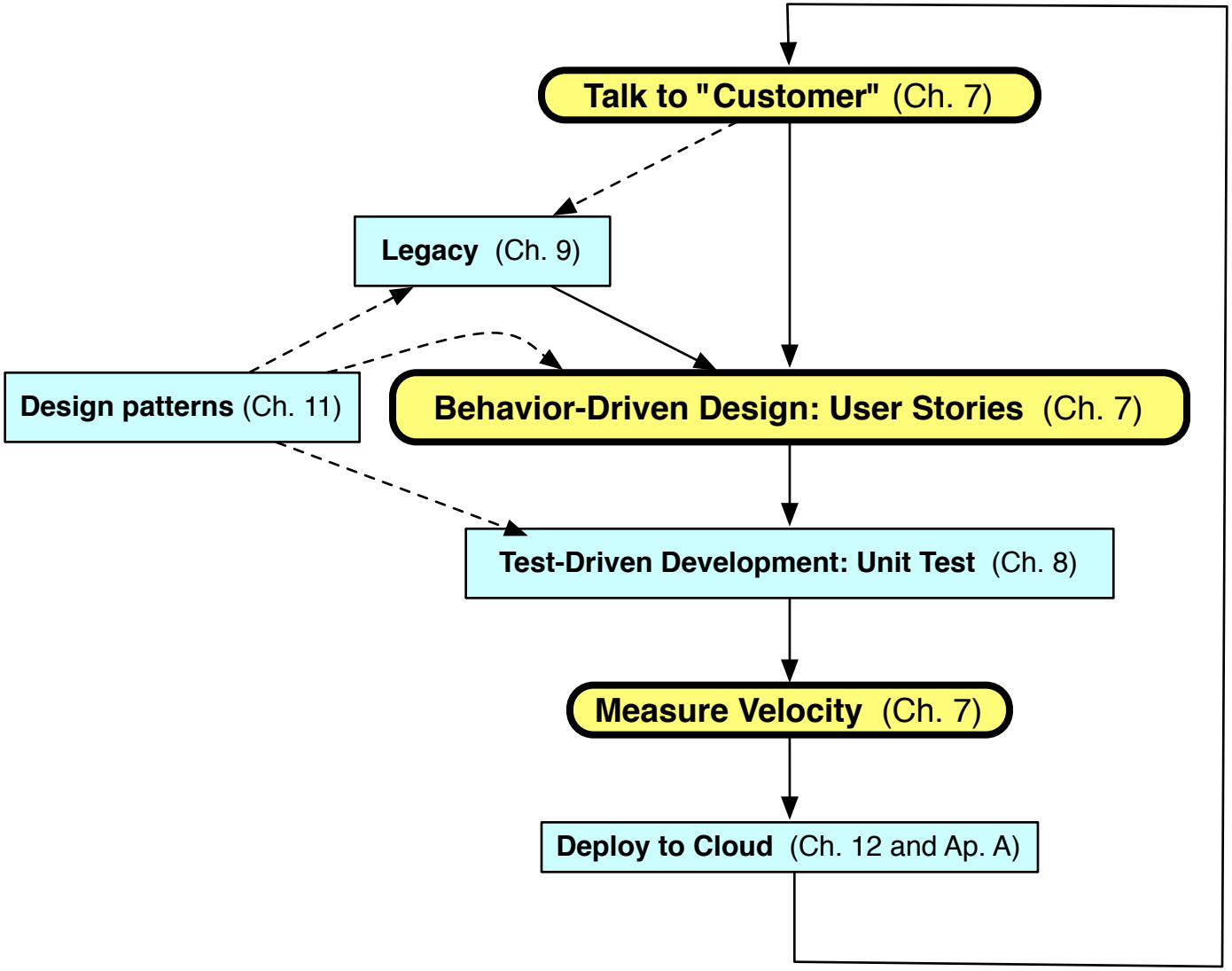
Why do SW Projects Fail?

- Don't do what customers want
- Or projects are late
- Or over budget
- Or hard to maintain and evolve
- Or all of the above
- How does Agile try to avoid failure?

Agile Lifecycle Review

- Work closely, continuously with stakeholders to develop requirements, tests
  - Users, customers, developers, maintenance programmers, operators, project managers, …
- Maintain working prototype while deploying new features every **iteration**
  - Typically every 1 or 2 weeks
  - Instead of 5 major phases, each months long
- Check with stakeholders on what's next, to validate building right thing (vs. verify)

Agile Iteration

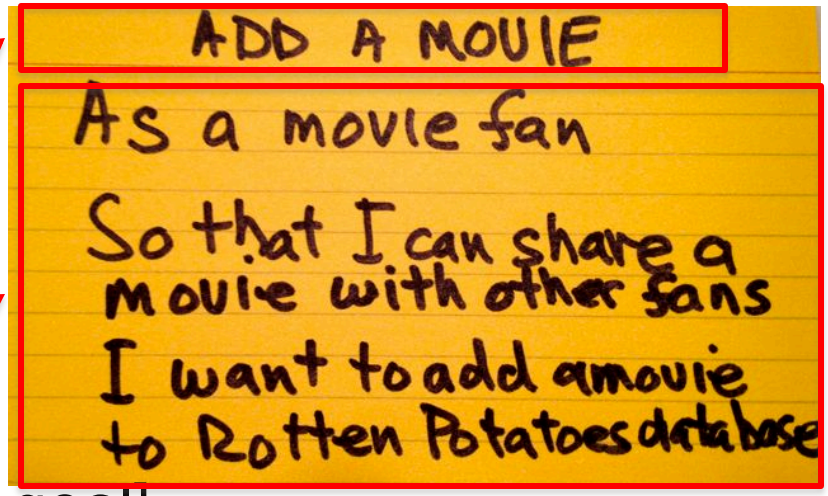


Behavior-Driven Design (BDD)

- BDD asks questions about behavior of app *before and during development* to reduce miscommunication
  - Validation vs. Verification
- Requirements written down as *user stories*
  - Lightweight descriptions of how app used
- BDD concentrates on *behavior* of app vs. *implementation* of app
  - Test Driven Development or TDD (future segments) tests implementation

User Stories

- 1-3 sentences in everyday language
  - Fits on 3" x 5" index card
  - Written by/with customer
- "Connextra" format:
  - Feature name
  - As a [kind of stakeholder],
    So that [I can achieve some goal],
    I want to [do some task]
  - 3 phrases must be there, can be in any order
- Idea: user story can be formulated as *acceptance test before* code is written



Why 3x5 Cards?

