Software Engineering
ECE444

Behavior-driven Design and User Stories

Most software projects fail

- don't do what customers want
- or projects are late
- or they are over budget
- or they are hard to maintain and evolve
- or all of the above

➔most software produced is never used; e.g.

  - long-gun registry
  - eHealth
  - . . .

Agile software engineering

Traditional SE:

- focuses mainly on verification: "building the thing right"
- full requirements → full design → implementation

- Agile SE:
  - focuses on validation: "building the right thing"
    i.e., focus is on behavior, not implementation
  - based on 12 principles that fit on one page:
    → https://agilemanifesto.org
    compiled by 17 anarchistical software engineers in 2001

Agile Manifesto, 2001

“We are uncovering better ways of developing SW by doing it and helping others do it. Through this work we have come to value

- Individuals and interactions over processes & tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Revolutionary: like Martin Luther nailing his Ninety-Five Theses on 31 October 1517 to the door of the Catholic Church, sparking the Reformation movement in Christianity

Note: change is not a problem to be solved, but a fact to be coped with

Agile life cycle

Approach:

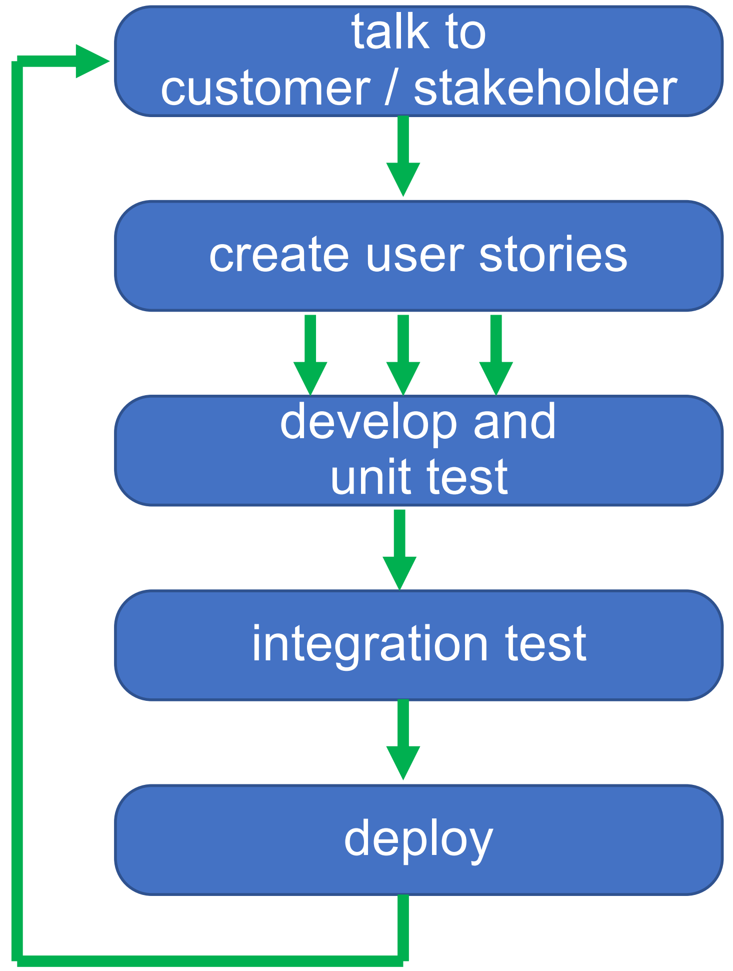
- maintain a working prototype
  even with much functionality missing
- deploy new features in small increments
  (that can be completed within a day or so)
- keep a long list of features/enhancements/extensions
  to be implemented and prioritize them
- work closely with stakeholders
  to develop requirements & tests;
  often show them a prototype to validate building right thing
  and what should be developed next.

“Extreme Programming” (XP). . .

version of Agile lifecycle:

- If short iterations are good, make them as short as possible (days vs. weeks)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other’ s shoulders.

Behavior-Driven Design (BDD)



1. ask questions about behavior of application to all stakeholders (before and during development). Don't ask about implementation.

