Lecture Notes
December 8, 2020

Volunteer final demos: Roller Coaster, JVM, Magikarp

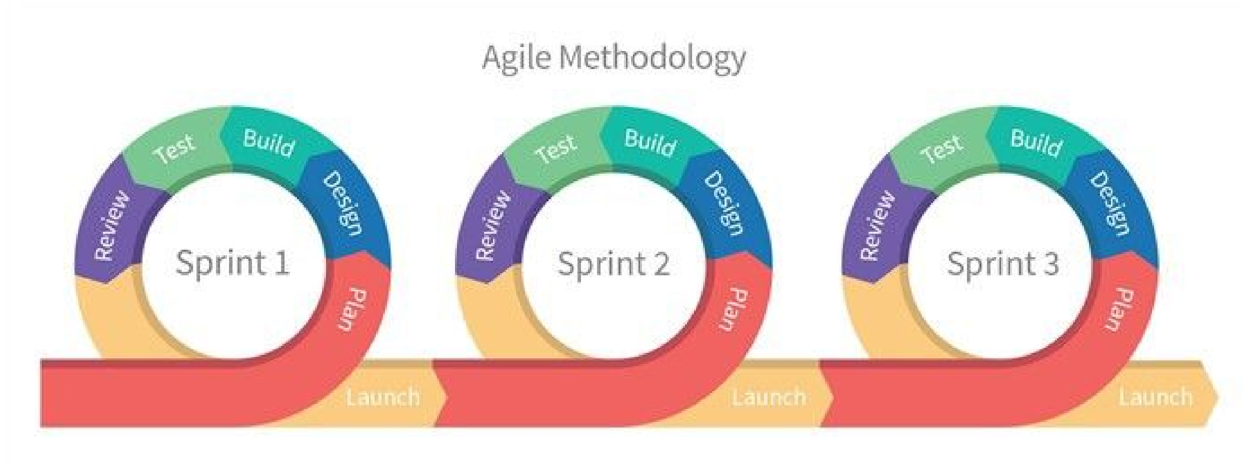
Working as a team: iteration planning

A traditional software engineering course starts with several lectures concerned with software process and iteration planning.

I spent part of one lecture on software process (September 29) and have otherwise pretty much ignored planning. Why? "YAGNI" = You Aren't Gonna Need It

One important mantra of agile software development is do not presumptively add any functionality to your product that might be needed later but is not needed now - *unless* it would be much more expensive to add later (e.g., security). I didn't add functionality to your process, since you weren't going to do it anyway during your two-iteration MVP project.

But real software engineering does require real planning.