- (from User Interface community)
- Nonthreatening => all stakeholders participate in brainstorming
- Easy to rearrange => all stakeholders participate in prioritization
- Since stories must be short, easy to change during development
  - As often get new insights during development

Different stakeholders may describe behavior differently

- *See which of my friends are going to a show*
  - As a theatergoer
  - So that I can enjoy the show with my friends
  - I want to see which of my Facebook friends are attending a given show

- *Show patron's Facebook friends*
  - As a box office manager
  - So that I can induce a patron to buy a ticket
  - I want to show her which of her Facebook friends are going to a given show

Product Backlog

- Real systems have 100s of user stories
- *Backlog*: User Stories not yet completed
- Prioritize so most valuable items highest
- Organize so they match SW releases over time

Related: Spike

- Short investigation into technique or problem
  - E.g. spike on recommendation algorithms
  - Experiment, hack, do whatever works
- Bound the time allotted
- When done, *throw code away*
  - Now that know approach you want, write it right!

Points, Velocity, and Pivotal Tracker

(Engineering Software as a Service § 7.2)

David Patterson

© 2013 David Patterson & David Patterson
Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License

Productivity and Tools

- 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 measured progress for Agile?

Measuring Productivity

- A measure of team productivity: calculate avg no. stories / week?

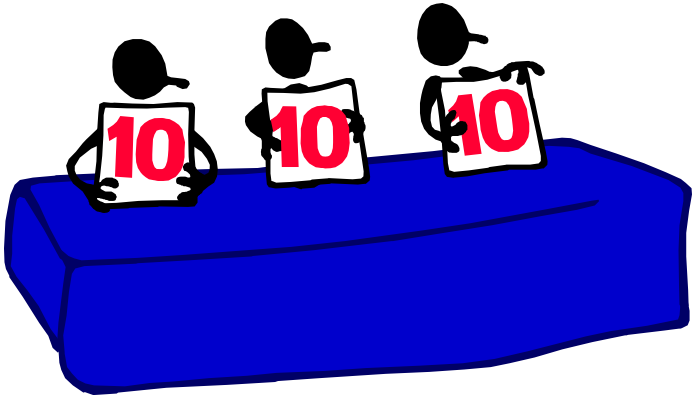
  – But some stories much harder than others
- Rate each user story in advance on a simple integer scale
  – 1 for straightforward stories, 2 for medium stories, 3 for very complex stories
- Velocity: avg number of *points* / week

More on Points

- Once get experience, Fibonnaci scale is commonly used: 1, 2, 3, 5, 8
  - (Each new number is sum of previous 2)
  - At Pivotal Labs, 8 is extremely rare
  - Advice: when starting, if ≥5 then **split up story!**
- Teams vote: hold up fingers, take average
  - If a big disagreement (2 and 5), discuss more

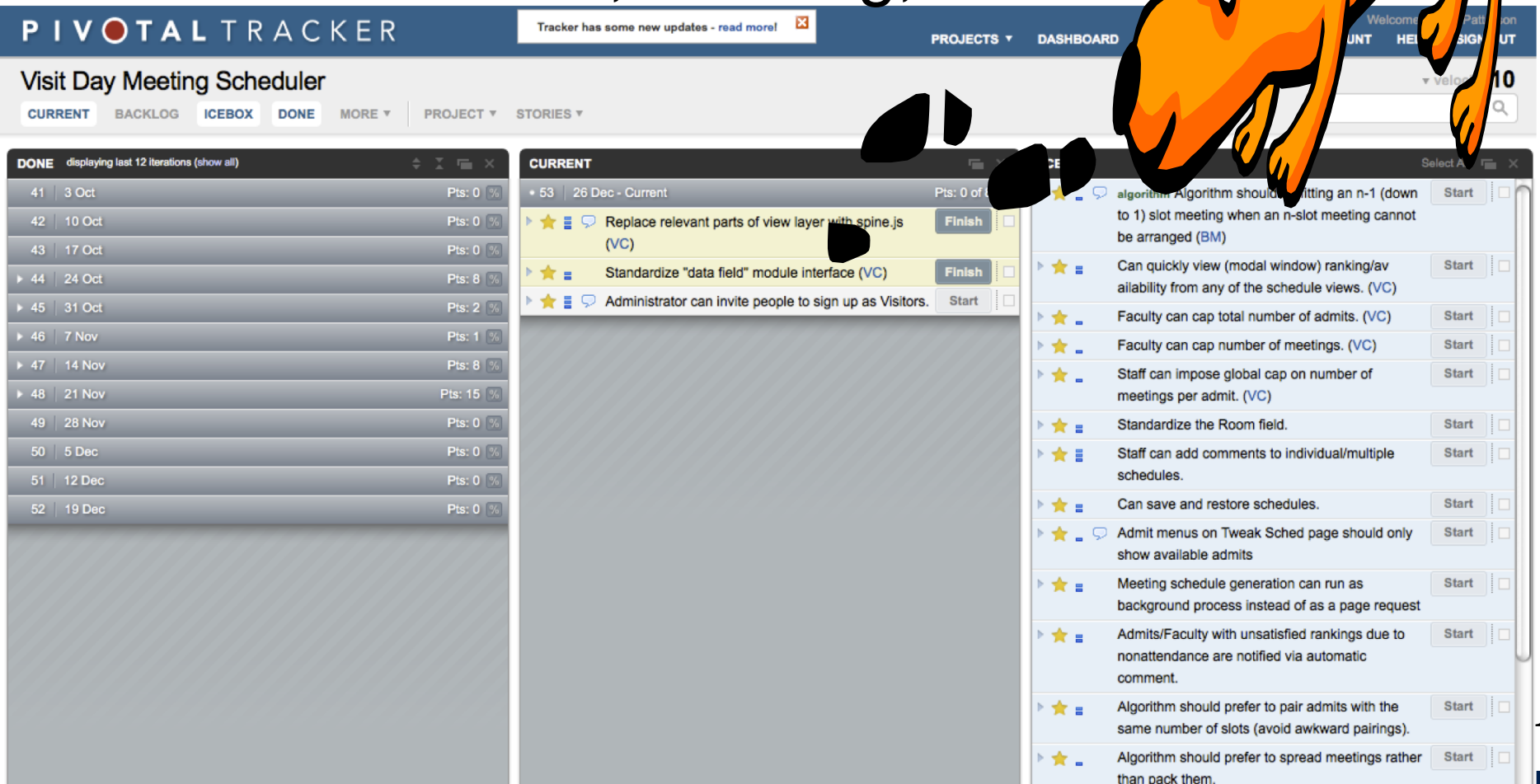
More on Points

- ≥5 => divide user story into simpler stories, group into *epics*
  - backlog not too demanding
- 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