2. write down requirements: user stories
   - each written on 3x5 card or sticky note
   - written by/with customer
   - each with following format

feature name
As a [kind of stakeholder]
So that [I can achieve some goal]
I want to [do some task]

Why 3x5 Cards?

- (from User Interface community)
- Nonthreatening
  ➔ all stakeholders participate in brainstorming
- Easy to rearrange
  ➔ all stakeholders participate in prioritization
  ➔ enhances brainstorming
- easy to change during development, since user stories must be short
  - because we often get new insights during development

Each user story should be:

a. understandable by customer
b. testable
c. small enough to implement in a day or two
d. SMART:
   - S: Specific
   - M: Measurable (e.g., responds in 2 seconds)
   - A: Achievable (can be done in one iteration)
   - R: Relevant (must have business value)
   - T: Timeboxed
     - → if not completed within timebox
       - → stop development
       - → throw all code and design away
       - → redesign / repartition / redo requirements
       - → start over again

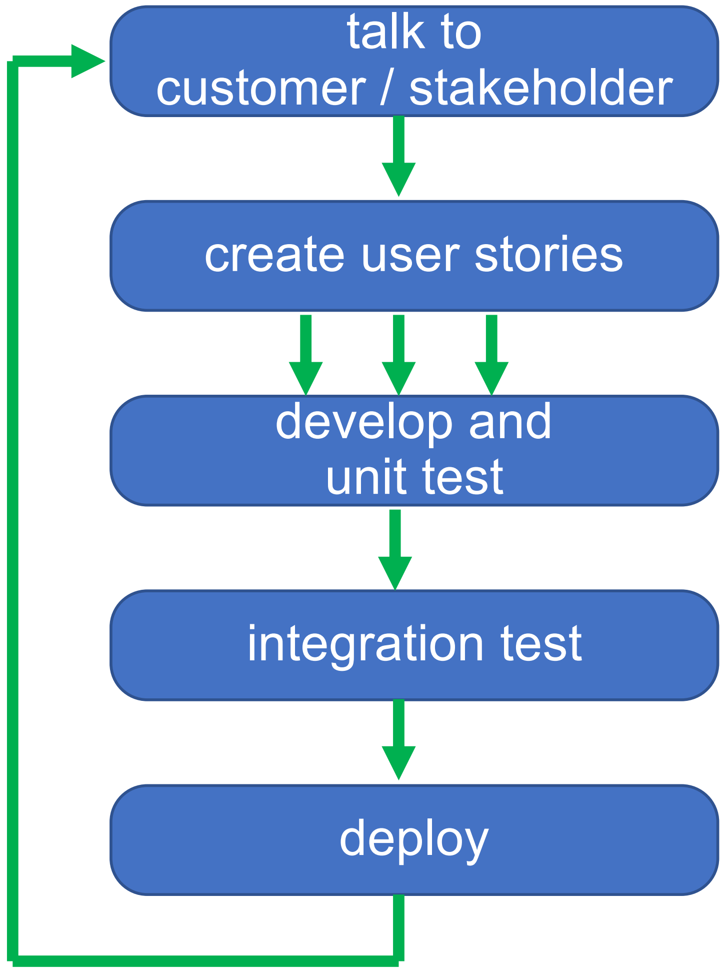
Alternative:

INVEST:
- Independent
- Negotiable
- Valuable
- Estimatable
- Small
- Testable

Stories better than 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”

