CSE 403

Project Announcements
10/03/12

Announcements

- You teams should have selected a project manager by now; inform instructor (and TAs)
- Your TA will coordinate a weekly meeting with your team/project manager
- Project manager needs to turn in a weekly status report by 11:59PM on Sunday
- You should have a good idea how you are going to organize yourselves
- Requirements due at 11:59PM on 10/8 (Monday)

“Divide and conquer” is your friend

- Your organization structure should help you divide up the responsibilities
- Does everyone need to “deep dive” into the requirements?
- Same for architecture, runtimes and tools investigations, data(base) design
- Understand the communication paths
- Playing, experimenting, prototyping is important at this stage
- Need to get started on requirements even though we haven’t lectured on it yet

Project proposal grading

- What is the vision of your project? (possible: 20)
- What problem are you trying to solve? (10)
- What existing solutions don't fully address this problem? (5)
- What is the proposed architecture of your product? (5)
- What is the proposed tool chain? (5)
- What is the minimum viable product (full credit if a minimum viable product is stated+stretch goals) (10)
- What are the risks? (5)
- What are you delivering in this project? (5)
- Made presentation in 3 minutes (10)
- Moving forward to be a project (5)
- Total points cast for project/Total points of highest project (20)

Stats

- High: 90
- Low: 60
- Mean: 82
- Median: 80
- Standard Deviation: 8.6
- 60, 70, 70, 75, 76, 78, 81, 82, 82, 85, 85, 88, 89, 89, 90