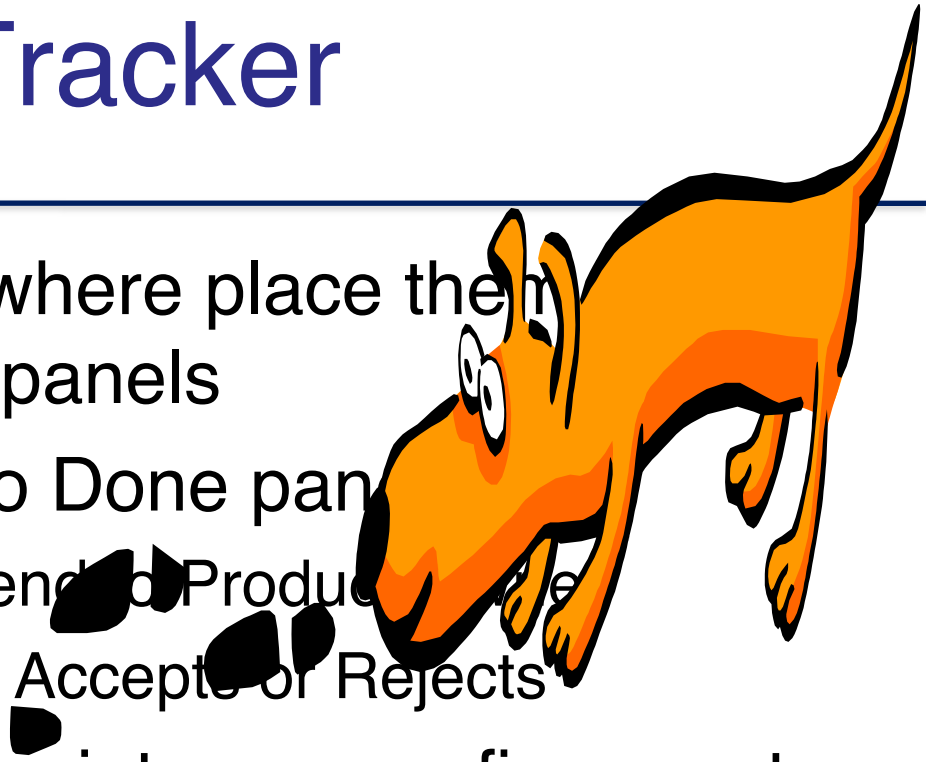


Demo: Pivotal Tracker

- Calculates velocity for team, manages stories: Current, Backlog, Icebox



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
  - 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
- *Epic* (with own panel)
  - Combine related user stories together
  - Ordered independent of user story in Backlog

Pivotal Tracker: Features vs. Chores

- Features
  - User stories that provide verifiable business value to customer
    - "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
    - "Find out why test suite is so slow"
    - "Refactor the Payments subsystem"
  - No points

Team Cyberspace Whiteboard

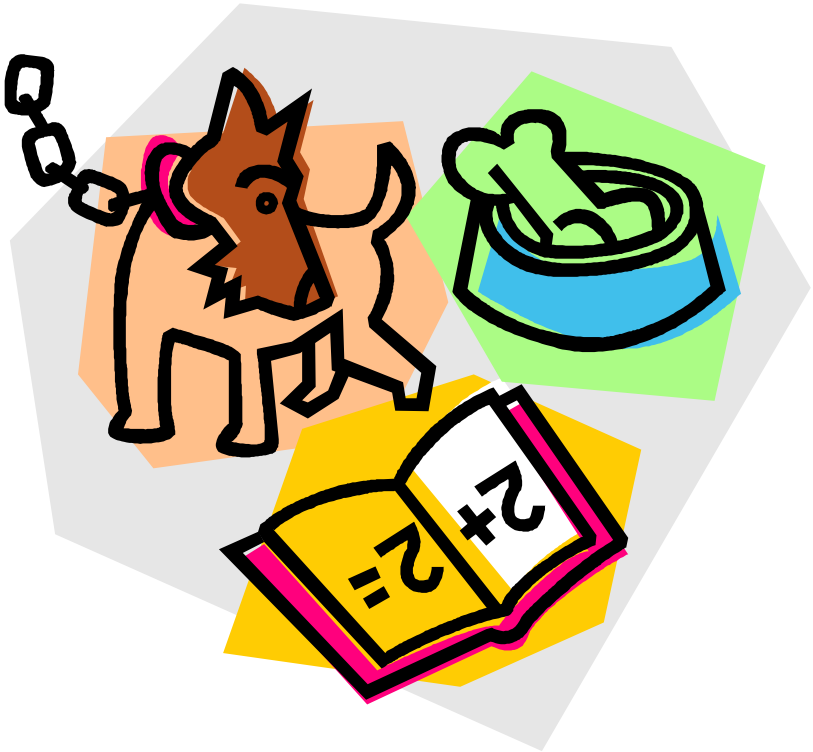
- Tracker allows attaching documents to User stories (e.g., LoFi UI)
- Wiki with Github repository
- Google Documents: joint creation and viewing of drawings, presentations, spreadsheets, and text documents
- Campfire: web-based service for password-protected online chat rooms

7. 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"

SMART User Stories

(Engineering Software as a Service § 7.3)



David Patterson

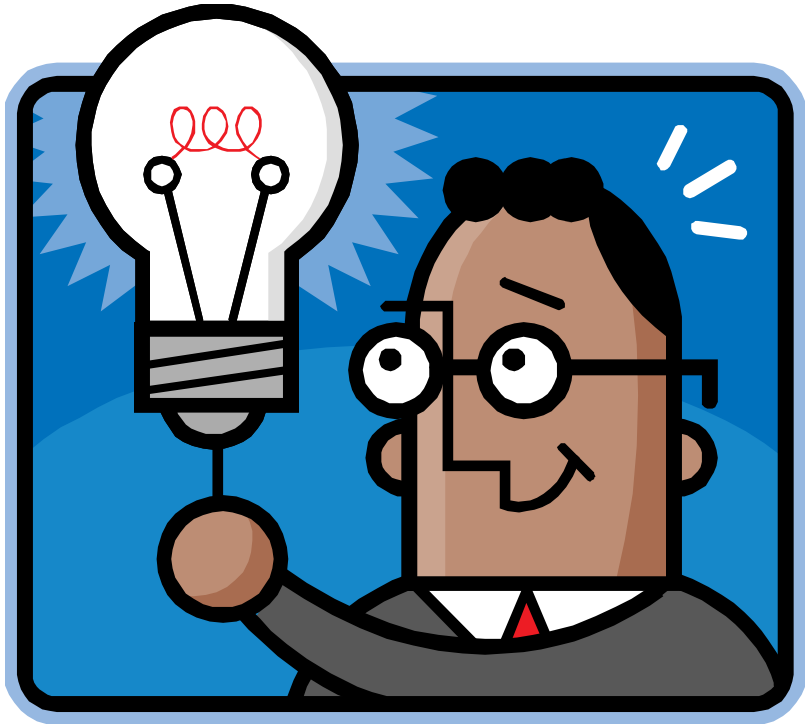
© 2013 David Patterson & David Patterson
Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License

Creating User Stories

- How do you know if you have a good user story vs. bad user story?
  - Right size?
  - Not too hard?
  - Is worthwhile?

SMART stories

- **S**pecific
- **M**easurable
- **A**chievable
  (ideally, implement in 1 iteration)
- **R**elevant
  ("the 5 why's")
- **T**imeboxed
  (know when to give up)



Specific & Measurable

- Each scenario testable
  - Implies known good input and expected results exist
- Anti-example: "UI should be user-friendly"
- Example: Given/When/Then.

