Requirements and Specification

Chung-Kil Hur

(Credit: Byung-Gon Chun & Many Slides from UCB CS169
taught by Armando Fox, David Patterson, and George Necula)

SWPP, CSE, SNU

Requirements Engineering

- The most important single part of building a software system is deciding what to build
  - Cripples the process if done wrong
  - Costly to rectify later
- Must be adapted to the software process
  - Elaborate requirements/specs for plan and document
  - Partial set of user stories for iterative processes

Determining Stakeholders and Needs

- Must determine stakeholders
  - Anyone who benefits from the system developed
  - E.g., who's client and who's user ?
- Try to understand what their needs are
- Reconcile different needs/points of view

Techniques

- Interviewing
- User stories
- Strawmen
- Prototypes

Interviewing

- One path is obvious
  - Sit down with client/user and ask questions
  - Listen to what they say, and what they don't say
- A less obvious path
  - Master-apprentice relationship
  - Have them teach you what they do
  - Go to workplace and watch them do the task
- In all types of interviews, get details
  - Ask for copies of reports, logs, emails on process
  - These may support, fill in, or contradict what the user said

Disadvantages of Talking

- Interviews are useful, but users/clients may
  - Not have the vocabulary to tell you what they need
  - Not know enough about computer science to understand what is possible or impossible
- Good idea to gather requirements in other ways, too

Extreme Programming – User Stories

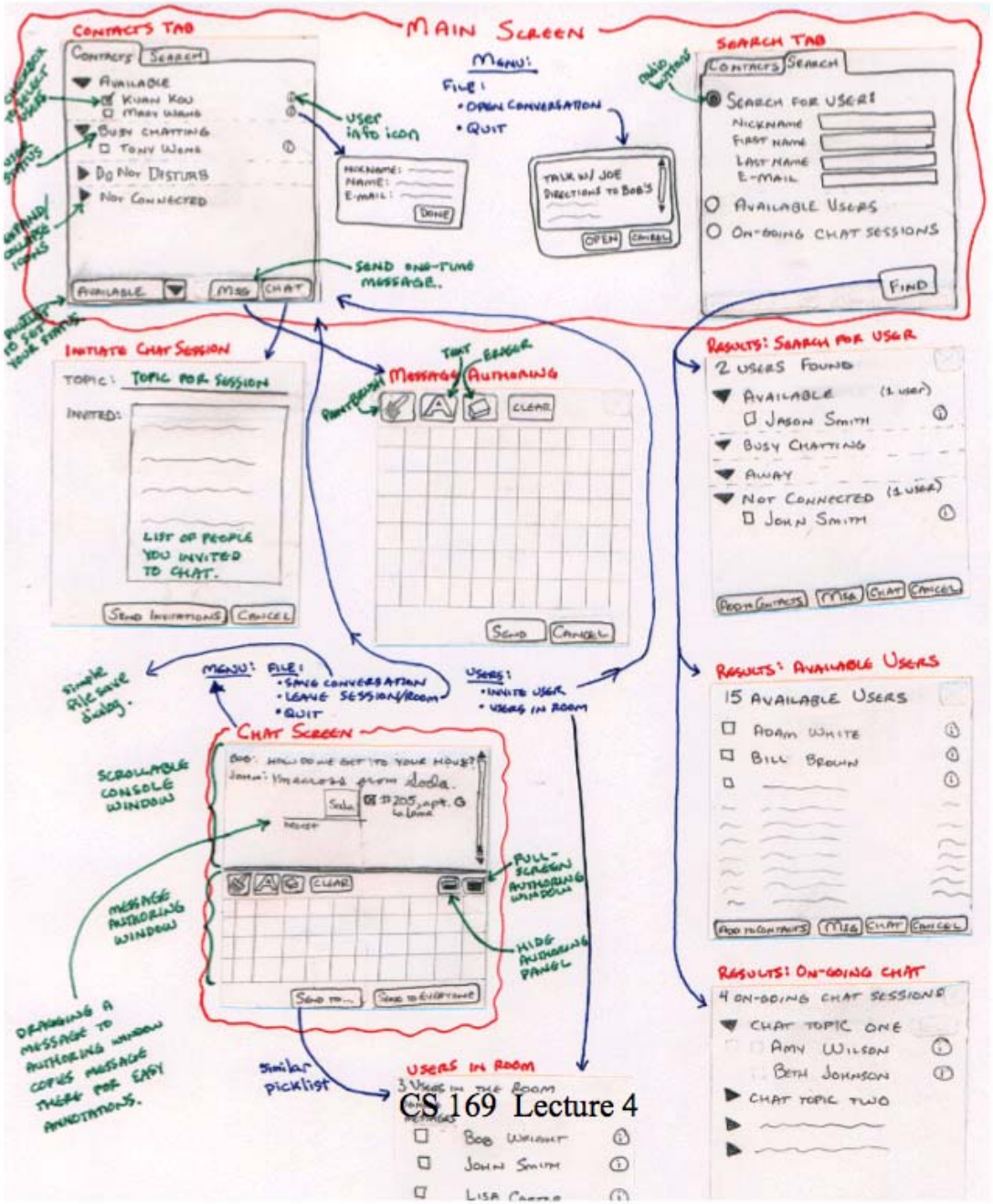
- Recall: client writes user stories
  - Using client vocabulary
- Describe usage scenarios of software
  - Title, short description
- Each user story has acceptance tests
  - Concrete instance of the story
  - Will tell you when the customer thinks story is done