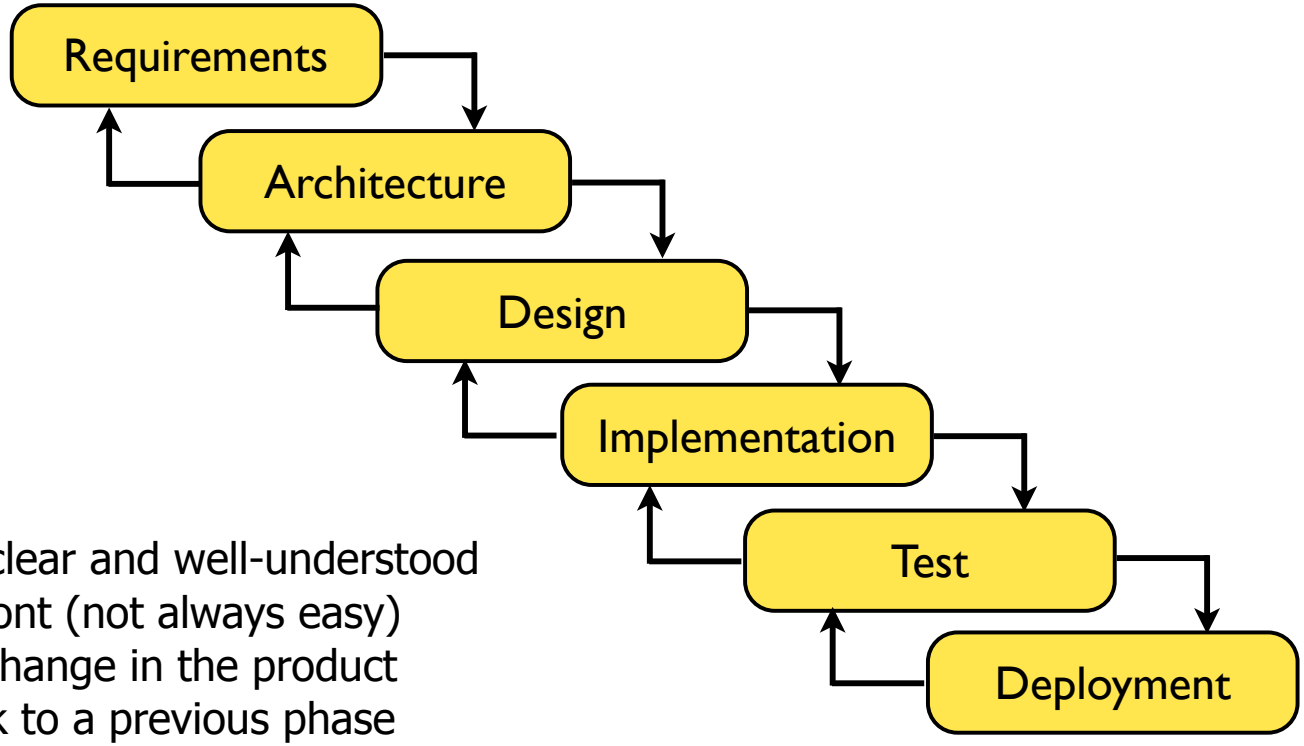
CSE 403

Software Lifecycle (continued)
Fall 2012

Lifecycle Tasks

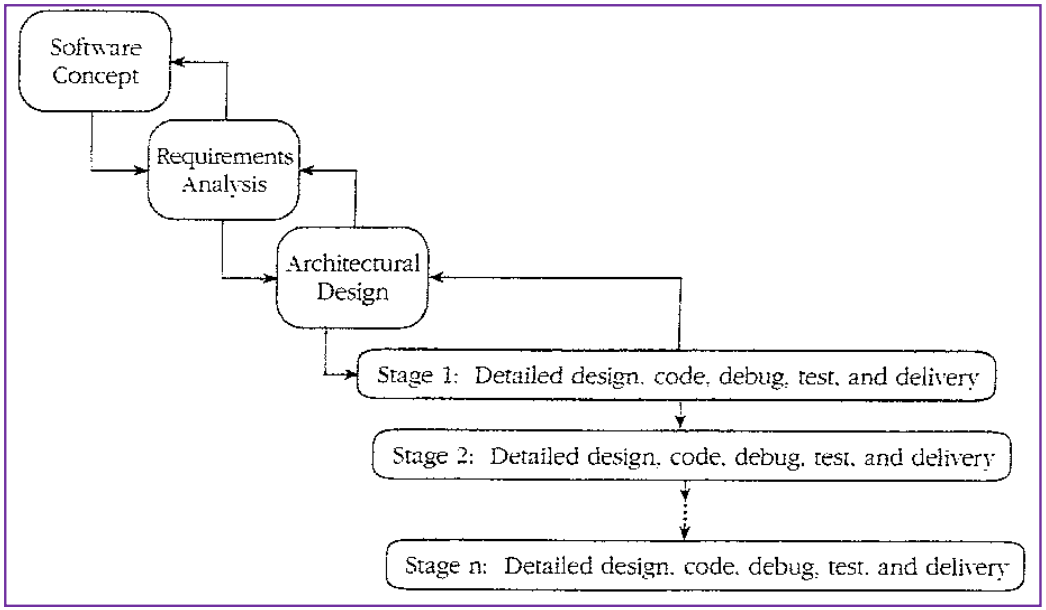
Requirements	What
Architecture	Big Picture
Design	Details (but...)
Implementation	Code
Test	Make sure it works
Deployment	What the customer sees
Support/Maintenance	Keeping it working
Compatibility	Old versions work with new

Waterfall Model



- assumes requirements will be clear and well-understood
- requires a lot of planning up front (not always easy)
- rigid, linear; not adaptable to change in the product
- costly to "swim upstream" back to a previous phase
- hard to “pipeline”
- nothing to show until almost done ("we're 90% done, I swear!")
- out of vogue, in 2012

Staged delivery model



- Waterfall-like beginnings
- Then, short release cycles: plan, design, execute, test, release, with delivery possible at the end of any cycle

Ad hoc development

Great for early stage development for a small team.