Behavior-Driven Design (BDD) II



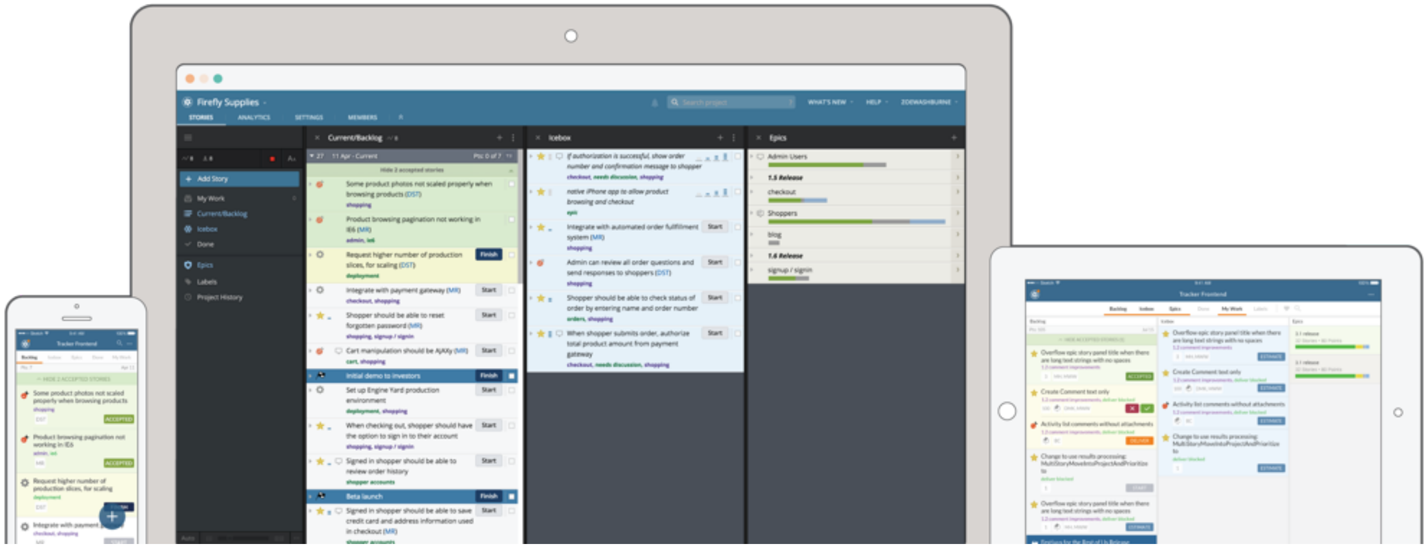
3. Scrum sessions:
   - team circles around sticky notes, reprioritizes and agrees who will work on what next
   - have one at least once a day
   - lasts 10-15 minutes max
   - cards may be modified
   - each team member states:
     - what they did yesterday
     - what they will work on today
     - what impediments are in the way

Note that just about every company does scrum today.
Just so they can say they are agile.
But scrum does not make agile;
agile implies scrum.

Tools to track the 100's of user stories

Many exist:

- Asana
- Github Project ← convenient
- Jira
- Pivotal Tracker ← recommended
- Trello



- prioritize
- track progress

Categories: Done Current / Backlog Inbox Icebox

Measuring Productivity through Velocity

- Iteration: timeframe for scheduling
  - traditional: 2 months
  - modern: 2 weeks
  - best: 1 day
- Assign points to each story for rough workload estim.
  - 1: straight-forward (1/2-1 day)
  - 2: medium (1-2 days)
  - 3: hard (3-5 days)
- Velocity = #points completed per iteration
  - measure of productivity or rate of progress
  - primarily used for self evaluation
- Developer does not decide when done;
  - product owner does

More on points

- Estimating user story implementation effort is hard
- Initially you will be off but improve through learning:
  - See what you actually accomplish in points per week vs. what you guess as optimistic programmers
  - Learn through mistakes; e.g. "gave a story 1 point but took 2 weeks; why was it hard?"
- Teams vote: hold up fingers, take average
  If a big disagreement (2 and 5), discuss more…
- If user story expected to take much effort
  ➔ split up story and group into "epics"