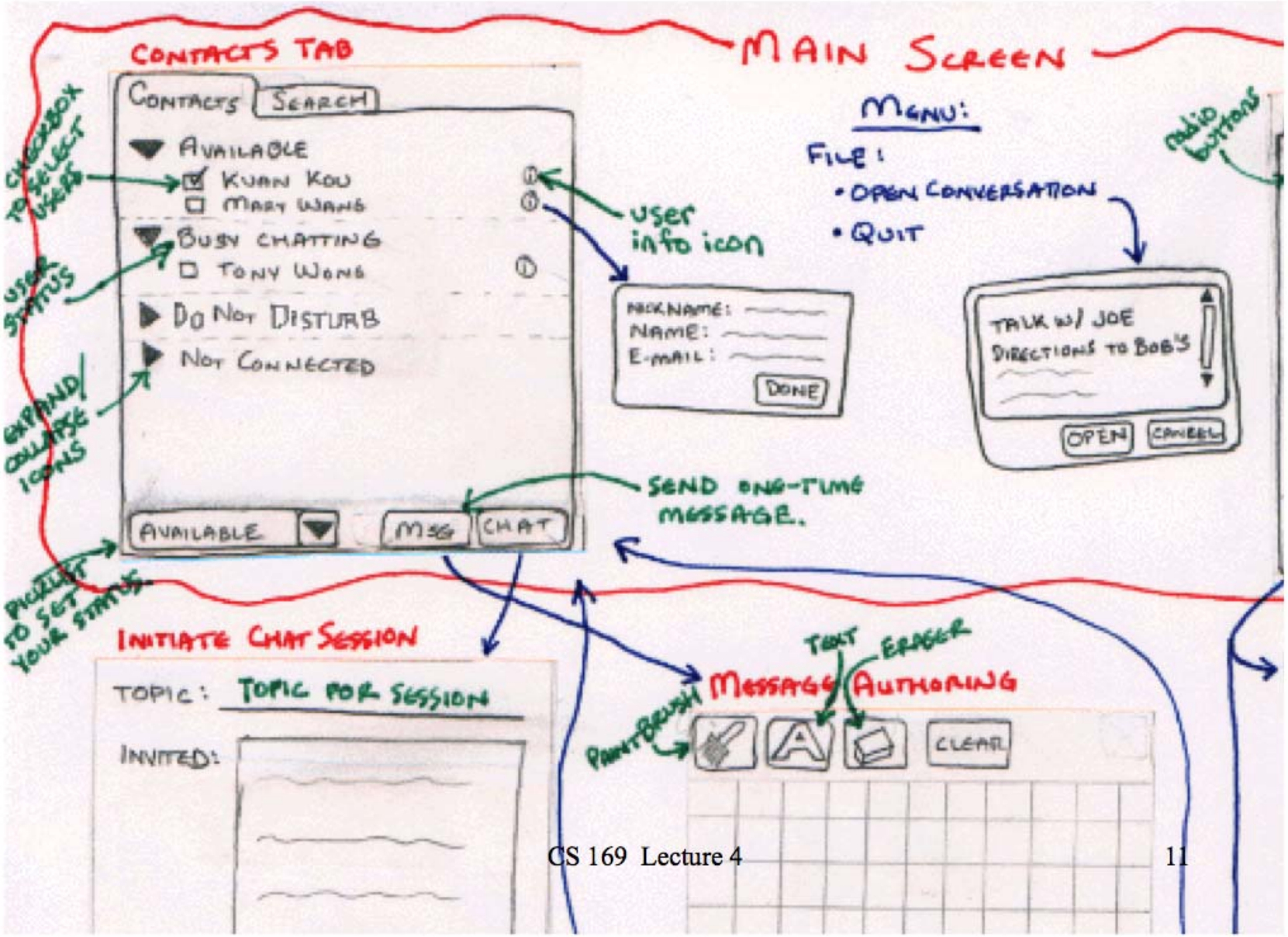
User Story Example

- Title: Delete account
- Actors: authenticated operator
- Preconditions: account must be empty
- Trigger: operator selects “Delete” menu option
- Scenario: ..., confirmation dialog box, ...
- Exceptions: account not empty, unknown user
- Priority: important, but not for first release
- Scalability: infrequent use
- Security: operator must be authenticated and authorized
- Acceptance test(s)

Strawmen

- Sketch the product for the user/client
  - Storyboards
  - Flowcharts
  - HTML mock-ups
  - Illustrate major events/interfaces/actions
- Anything to convey ideas without writing code!