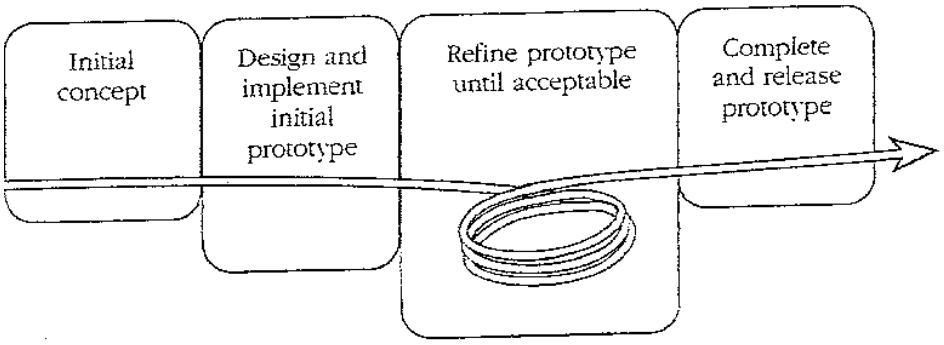
- Get early feedback quickly
- Efficiently deploy a lot
- Low overhead

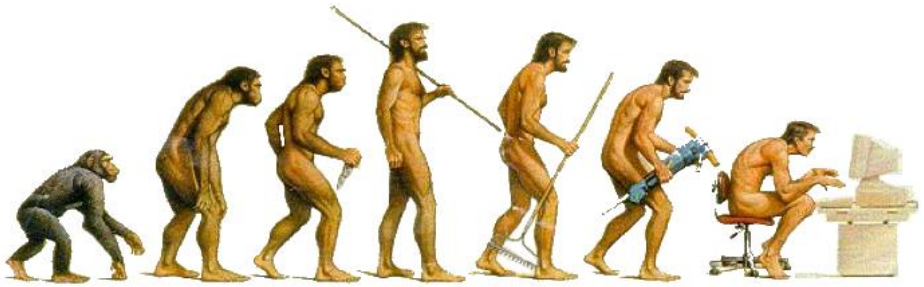
...but not without down sides

- Are you building the right thing?
- Will it scale (across multiple dimensions)?
- Susceptible to disasters
- Progress grinds to a halt...
- Not "engineering?"

Evolutionary prototyping model

- Develop a skeleton system and evolve it for delivery
- Staged delivery: requirements are known ahead of time
- Evolutionary: discovered by customer feedback on each release





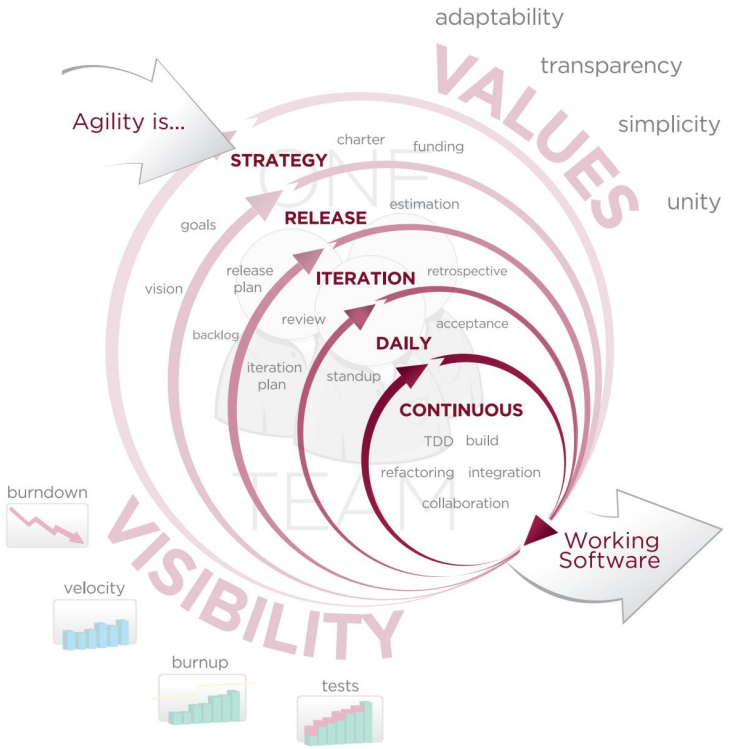
Agile Development

agile software development: An adaptive, iterative process where teams self-organize and build features dynamically.