1. *Given* some specific starting condition(s),
2. *When* I take specific action X,
3. *Then* one or more specific thing(s) should happen

Achievable

- Complete in 1 iteration
- If can't deliver feature in 1 iteration, deliver subset of stories
  - Always aim for working code @ end of iteration
- If <1 story per iteration, need to improve point estimation per story

Relevant: “business value”

- Discover business value, or kill the story:
  - Protect revenue
  - Increase revenue
  - Manage cost
  - Increase brand value
  - Making the product remarkable
- Can you include stories that don’t have obvious business value?

5 Whys to Find Relevance

- *Show patron's Facebook friends*

  As a box office manager

  So that I can induce a patron to buy a ticket

  I want to show her which Facebook friends are going to a given show

  1. Why?

  2. Why?

  3. Why?

  4. Why?

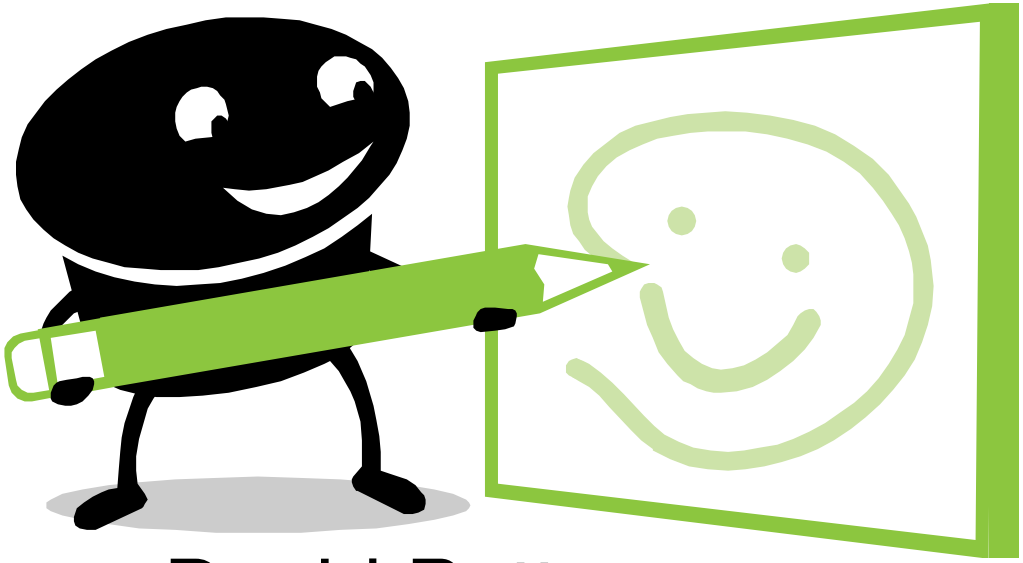
  5. Why?

Timeboxed

- Stop story when exceed time budget
  - Give up or divide into smaller stories or reschedule what is left undone
- To avoid underestimating length of project
- Pivotal Tracker tracks velocity, which helps avoid underestimate

Lo-Fi UI Sketches and Storyboards

(Engineering Software as a Service § 7.4)



David Patterson

© 2012 David Patterson & Armando Fox
Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License

Building Successful UI

- SaaS apps often faces users

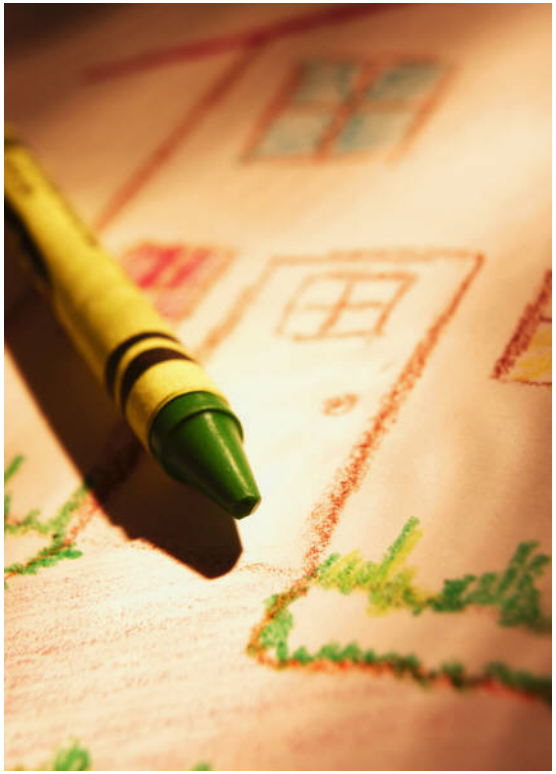
⇒User stories need User Interface (UI)

- How get customer to participate in UI design so is happy when complete?
  - Avoid WISBNWIW* UI?
  - UI version of 3x5 cards?
- How show interactivity without building prototype?