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

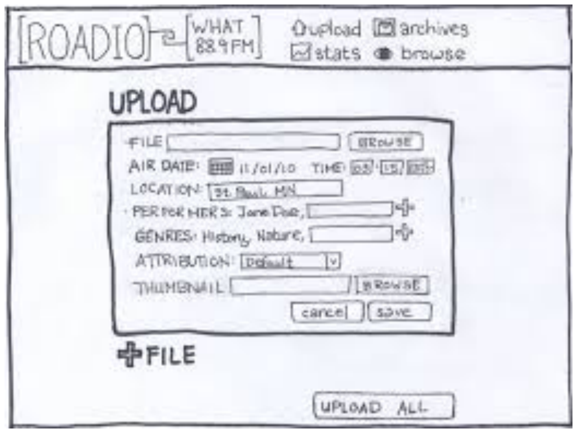
Attach documents to user stories

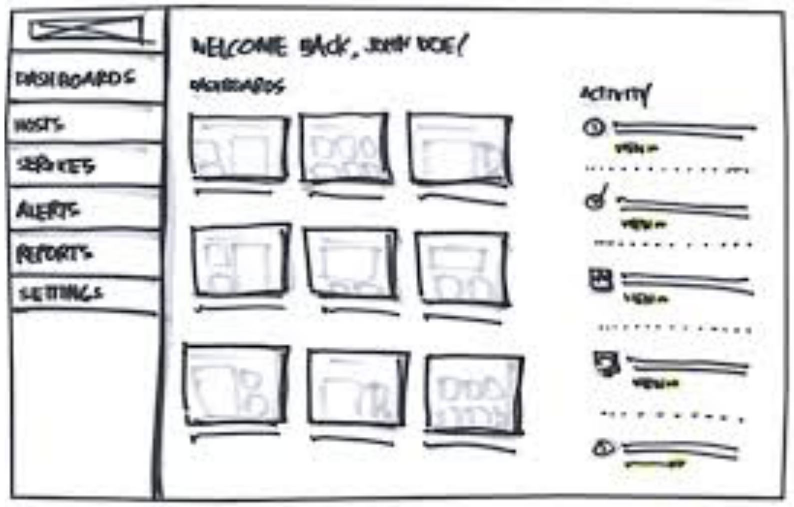
- design docs
- Lo-Fi UI sketches
- Story boards

Using one of the many tools like:
- Pivotal tracker
- Wiki
- Google Docs
- Basecamp
- Slack
- . . .

LoFi Sketches

- Using:
  - crayons, construction paper or poster-sizes postit sheets, scissors, etc.
  - drawing prog on tablet