- Extreme Programming
- Scrum

values:

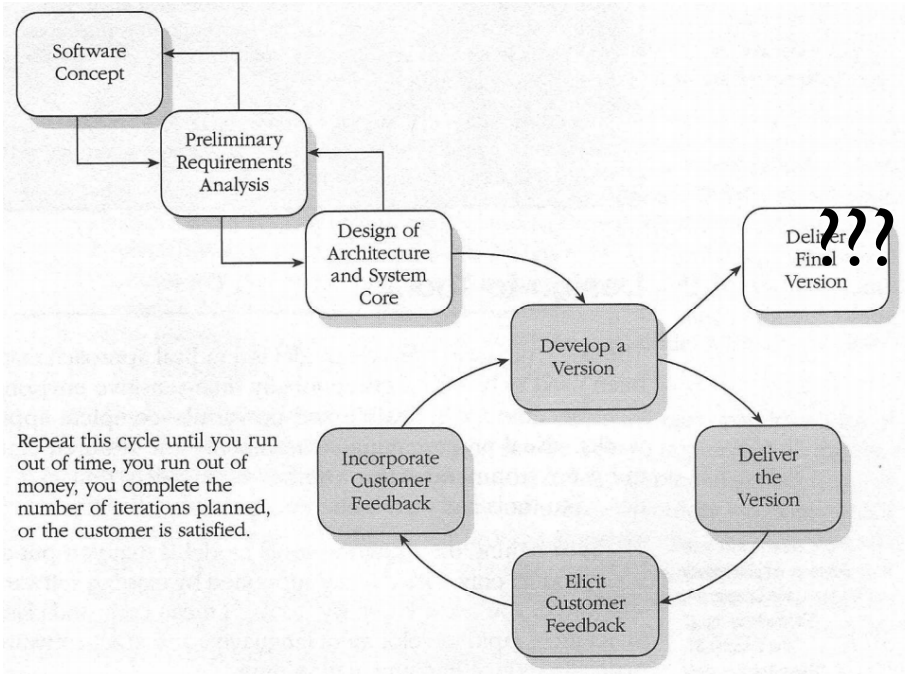
- **Individuals and interactions** over processes and tools
- **Working software** over documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan



Agile (a practical view point)

- Release cycles are typically shorter
- Small features with small requirements
- Decoupled features
- “Bottom up” (maybe)
- Better regression/unit tests (more on this later)
- Frequent but short meetings
- More “developer friendly” and in vogue
- Faster feedback

“Evolutionary” delivery



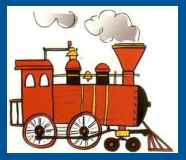
Similar to staged delivery but requirements, design (and architecture) pushed downstream

When do you release software?

- Feature-based processes
- Train-based processes
- Continuous processes

“Feature Based” Processes

- Decide what you want to release
- Figure out how long it takes
- Determine a release date
- Release “cycles” have variable length
- Execute
- Process is brittle because of errors in estimation of how long it takes
- (False) belief that you can “slip in” one more feature, pull in the date by cutting a feature, add a feature by just adding a little more time