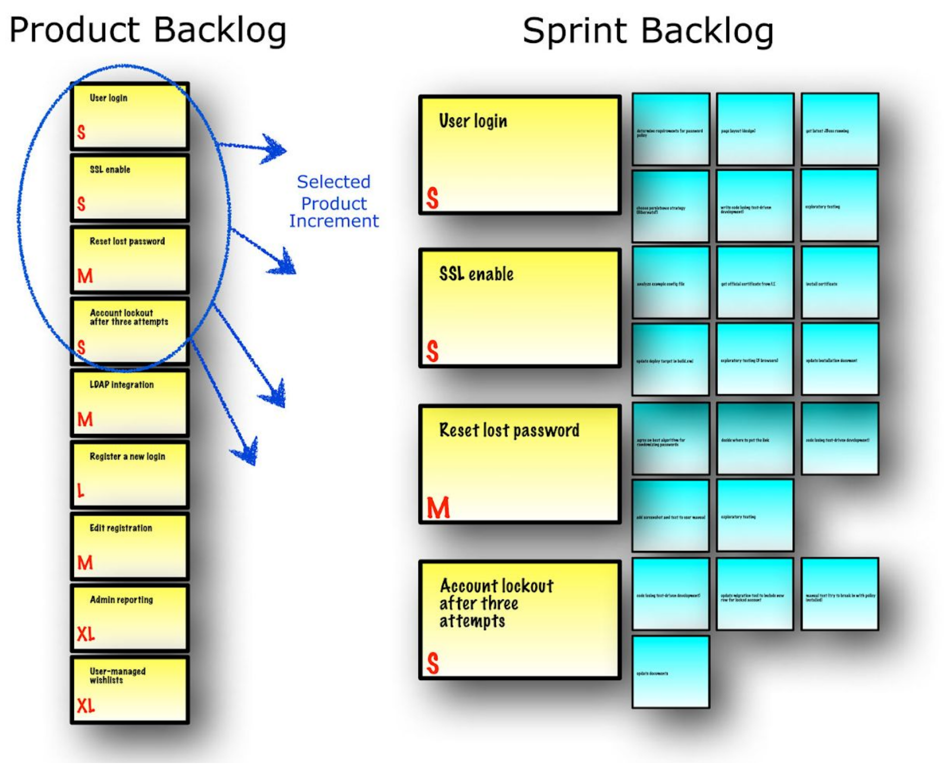


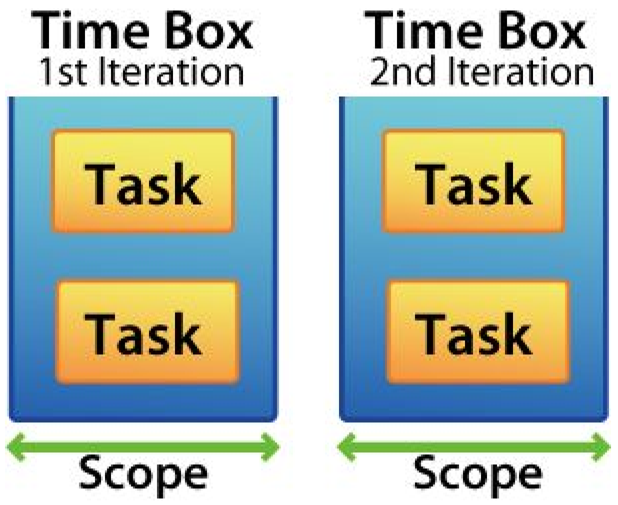
An agile project builds the software product in a series of iterations ("sprints"). Break down big, complex projects into bite-sized chunks.

Ideally, a new version of the product is delivered to customers after every iteration (possibly even after every successful build for devops), but typically it takes several iterations to develop each release

Planning for the first release: Start with a minimum viable product (MVP) set of user stories, and fit roughly into a series of iterations. (Short video motivating user stories.)

For a new project (that is not very similar to some previous project), the initial schedule may be pure guess





Since iterations are fixed-length (typically two weeks), there are a maximum number of hours/days available =
# of developers (or pairs)
x 10 working days
x 8 hours/days

More realistically,
# developers (or pairs)
x 9 working days
x 7 hours/day

Why?

Why? Meetings

In many organizations, every sprint starts with half-day pre-sprint planning and ends with half-day post-sprint review and retrospective, plus “daily standups” and other meetings take a chunk out of productive development time

To fit user stories into sprints, the developers need to *estimate* how long each user story will take to complete

Estimates typically come from previous experience, e.g., formal or informal metrics from past projects

Sometimes the estimation is done alone by a "team lead", sometimes it's done by the whole team, in either case this usually takes a half-day for *someone*

One agile whole-team estimation approach is "planning poker" - playing cards marked ½, 1, 2, 3, 5, 8, 13, 20, 40, ∞, ? days.

For each user story, each developer independently plays a card - then team discusses and comes to a consensus

	PLANNING POKER There's always someone who shows their cards too early...

Many organizations use relative “story-points” instead of absolute time - 1 point for smallest/simplest stories, 3 points for a story that seems about 3 x as long/difficult, etc.

Small
1 pt
No sweat
Medium
3 pts
Nothing we can’t handle
Large
5 pts
This is going to take some effort

Then convert story-points to days/hours afterwards (how long did it really take?) to guide future planning

Even when using time-based estimation, still recalibrate at end of every iteration, e.g., if estimated ½ day really took 2 person-days, estimated 1 day really took 4 person-days, etc., need to multiply estimates by 4

If wide divergence among developers, something is wrong

Estimates longer than 8 days (nearly an entire iteration) are suspect:

- ➢“I have no clue how long this will take”
- ➢Probably need more information and/or break down into smaller user stories

Spike solutions is an agile approach to longer/riskier stories - invest ½ day or full day on quick and dirty *throwaway* version separate from main development, to better estimate how long real version will take

After time estimates assigned to every user story for first iteration, cut down scope so time estimates total to no more than 70% (or 80%, etc.) of time available

Why not 100%? *Project velocity* accounts for lost time

If team runs out of work to do, add more stories from backlog

If team cannot finish everything, move some scheduled stories to backlog

How many estimated time units or story-points did the team really finish and demo/deliver? This is the project velocity for that iteration

$$\frac{\sum \text{Story points}}{\text{Team velocity}} \times \text{Safety coeficient} = \#\ \text{days}$$

e.g.:

$$\frac{46+38+57}{10} \times 1.3 = 18\ \text{days}$$

Why do we need project velocity?