MAIN SCREEN
CONTACTS TAB
SEARCH TAB
MENU:
FILE:
• OPEN CONVERSATION
• QUIT
USER INFO ICON
SEND END-TIME MESSAGE..
INITIATE CHAT SESSION
TOPIC: TOPIC FOR SESSION
INVITED:
LIST OF PEOPLE YOU INVITED TO CHAT.
MESSAGE AUTHORING
RESULTS: SEARCH FOR USER
2 USERS FOUND
JASON SMITH
JOHN SMITH
RESULTS: AVAILABLE USERS
15 AVAILABLE USERS
ADAM WHITE
BILL BROWN
RESULTS: ON-GOING CHAT
4 ON-GOING CHAT SESSIONS
CHAT TOPIC ONE
AMY WILSON
BETH JOHNSON
CHAT TOPIC TWO
CHAT SCREEN
SCROLLABLE CONSOLE WINDOW
MESSAGE AUTHORING WINDOW
FULL-SCREEN AUTHORING WINDOW
HIDE AUTHORING PANEL
USERS IN ROOM
BOB WRIGHT
JOHN SMITH


CONTACTS TAB
MAIN SCREEN
CONTACTS
SEARCH
AVAILABLE
KUAN KOU
MARY WANG
BUSY CHATTING
TONY WONG
DO NOT DISTURB
NOT CONNECTED
AVAILABLE
MSG
CHAT
CHECKBOX TO SELECT USERS
USER STATUS
EXPAND/COLLAPSE ICONS
PICKLIST TO SET YOUR STATUS
USER INFO ICON
NICKNAME:
NAME:
E-MAIL:
DONE
MENU:
FILE:
• OPEN CONVERSATION
• QUIT
TALK W/ JOE
DIRECTIONS TO BOB'S
OPEN
CANCEL
RADIO BUTTONS
SEND ONE-TIME MESSAGE.
INITIATE CHAT SESSION
TOPIC:
TOPIC FOR SESSION
INVITED:
TEXT
ERASER
MESSAGE AUTHORING
PAINTBRUSH
CLEAR

Summary of Requirements

- Find out what users/clients need
  - Not necessarily what they say they want

- Use
  - Interviews
  - User stories
  - Strawmen
  - As appropriate . . .

Specifications

- Describe the functionality of the product
  - Precisely
  - Covering all circumstances
- Move from the finite to the infinite
  - Finite examples (acceptance tests) to infinite set of possible cases
  - This is not easy

Different Views of Specifications

- Developer's
  - Specification must be detailed enough to be implementable
  - Unambiguous
  - Self-consistent
- Client's/user's
  - Specifications must be comprehensible
  - Usually means not too technical
- Legal
  - Specification can be a contract
  - Should include acceptance criteria
    - If the software passes tests X, Y, and Z, it will be accepted

Problems with Informal Specs

- Informal Specs: written in natural language
- Informal specs of any size inevitably suffer from serious problems
  - Omissions
    - Something missing
  - Ambiguities
    - Something open to multiple interpretations
  - Contradictions
    - Spec says both “do A” and “do not do A”

These problems will be faithfully implemented in the software unless found in the spec

Informal Specifications Example

“If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%”

Informal Specifications Example

- “If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%”

What are sales? Orders received but not yet paid for? Only orders paid for?

Informal Specifications Example

- "If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%"

Specification implies, but does not say, that there are monthly sales targets. Are there separate monthly targets, or is the monthly target e.g., 1/3 of the quarterly target?

Informal Specifications Example

"If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%"

"unless A or B" means:

- If!(A||B)
- If(!A||B)
- May be the comma before "or" matters

Informal Specifications Example

“If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5% ”

5% of the target sales, or of the actual sales?

Comments on Informal Specification

- Informal specification is universally disliked
  - By academics
  - By “how to” authors
- Informal specification is also widely practiced
  - Why?

Why Do People Use Informal Specs?

- The common language is natural language
  – Customers can't read formal specs
  – Neither can most programmers
  – Or most managers
  – A least-common denominator effect takes hold
- Truly formal specs are very time-consuming
  – And hard to understand
  – And overkill for most projects