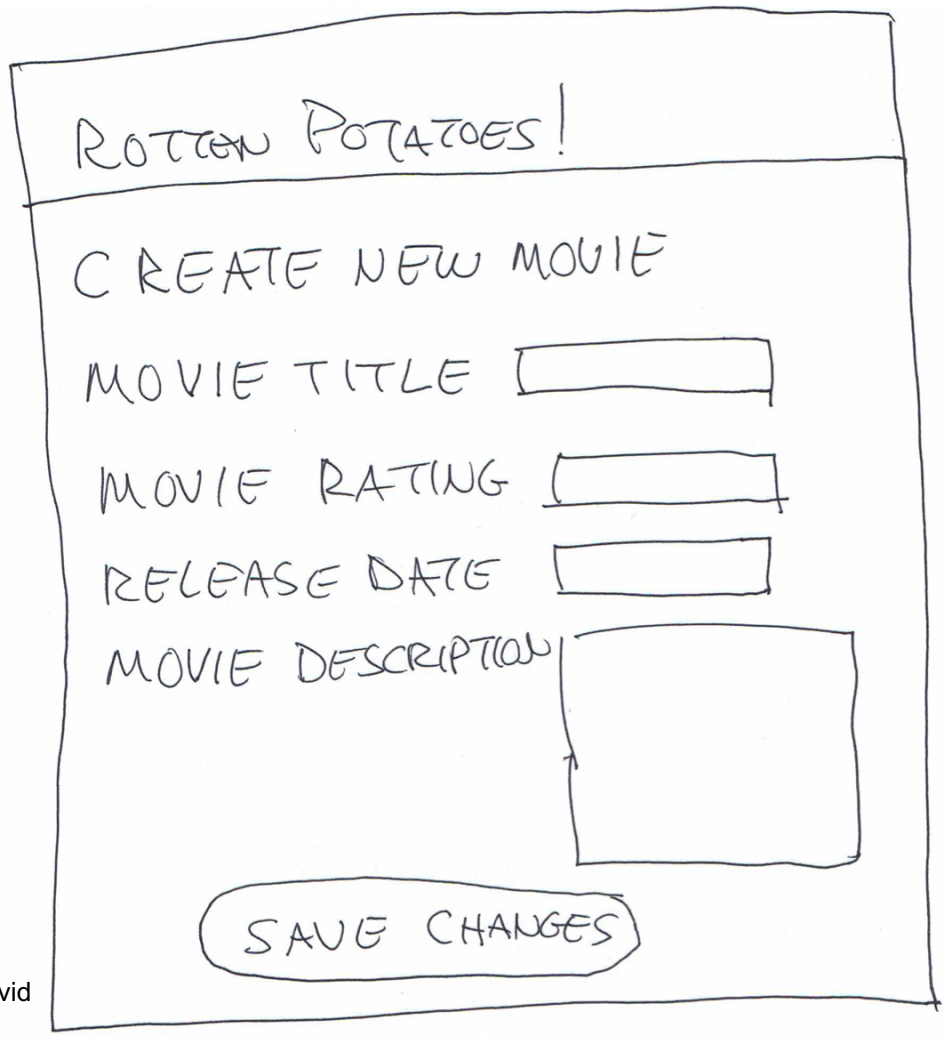
* What-I-Said-But-Not-What-I-Want

SaaS User Interface Design

- UI Sketches: pen and paper drawings or "Lo-Fi UI"



Lo-Fi UI Example

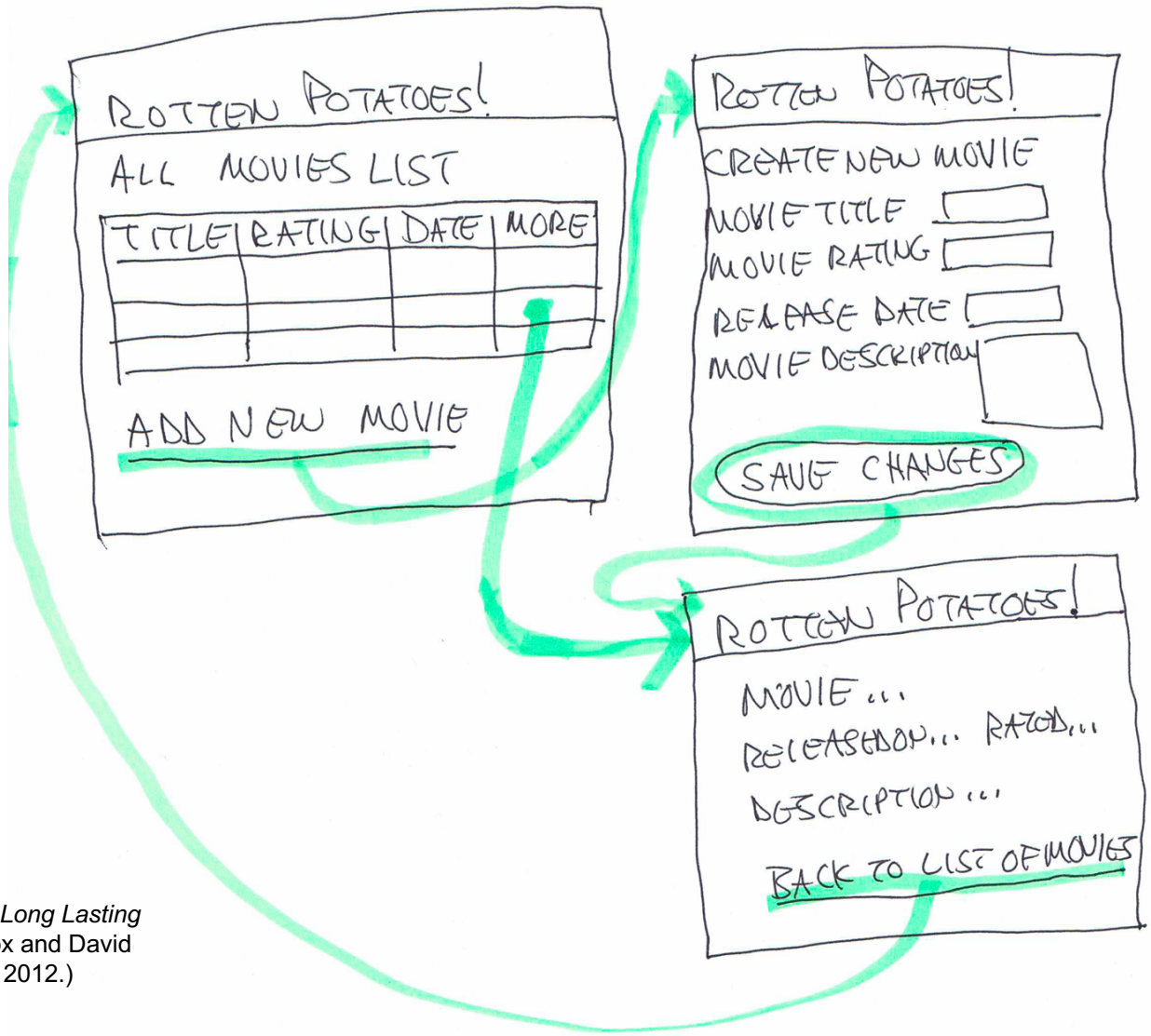


(From *Engineering Software as a Service* © by Armando Fox and David Patterson, used with permission)

Storyboards

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

Example Storyboard



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

Lo-Fi to HTML

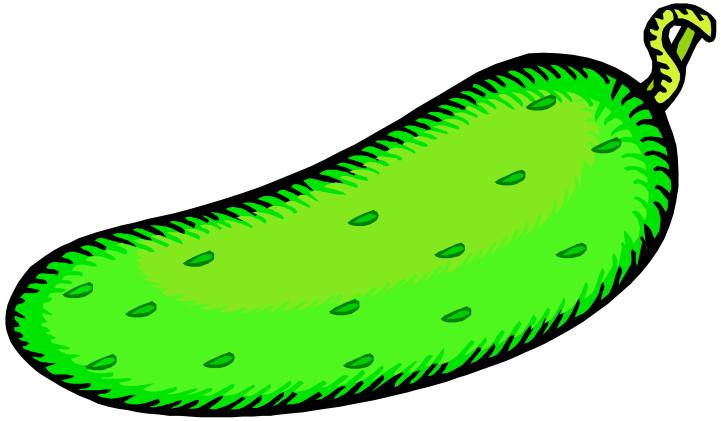
- Tedious to do sketches and storyboards, but easier than producing HTML! And…
  - Less intimidating to nontechnical stakeholders
  - More likely to suggest changes to UI if not code behind it
  - More likely to focus on *interaction* rather than colors, fonts, …
- CSS (Cascading Style Sheets) can make it look nice later

#8. Working with the customer: 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"

Introducing Cucumber & Capybara

(Engineering Software as a Service § 7.5)



David Patterson

© 2013 Armando Fox & David Patterson, all rights reserved

User stories => Acceptance Tests?

- Wouldn't it be great to automatically map 3x5 card user stories into tests for user to decide if accept the app?
- How would you match the English text to test code?
- How could you run the tests without a human in the loop to perform the actions?