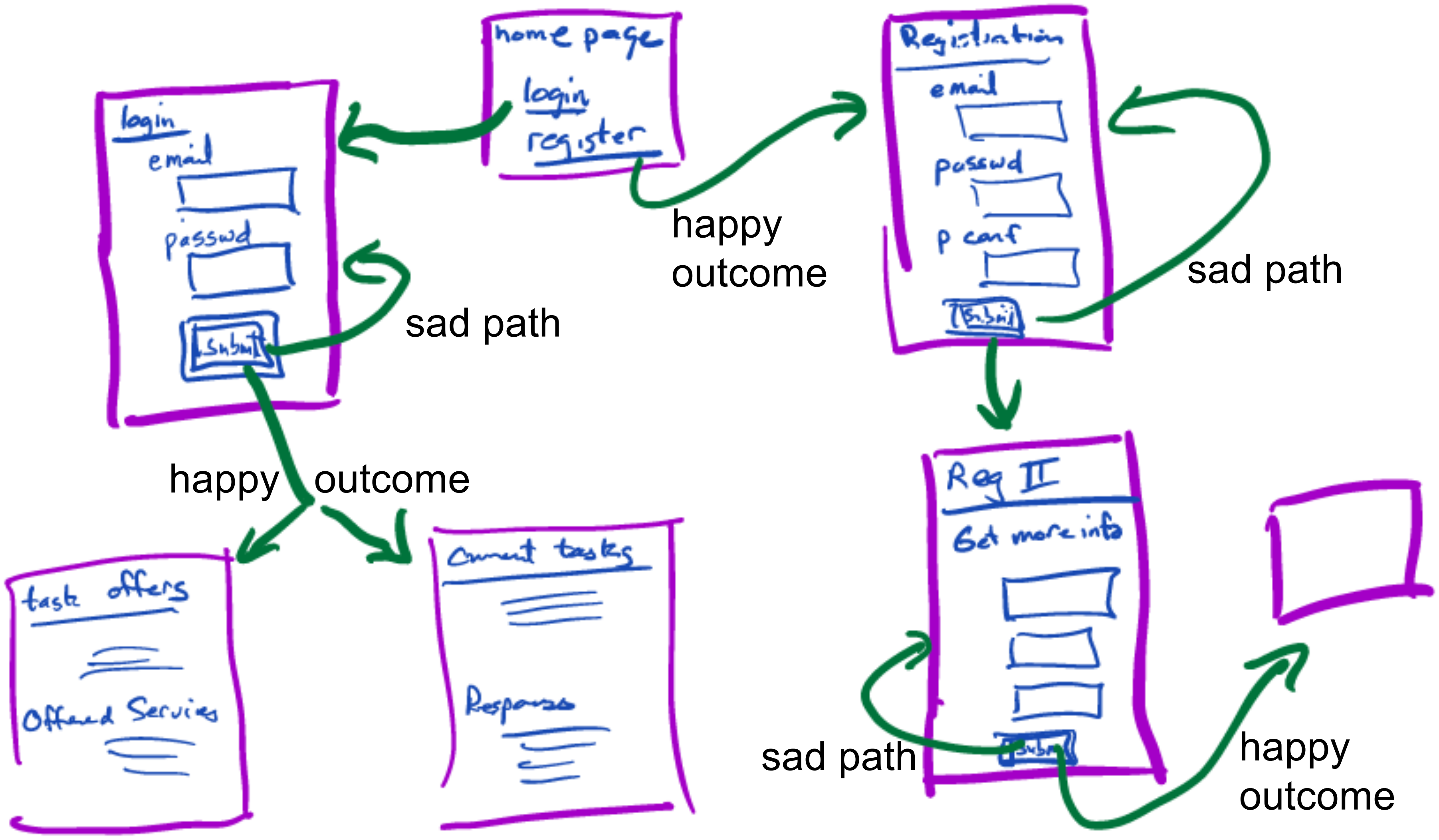


- Simple:
  - UI version of 3x5 cards
  - Fast
  - Allows customer to participate in UI design
  - With cust. involvement: no WISBNWIW* UI?

* What I said but not what I want

LoFi storyboard

- To show how UI changes based on user actions
- Like scenes in a movie, but not linear



LoFi before HTML

- It may be 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 can make it look nice later . . .

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”

Ready to implement?

Some recommendations:

- Don't make UI pretty initially – make pretty at very end
  Don't use CSS initially
  Just get appr. layout right
  (more recommendations on UI will be presented later. . .)

- For layout: always assume mobile first.

- Always implement "sad path" first
  → easier
  → gets to working functionality faster

Software project cost estimates

Big dilemma:

- your client wants to reduce risks w.r.t. costs:
  ➔ wants a price for all software needed
- you want to minimize risk
  ➔ want time & material, not fixed bid

Note: client here is

- actual client if you are in "consultancy business"
- management (so they can make projections, prioritize projects, promise deliverables, etc.)

Big dilemma:

- cannot do agile with fixed upfront cost promise
  yet agile leads to:
  - higher developer productivity
  - more desirable software
- fixed cost guarantee mandates traditional approach to software creation (that has proven not to work very well).

Traditional approach

1. Requirements elicitation
    - interview all stakeholders
    - create scenarios
    - create use cases
2. Requirements documentation: specification
    - often 100's of pages – basis of a contract
    - difficult to do:
        - must be understandable to stakeholders
        - must be useful for engineers
    - there is an IEEE standard: 29148:2011 to ensure all req's are:
        - valid and necessary
        - consistent
        - complete
        - feasable

Traditional approach II

3. Architect the software
4. Come up with cost estimate
    - see below…
5. Scheduling:
   given tasks and projected effort to complete them:
    - use a scheduling tool to identify dependencies and which tasks can be worked on in parallel – e.g., PERT chart
    - assign people to tasks
6. Monitor progress
7. Manage change management

Coming up with cost estimate

a. experienced person guesses
   and multiplies by three or four

b. experienced architect decomposes spec into
   n month-long person tasks

c. use quantitative approach: COCOMO
   (Construction Cost Model)

$$\text{Effort} = \text{Org\_factors} \times \text{codesize}^{\text{size\_penalty}} \times \text{team\_capability}$$

but used by less than 10% of companies.
and less than 20% of projects completed within budget

COCOMO II: adds 3 more formulas and 23 new variables

Coming up with cost estimate ||

Problem: Building software is simply not like building a bridge or a condo tower

Also: None of these methods take into account the fact that some programmers are 50-100 times as productive as others

➔ such cost estimates rarely work well

Bottom line: "time and material" billing much better than "fixed bid" billing