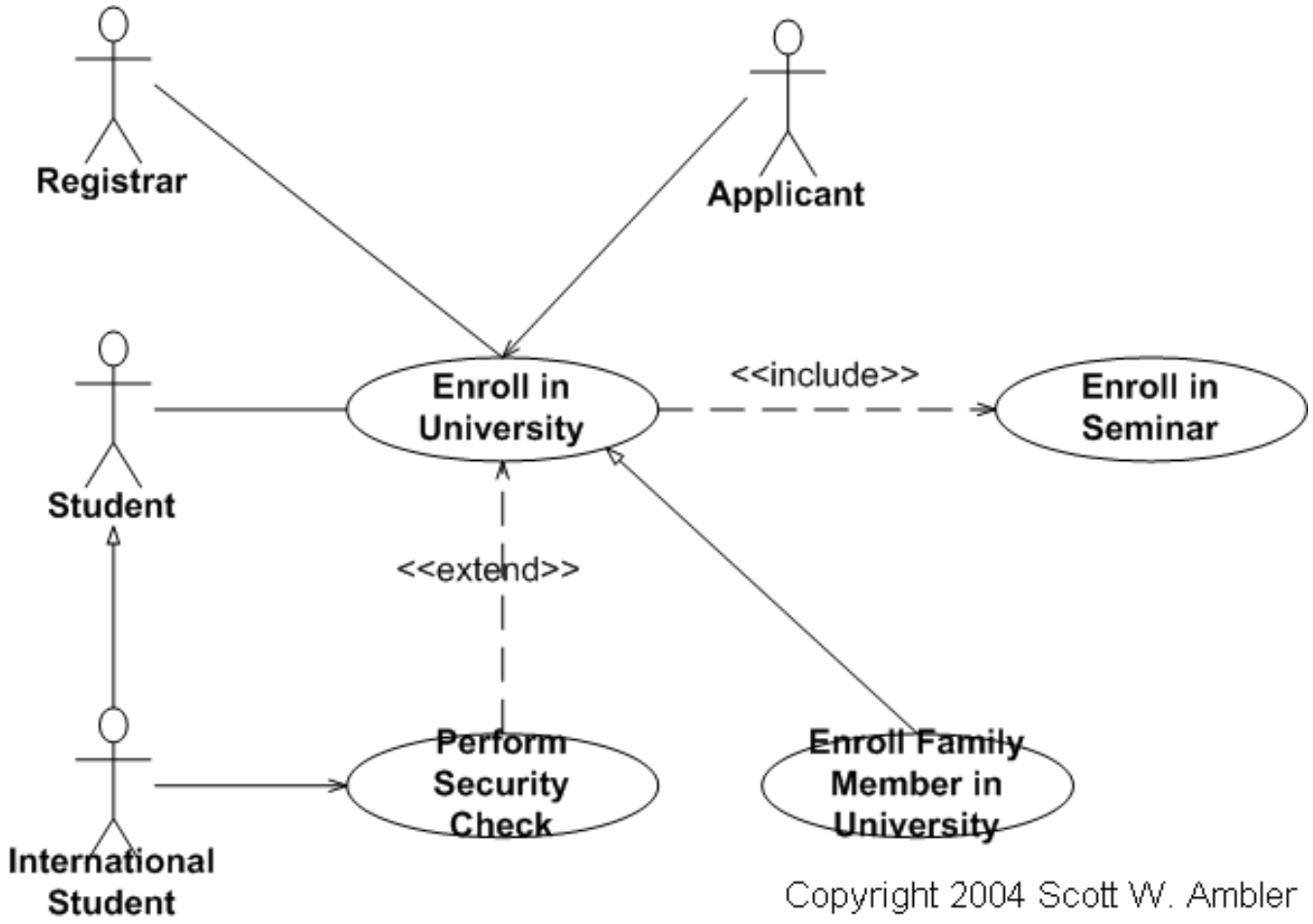
Formal Spec Example (TLA+)

```
---------- MODULE Euclid ----------
EXTENDS Integers

p | q == \E d \in 1..q : q = p * d
Divisors(q) == {d \in 1..q : d | q}
Maximum(S) == CHOOSE x \in S : \A y \in S : x \geq
y
GCD(p,q) == Maximum(Divisors(p) \cap
Divisors(q))
Number == Nat \ {0}

CONSTANTS M, N
VARIABLES x, y

Init == (x = M) /\ (y = N)

Next == \/ /\ x < y
        /\ y' = y - x
        /\ x' = x
     \/ /\ y < x
        /\ x' = x-y
        /\ y' = y

Spec == Init /\ [][Next]_<<x,y>>
ResultCorrect == (x = y) => x = GCD(M, N)
THEOREM Correctness == Spec =>
[]ResultCorrect

==================================
```