Cucumber: Big Idea

- Tests from customer-friendly user stories
  - Acceptance: ensure satisfied customer
  - Integration: ensure interfaces between modules consistent assumptions, communicate correctly.
- Cucumber meets halfway between customer and developer
  - User stories not code, so clear to customer and can be used to reach agreement
  - Also not completely freeform, so can connect to real tests

Example User Story

```
Feature: User can manually add movie
```

1 Feature

```
Scenario: Add a movie
```

≥1 Scenarios / Feature

```
  Given I am on the RottenPotatoes home page
  When I follow "Add new movie"
  Then I should be on the Create New Movie page
  When I fill in "Title" with "Men In Black"
  And I select "PG-13" from "Rating"
  And I press "Save Changes"
  Then I should be on the RottenPotatoes home page
  And I should see "Men In Black"
```

3 to 8 Steps / Scenario

Cucumber User Story, Feature, and Steps

- User story: typically maps to one feature
- Feature: ≥1 scenarios that show different ways a feature is used
  - Keywords `Feature` and `Scenario` identify respective components
  - both *happy path* & *sad path* scenarios

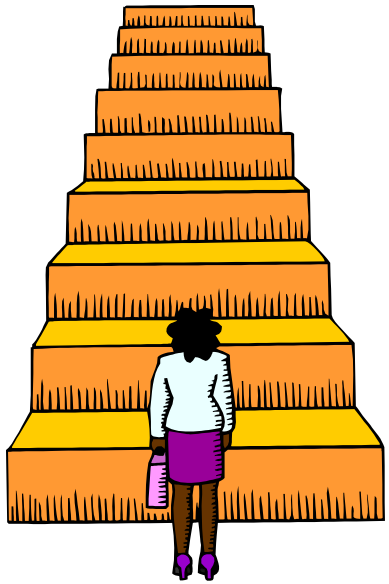
`features/*.feature`

- Scenario: typically 3 - 8 steps
- Step definitions: Ruby code to test steps

`features/step_definitions/*_steps.rb`

5 Step Keywords

1. `Given` steps represent state of world before event: preconditions
2. `When` steps represent event
   - e.g., simulate user pushing a button
3. `Then` steps represent expected postconditions; check if true
4. / 5. `And` & `But` extend previous step



These are all aliases for same method

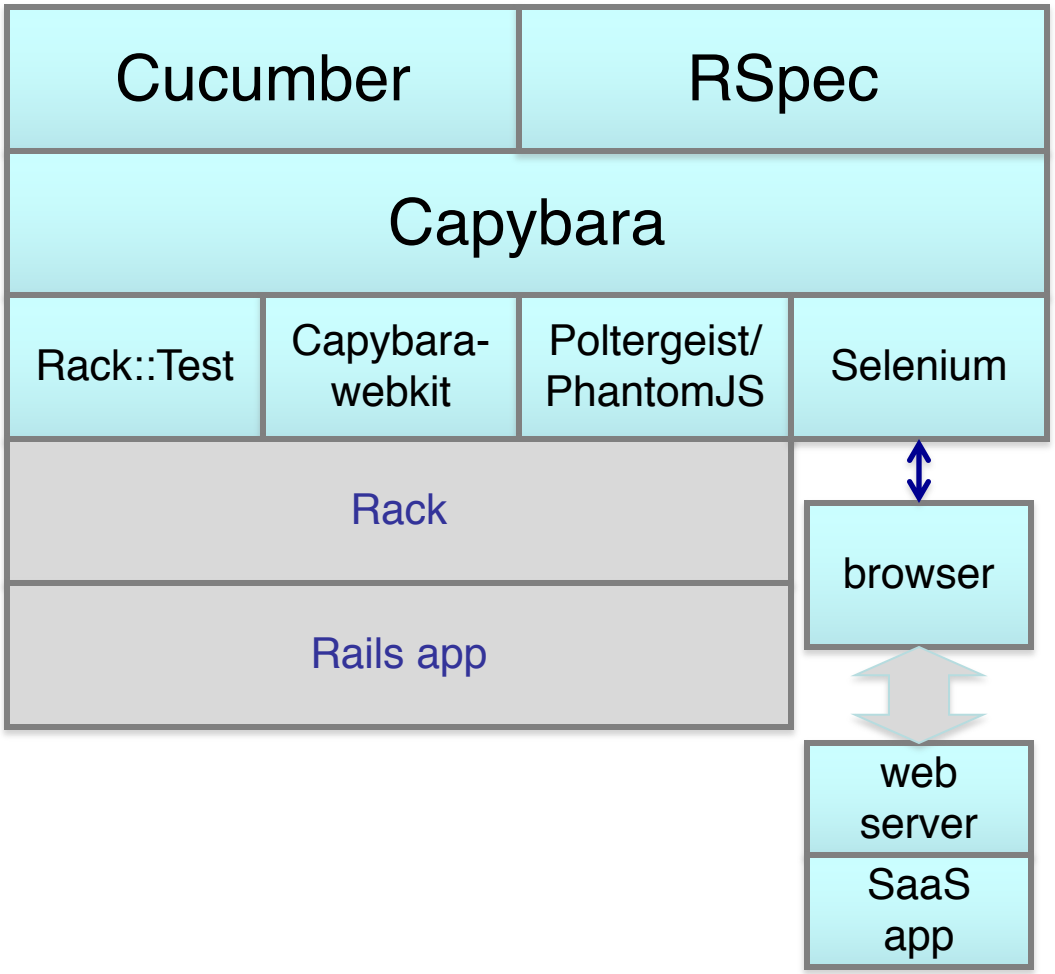
Steps => Step Definitions via Regular Expressions

- ***Regexes match English phrases in steps of scenarios to step definitions!***
- `Given /^(?:|I )am on (.+)$/`
- `"I am on the Rotten Potatoes home page"`
- Step definitions (Ruby code) likely use captured string
  - `"the Rotten Potatoes home page"`

Fake User to try Scenarios?

- Tool that pretends to be user to follow scenarios of user story
- Capybara simulates browser
  - Can interact with app to receive pages
  - Parse the HTML
  - Submit forms as a user would

Cucumber testing stacks

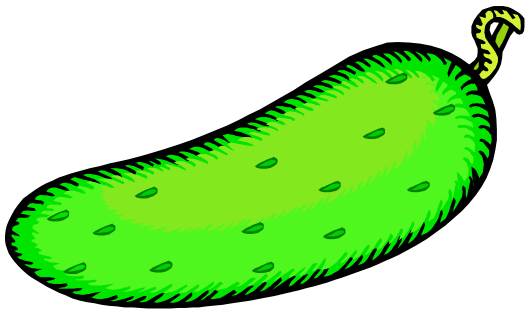


- With Selenium, can script completely external interactions
- SauceLabs.com will run your Selenium tests and send you videos of results

From Red to Green

- `cucumber` *filename* to run one feature, `rake cucumber` runs all
- Green for passing steps
- Yellow for not yet implemented
- Red for failing
  (then following steps are Blue)
- Goal: Make all steps green for pass

  (Hence green vegetable for name of tool)



Cucumber Summary

- New feature => UI for feature, write new step definitions, even write new methods before Cucumber can color steps green
- Usually do happy paths first
- Background lets us DRY out scenarios of same feature
- BDD/Cucumber test behavior; TDD/RSpec in folllowing chapter is how write methods to make all scenarios pass

Agile Cost Estimation

(Engineering Software as a Service § 7.4)

David Patterson

© 2013 David Patterson & David Patterson
Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License

Agile Cost Estimation

- Real world needs to estimate cost before customer can agree to project
- If no careful planning and schedule in Agile, how can you estimate the cost of a project?