Computers have lives (hardware and software maintenance and downtime)

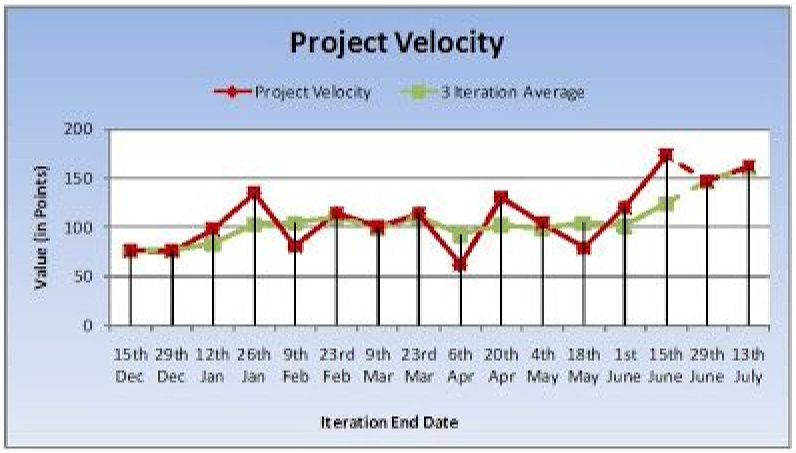
Developers have lives (personal days, sick days, unproductive days)

Projects have lives (training, impromptu meetings, customer interruptions)

Organizations have lives (CTO unexpectedly schedules full-day all-hands meeting)

At end of each iteration or beginning of next iteration, developers re-estimate time or story points for remaining MVP features. Customer or "product owner" may reconsider what must be included in MVP - emphasis should be on *viable* not *minimal*

Then calculate project velocity: Use project velocity from Nth iteration to plan schedule for N+1th iteration OR use average project velocity of several recent iterations

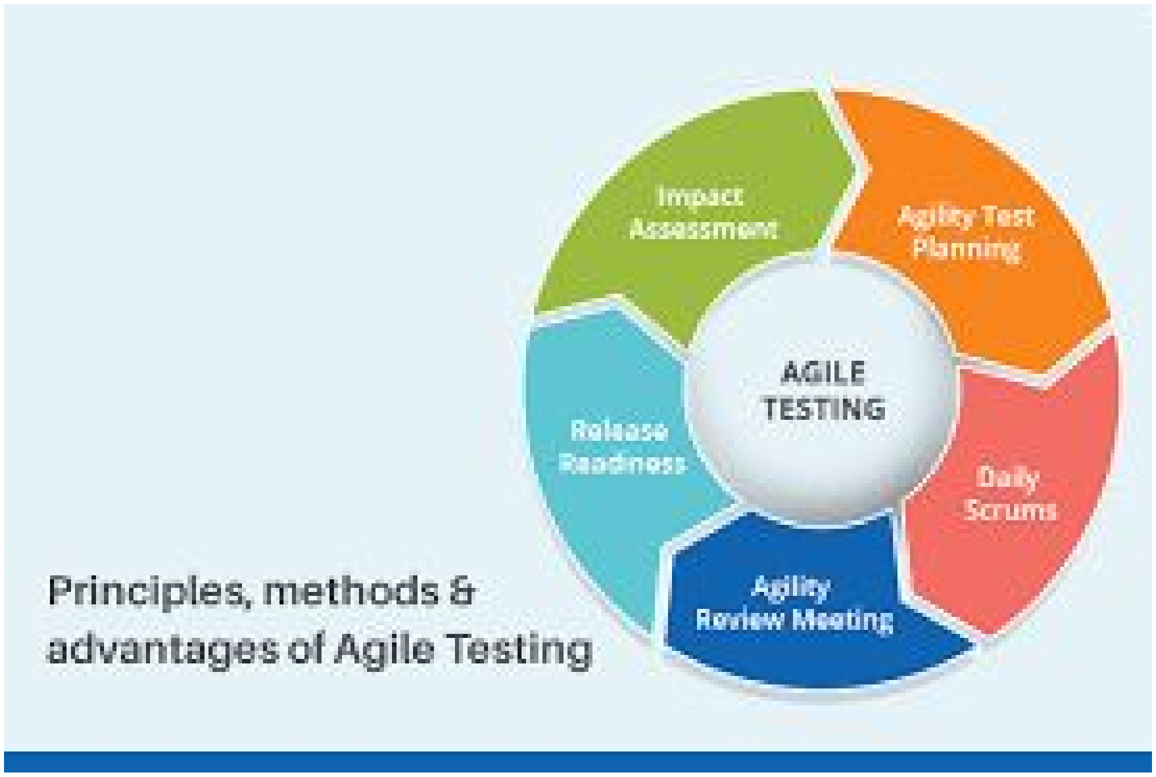


Planning for later releases: Customer or product owner re-prioritizes remaining / newly added user stories, team re-estimates (with greater knowledge) for next set of iterations

How do testing and debugging fit into iterations?