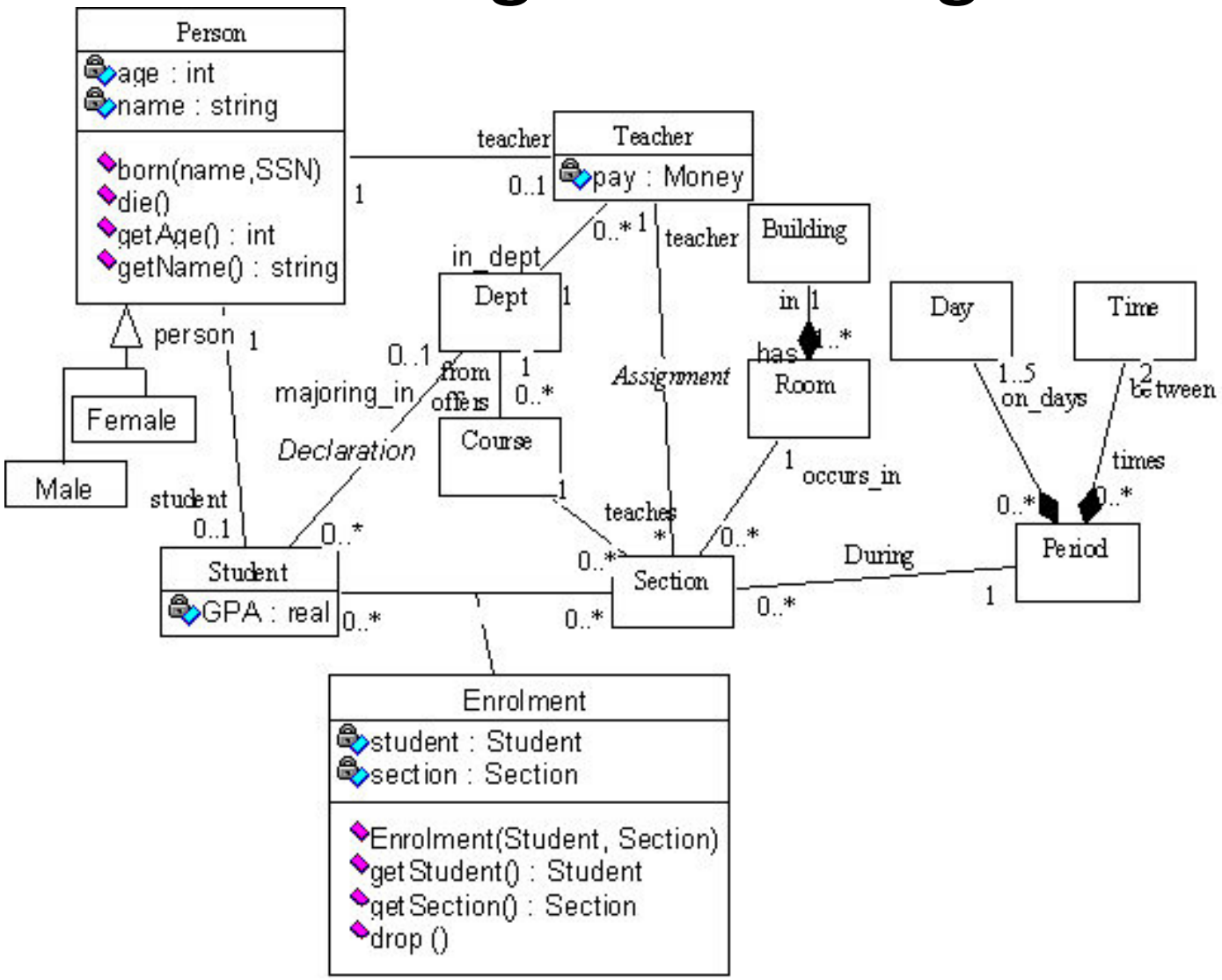
Semi-Formal Specs

- Best current practice is “semi-formal” specs
  - Allows more precision than natural language where desired
- Usually a boxes-and-arrows notation
  - Must pay attention to:
  - What boxes mean
  - What arrows mean
  - Different in different systems!

Semi-formal Spec Example (UML Use Case)



Semi-formal Spec Example
(UML Class Diagram – Design Phase)



Introduction to Behavior-Driven Design and User Stories

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

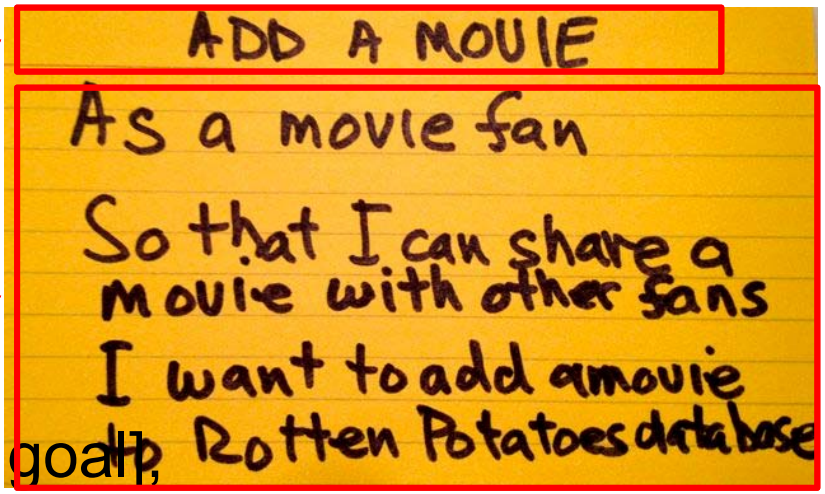
- Work closely, continuously with stakeholders to develop requirements, tests
  - Users, customers, developers, maintenance programmers, operators, project managers, …
- Maintain working prototype while deploying new features every **iteration**
  - Typically every 1 or 2 weeks
  - Instead of 5 major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

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 Design 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
  - (We'll see Backlog again with Pivotal Tracker)
- Prioritize so most valuable items highest
- Organize so they match SW releases over time

Related Issue

- Spike
  - Short investigation into technique or problem
    - E.g. spike on recommendation algorithms
  - After spike done, code *must* be thrown away
    - Now that know approach you want, write it right