Pivotal Labs Model

- Pivotal Labs teaches customer Agile
- Using Agile, Pivotal never commits to delivering features X, Y, and Z by date D
- Instead, commits resources to work in the most efficient way possible up to date D
  - Customer works with team to define priorities continuously up to date D
- Still need estimate for project

Pivotal Labs Agile Estimate

1. 1 hour phone call to explain method
   - Joint effort, customer time commitment, …
2. Customer visits for 1.5 hour "scoping"
   - Customer brings designer, developer, designs
     - Anything to clarify what want done
   - Pivotal brings 2 engineers who ask questions
     - Trying to identify what adds uncertainty to estimate
3. Engineers take ½ hour to estimate weeks
   - Little vs. large uncertainty: 20-22 vs. 18-26 wks
4. Bid cost as time and materials to customer

Plan-And-Document Perspective

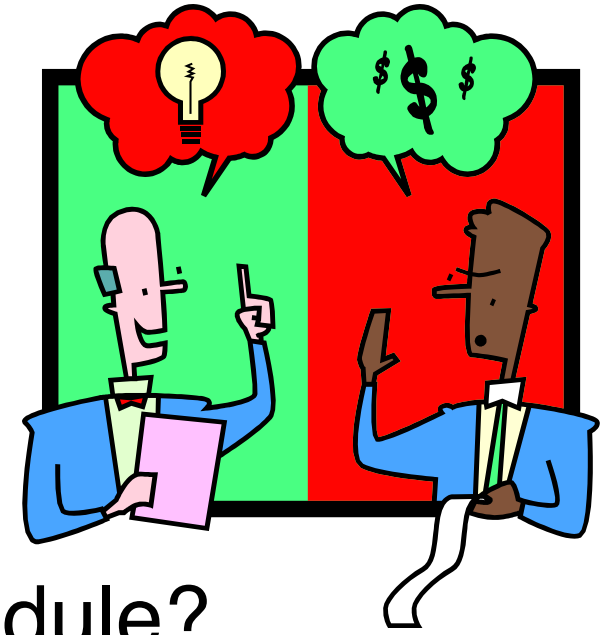
(Engineering Software as a Service § 7.10)

David Patterson

© 2013 David Patterson & David Patterson
Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License

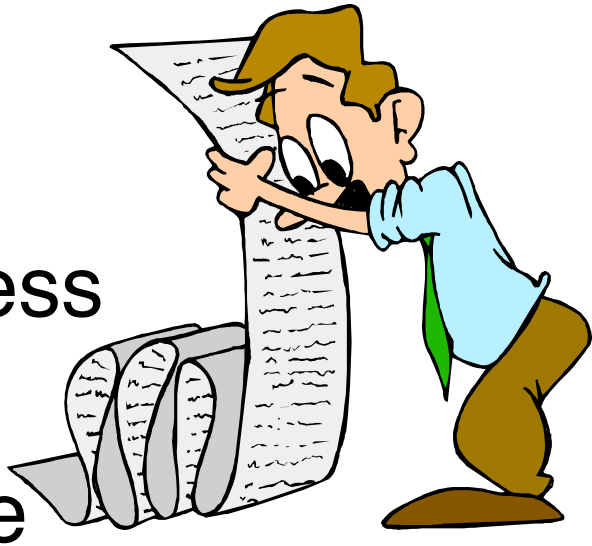
Introduction

- What does Plan-and-Document do instead of
  - User stories?
  - Points?
  - Velocity?
- How does a project manager estimate costs? Make a Schedule?



P&D Equivalents+

1. Requirements Elicitation
2. Requirements Documentation
3. Cost Estimation
4. Scheduling & Monitoring Progress
5. Change Management for Requirements, Cost, &Schedule
6. Ensuring Implementation Matches Requirement Features
7. Risk Analysis & Management



P&D Requirements Elicitation

- Elicit *functional*&*non-functional* requirements:

1. *Interviewing* - see how currently *really* done
   - Stakeholders answer predefined questions
   - Or just have informal discussions
2. Cooperatively create *Scenarios*
   - Initial state, show flow for happy & sad paths, what is concurrent, final state
3. Create *Use Cases*
   - List of steps to achieve goal between user and system; has language (UML) to describe

P&D Requirements Documentation

- Document requirements via *Software Requirements Specification* (*SRS*)
  - 100s of pages; IEEE standard for SRS!
- Have stakeholders read SRS, or build basic prototype, or generate test cases to check:
  - *Validity*: are all requirements necessary?
  - *Consistency*: do requirements conflict?
  - *Completeness*: are all requirements and constraints included?
  - *Feasibility*: can requirements be implemented?

P&D Cost Estimation

- Manager decomposes SRS into tasks
- Estimates weeks per tasks
  - 1 week ≤ Tasks ≤ 8 weeks
- Covert person-weeks into $ via salaries and overhead
- Estimate *before* & *after* contract
  - Add safety margin: 1.3 to 1.5X
  - Make 3 estimates (best case, expected, worst) then make best guess