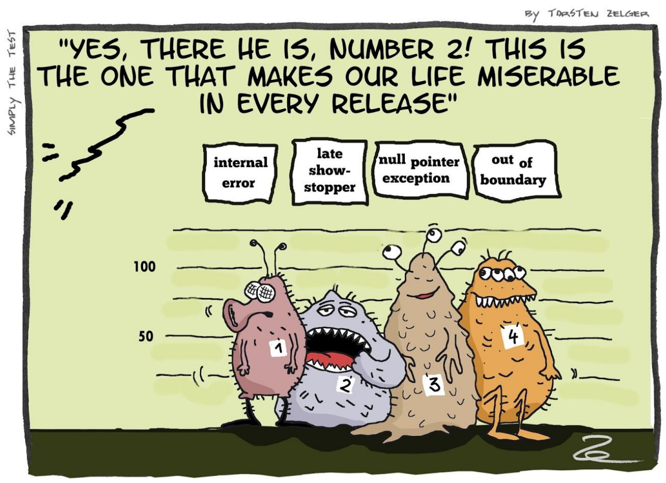
Unit testing (and corresponding debugging) are part of completing a user story. Code review is often also required before considered "done" (sometimes before commit). Need to include this time in estimate, not just coding time

Bugs may be reported by a separate testing team or by users. The separate testing team has its own iterations parallel to development team.



Debugging these bugs can be defined as special kind of user story - also prioritized and estimated

Typically fit stories for show-stopper bugs into current iteration by pushing some previously planned feature stories to backlog, to be re-considered for next iteration



Lining up for the aftermath session

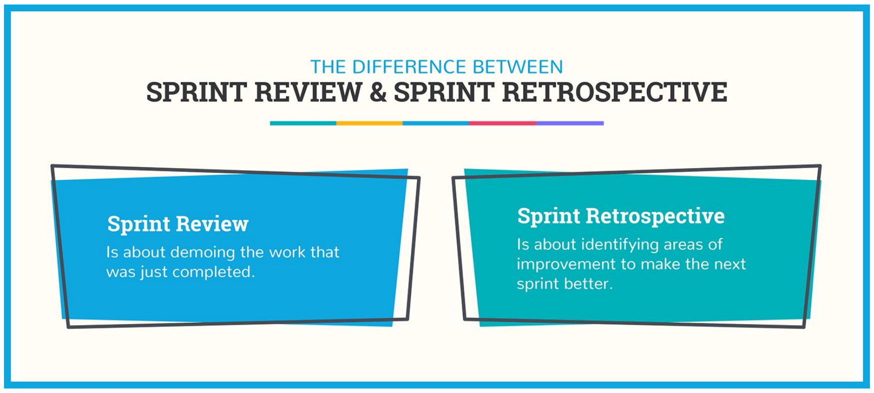
Include remaining highest priority bug stories in later iteration just like new features.

Often a “hardening” iteration is the last iteration before release, with only testing and bug fixes. Many organizations also devote one day per iteration, or one day per week, to debugging. Some have entire iterations devoted to bug-fixing alternating with feature iterations

I mentioned earlier: In many organizations, every sprint starts with half-day pre-sprint planning and ends with half-day post-sprint review and retrospective, plus "daily standups" and other meetings take a chunk out of productive development time

A post-sprint "review" is a meeting with the customer or product owner that reviews the progress made and what should be done next, usually with a demo. This meeting focuses on the *product*, and how to improve the product.

What is a post-sprint "retrospective"? This meeting focuses on the *process*, and how to improve it.


THE DIFFERENCE BETWEEN
SPRINT REVIEW & SPRINT RETROSPECTIVE
Sprint Review
Is about demoing the work that was just completed.
Sprint Retrospective
Is about identifying areas of improvement to make the next sprint better.