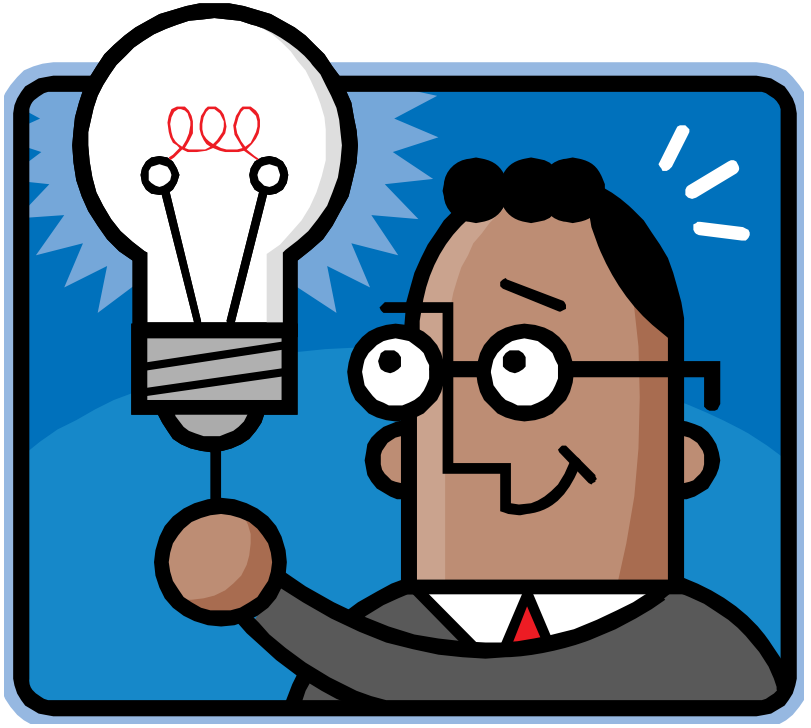
SMART User Stories

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 do X,
  3. *Then* one or more specific thing(s) should happen

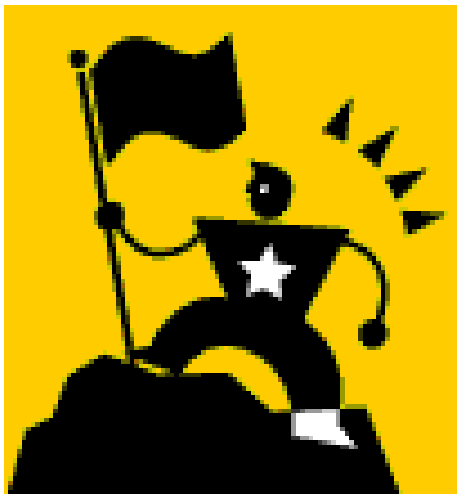
Examples

- Feature: User can search for a movie (vague)
- Feature: User can search for a movie by title (specific)

- Feature: Rotten Potatoes should have good re sponse time (unmeasurable)
- Feature: When adding a movie, 99% of Add M ovie pages should appear within 3 seconds (measurable)

Achievable

- Complete in 1 iteration
- If can't deliver feature in 1iteration, 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
  - Providing more value to your customers

5 Whys to Find Relevance

- *Show patron's Facebook friends*

  As a box office mana

  So that I can induce a
  buy a ticket

  I want to show her w
  friends are going to a given show

  1. *Why?*
  2. *Why?*
  3. *Why?*
  4. *Why?*
  5. *Why?*

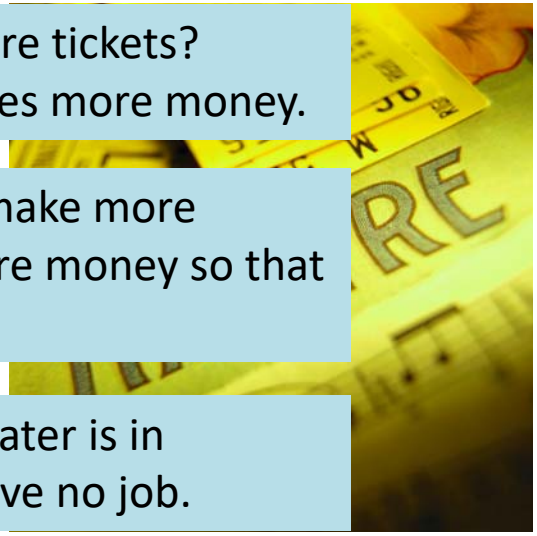
1. Why add the Facebook feature? As box office manager, I think more people will go with friends and enjoy the show more.

2. Why does it matter if they enjoy the show more? I think we will sell more tickets.

3. Why do you want to sell more tickets? Because then the theater makes more money.

4. Why does theater want to make more money? We want to make more money so that we don't go out of business.

5. Why does it matter that theater is in business next year? If not, I have no job.



Timeboxed

- Timeboxing means that you stop developing a story once you've exceeded the time budget. Either you give up, divide the user story into smaller ones, or reschedule what is left according to a new estimate. If dividing looks like it won't help, then you go back to the customers to find the highest value part of the story that you can do quickly.
- The reason for a time budget per user story is that it is extremely easy to underestimate the length of a software project. Without careful accounting of each iteration, the whole project could be late, and thus fail. Learning to budget a software project is a critical skill, and exceeding a story budget and then refactoring it is one way to acquire that skill.