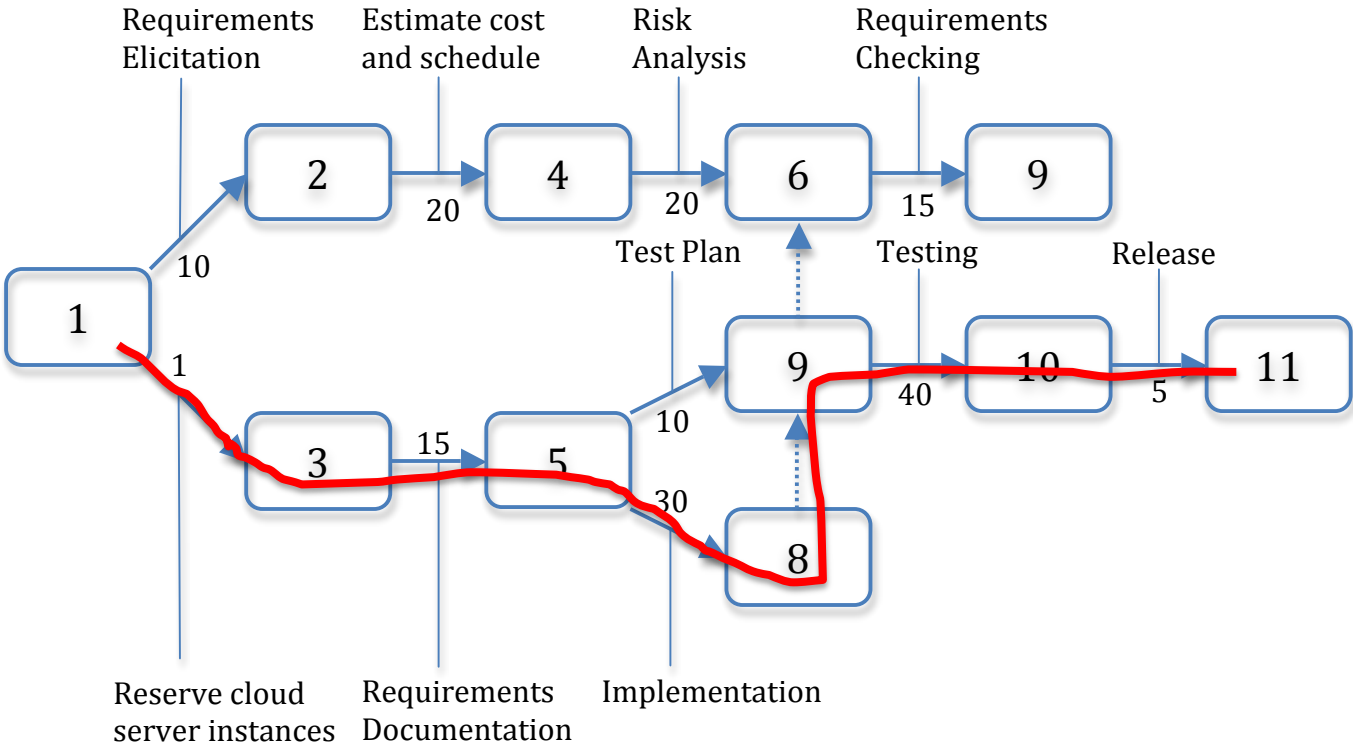
P&D Cost Estimation

1. Experiential Estimate
   - Rely on manager's experience to be accurate
2. Quantitative Estimate
   - Estimate tasks in lines of code (LOC), divide by LOC/person-month
   - COCOMO (Constructive Cost Model): Effort = OrgFactor*CodeSize^{Penalty} * ProdFactor
     - Organization Factor = 2.94, 1.10≤SizePenalty≤1.24, 0.90 ≤ Product Factor ≤1.40

- 92% use experiential vs. formula

P&D Scheduling

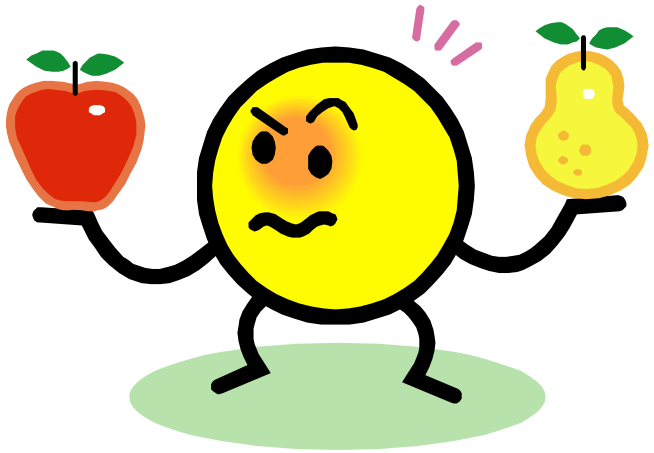
- Use PERT chart to show task parallelism and critical path to make schedule



  - Nodes are milestones, link names are tasks, link numbers are effort, arrows are dependencies

P&D Monitoring Progress

- Compare predicted to actual
  - Time for tasks
  - Expenditures
- Intermediate milestone help all stakeholders see if project is on schedule & on budget



P&D Requirements Checking

- Manager uses tools for *requirements traceability*
- Tool has cross references between
  - Portion of SRS with requirement
  - Portion of code that implements requirement
  - Tests that validate requirement

P&D Risk Analysis & Management

- To improve accuracy of budget/schedule
- Identify risks early to
  - Do extra work to reduce risk
  - Change plan to avoid risk
- Technical: RDB can't scale
- Organizational: J2EE???
- Business: too late for market
- Create Risk Table: % chance x impact
  - Address top 20%, hoping 80% of risk

P&D vs. Agile Requirements and Schedule/Cost Estimation

Tasks	*In Plan and Document*	*In Agile*
Requirements Documentation	Software Requirements Specification such as IEEE Standard 830-1998	User stories, Cucumber, Points, Velocity
Requirements Elicitation	Interviews, Scenarios, Use Cases	
Change Management for Requirements, Schedule, and Budget	Version Control for Documentation and Code	
Ensuring Requirements Features	Traceability to link features to tests, reviews, and code	
Scheduling and Monitoring	Early in project, contracted delivery date based on cost estimation, using PERT charts. Milestones to monitor progress	
Cost Estimation	Early in project, contracted cost based on manager experience or estimates of task size combined with productivity metrics	Evaluate to pick range of effort for time and materials contract
Risk Management	Early in project, identify risks to budget and schedule, and take actions to overcome or avoid them	

Figure 7.14: The relationship between the requirements related tasks of Plan-and-Document versus Agile methodologies.

And in Conclusion: § § 9.4-9.5, 7.1-7.4, 7.7

- SOFA methods: Short, do One thing, Few arguments, consistent Abstraction
  - Metrics point to problem pieces of program
- BDD: User stories to elicit requirements
  - SMART: Specific, Measureable, Achievable, Relevant, Timeboxed
- Points/Velocity to calculate progress (Tracker)
- Lo-Fi UI/storyboard: sketch to elicit UI design
- Lifecycles: Plan&Document (careful planning & documentation) v. Agile (embrace change)

Fallacies & Pitfalls, BDD Pros & Cons, End of Chapter 7

(Engineering Software as a Service § 7.10- § 7.12)

David Patterson

© 2013 Armando Fox & David Patterson, all rights reserved

Pitfalls

- Delivering a story as "done" when only the happy path is tested
  - Need to test both happy path *and* sad path
- Correct app behavior
  when user accidentally does wrong thing
  is just as important as
  correct behavior when does right thing
  - To err is human

Pitfalls

- Careless use of negative expectations
  - Beware of overusing "Then I should not see…."
  - Can't tell if output is what want, only that it is not what you want
  - Many, many outputs are incorrect
- Include positives to check results "Then I should see …"

Pitfalls

- Careless use of positive expectations
  - `Then I should see “Emma”`
    what if string appears multiple times on page?
  - Can pass even if feature not working
- Use Capybara’s `within` helper
  - Constrains scope of matchers in a CSS selector
  - `Then I should see “Emma” within “div#shopping_cart”`
  - See Capybara documentation