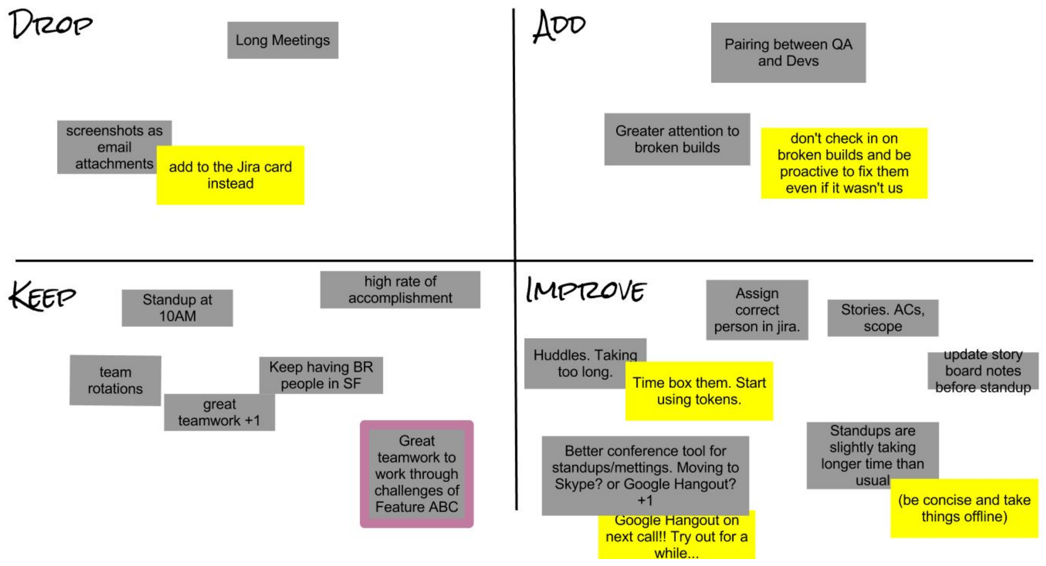
Process Improvement can and should be part of any process, whether based on waterfall, agile, or "transitioning" between process models, but is most often associated with agile

The general idea is to set **SMART** goals = Specific, Measurable, Attainable, Realistic/Relevant, Timely

SaMoLo feedback - same as (reinforce), more of (encourage or increase), less of (discourage or reduce)

What should the team

- start doing?
- stop doing?
- continue doing?



Drop
Long Meetings
screenshots as email attachments
add to the Jira card instead
Add
Pairing between QA and Devs
Greater attention to broken builds
don't check in on broken builds and be proactive to fix them even if it wasn't us
Keep
Standup at 10AM
high rate of accomplishment
team rotations
Keep having BR people in SF
great teamwork +1
Great teamwork to work through challenges of Feature ABC
Improve
Assign correct person in jira.
Stories. ACs, scope
Huddles. Taking too long.
Time box them. Start using tokens.
update story board notes before standup
Better conference tool for standups/mettings. Moving to Skype? or Google Hangout? +1
Google Hangout on next call!! Try out for a while...
Standups are slightly taking longer time than usual
(be concise and take things offline)

Several techniques for organizing retrospective meetings are outlined here.

Skip this

So far we have been talking about “engineering sprints”. Engineering sprints are sometimes preceded by or interleaved with “design sprints”. One minute video here.

Concept developed by Google Ventures (GV) for their startup portfolio

> Many GV companies are **not** in the software industry, design sprints apply to any industry
>
> Intended to answer business questions quickly (in one week) without full development and launch cycle

0th day (ahead of time) - assemble team, get space, buy supplies, etc.

Core team members need to clear schedules for week

Facilitator runs meeting throughout, *Decider* makes ultimate decision in any debates

Monday morning

“Start at the end” - what is the long-term goal? What are the risks?

“Map the challenge” - workflow sketch of customers/key players interacting with product

Monday afternoon

“Ask the experts” - bring in a few domain and business experts for short interviews

“How might we” - approach to asking questions and taking notes during interviews, reframe problems as opportunities