Which feature below is LEAST SMART?

1. User can search for a movie by title
2. The movie site should have good response time
3. When adding a movie, 99% of Add Movie pages should appear within 3 seconds
4. As a customer, I want to see the top 10 movies sold, listed by price, so that I can buy the cheapest ones first

Lo-Fi UI Sketches and Storyboards

Building Successful UI

- SaaS apps often faces users

$\Rightarrow$User stories need User Interface (UI)

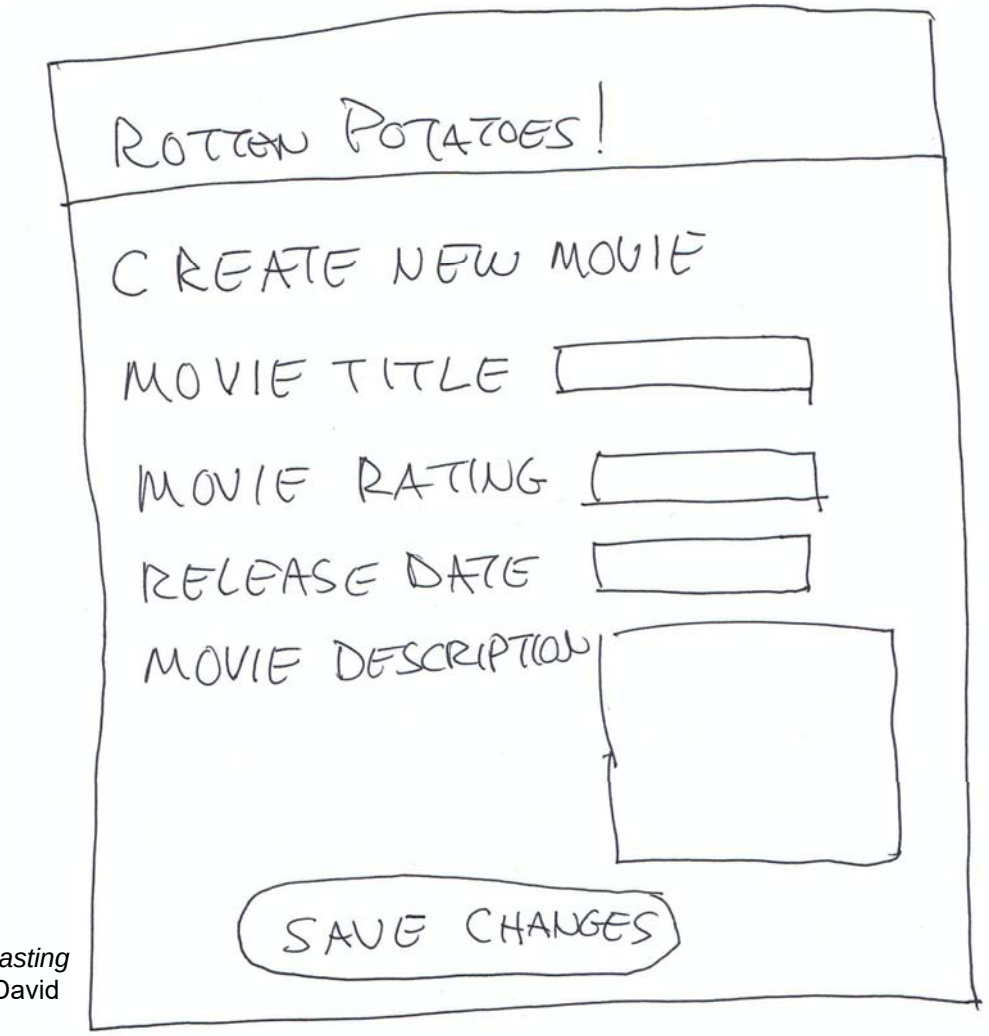
- How get customer to participate in UI design so is happy when complete?
  - Avoid WISBNWIW* UI?
  - UI version of 3x5 cards?
- How show interactivity without building prototype?

* 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

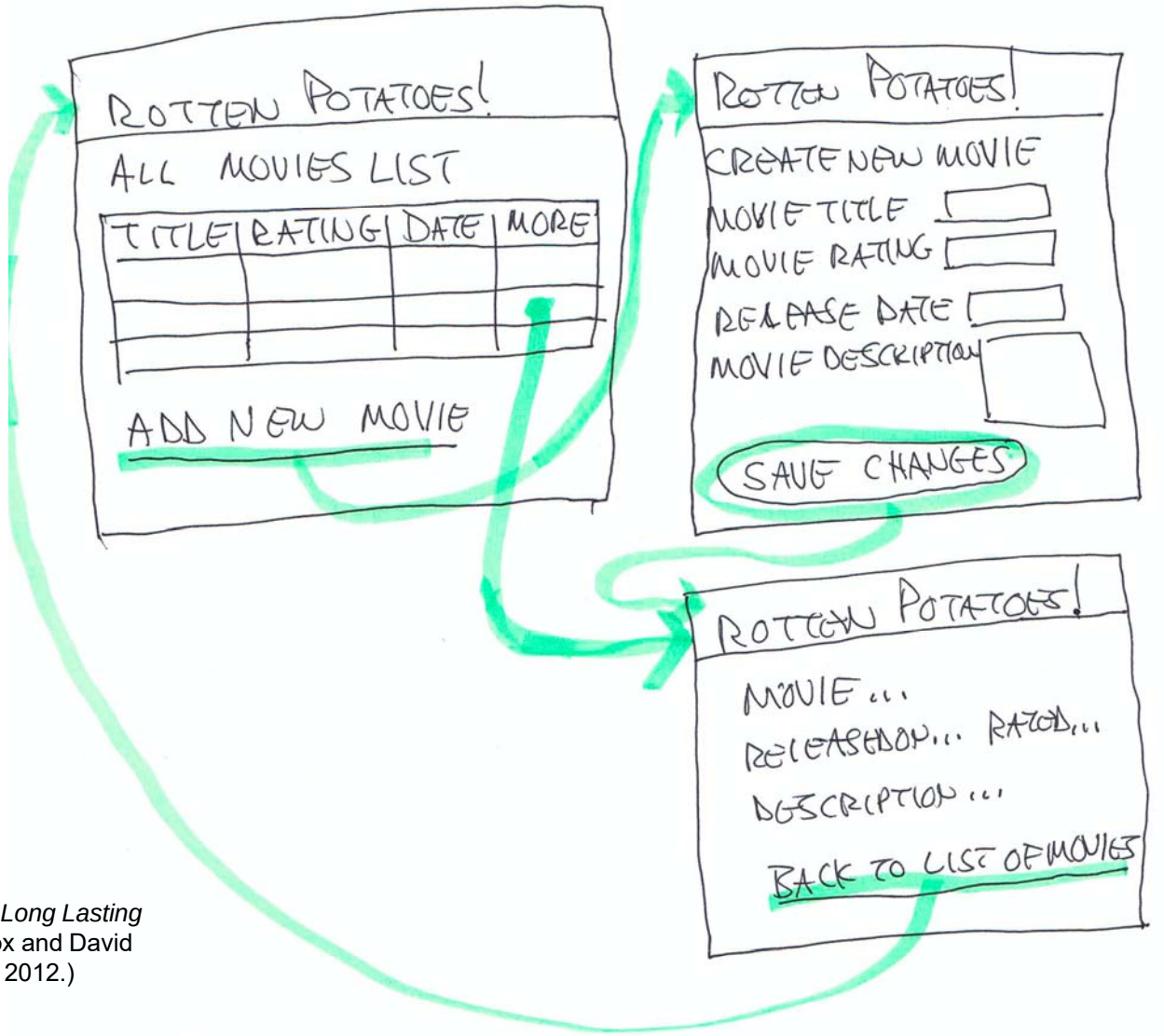


(Figure 4.3, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

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!
  - Also less intimidating to nontechnical stakeholders => More likely to suggest changes to UI if not code behind it
  - More likely to be happy with ultimate UI
- Next steps: CSS (Cascading Style Sheets) and Haml (later)
  - Make it pretty *after* it works

Which is FALSE about Lo-Fi UI?

- ☐ Like 3x5 cards, sketches and storyboards are more likely to involve all stakeholders vs. code
- ☐ The purpose of the Lo-Fi UI approach is to debug the UI before you program it
- ☐ SaaS apps usually have a user interfaces associated with the user stories
- ☐ While it takes more time than building a prototype UI in CSS and Haml, the Lo-Fi approach is more likely to lead to a UI that customers like

Summary

- BDD: User stories to elicit requirements
  - SMART: Specific, Measureable, Achievable, Relevant, Timeboxed
- Lo-Fi UI/storyboard: sketch to elicit UI design
- Lifecycles:
  - Plan&Document (careful planning & documentation)
  - Agile (embrace change)

Fallacies & Pitfalls, BDD Pros & Cons

Pitfalls

- Customers confuse digital mock-ups with completed features
  - Nontechnical customers think highly polished digital mock-up = working feature
- Use Lo-Fi mockups, as clearly representations of proposed feature

Pitfalls

- Sketches without storyboards
  - Need to reach agreement with customer on interaction with pages as well as page content
- Storyboards / “animating” sketches reduces misunderstandings

Pitfalls

- Adding cool features that do not make the product more successful
  - Customers reject what programmers liked
- Trying to predict what code you need before need it
  - BDD: write tests *before* you write code you need, then write code needed to pass the tests
- User stories help prioritize & BDD minimizes what you code => reduce wasted effort

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

BDD Good & Bad

- User stories - common language for all stakeholders, including nontechnical
  - 3x5 cards
  - LoFi UI sketches and storyboards
- Write tests before coding
  - Validation by testing vs. debugging
- Difficult to have continuous contact with customer?
- Leads to bad software architecture?
  - Will cover patterns, refactoring 2nd half of course