After experts leave, sort notes into categories

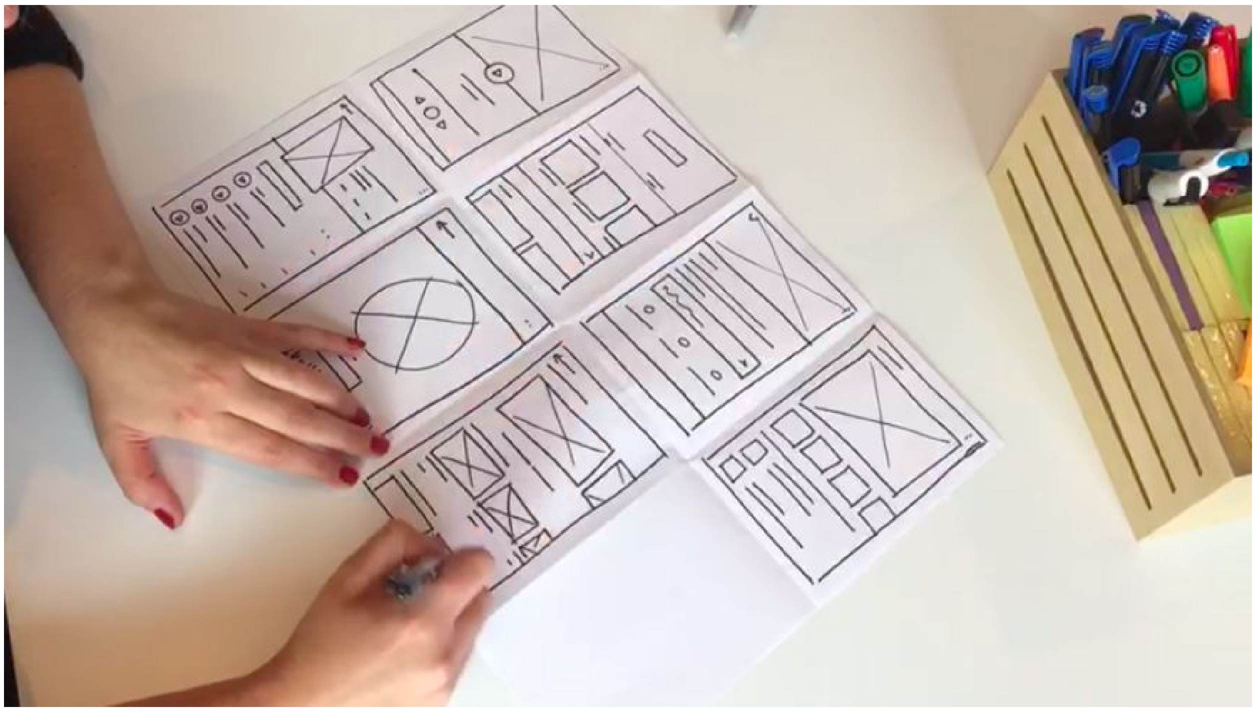
Pick target to focus on during rest of design sprint

Tuesday

Review existing ideas (e.g., from other products) to remix and improve

Team members separately sketch ideas, either competing alternatives or dividing up the problem

“Crazy 8s” - set of 8 quick sketches in 8 minutes



Start recruiting customers for tests to occur on Friday

Wednesday

Go through sketches, critique each, choose best

Weave into a storyboard and fill in gaps.

Storyboard creator tool here. There are also free wireframing tools online, e.g., wireframe.cc, as well as other drawing tools that can be used for drawing wireframes and storyboards, e.g., diagrams.net (aka draw.io)

Thursday

Implement “fake it” prototype (facade, possibly just marketing materials)

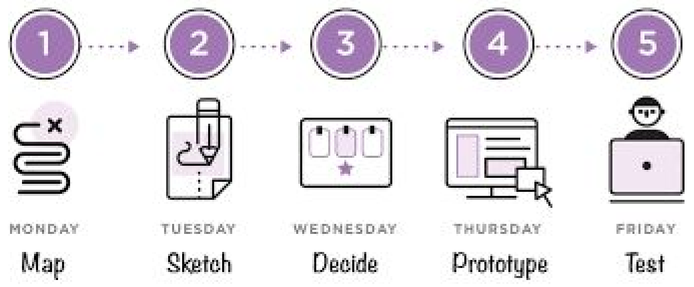
Develop interview script and do trial run (test!)

Friday

1-on-1 interviews with 5 “customers”, show prototype, open-ended questions

Rest of team watches video feed and takes notes

Review “customer” patterns - positive, negative, neutral - and decide how to follow up



Assignment T6: Final Demo: https://courseworks2.columbia.edu/courses/104335/assignments/491047 due December 10

Second Assessment available 12:01am December 11, due 11:59pm December 14. The deadline for students with an “extended time” arrangement is 11:59pm December 16.

Extra Credit: Optional Demo Video https://courseworks2.columbia.edu/courses/104335/assignments/543277 due December 20