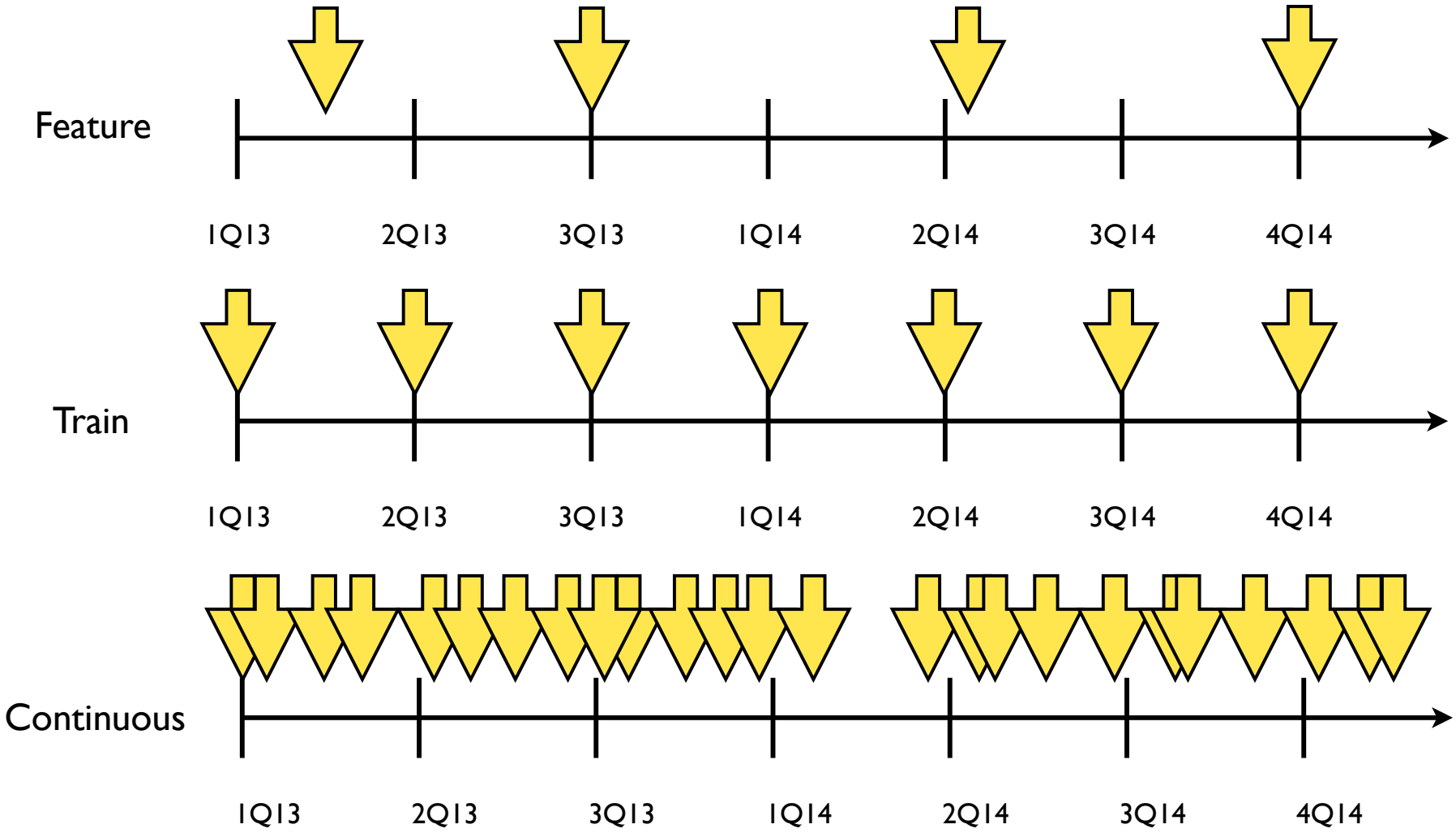
"Train-based" processes

- Release software on a fixed schedule (monthly, quarterly, yearly)
- Load or unload features onto the train when they are ready
- Predictability of "when" -- less predictability of "what"
- Harder to be agile?
- Makes engineers lazy?
- Analogy: Trains comes by every day on schedule; could be crowded, could be empty

Continuous Release Processes

- Can release a new feature at any time
- Seems flexible and efficient, possibly ad hoc
- But all steps are executed (or worse, ignored)
- Queue of (small) features
- Delivering big features harder? (Stateful/db changes trickier)
- How do you regression test?
- Developer friendly(?)
- Works for small teams, new products, with evidence that it scales to big companies too

Feature v. Train v. Continuous



Summary: Lifecycles vs. release cycles

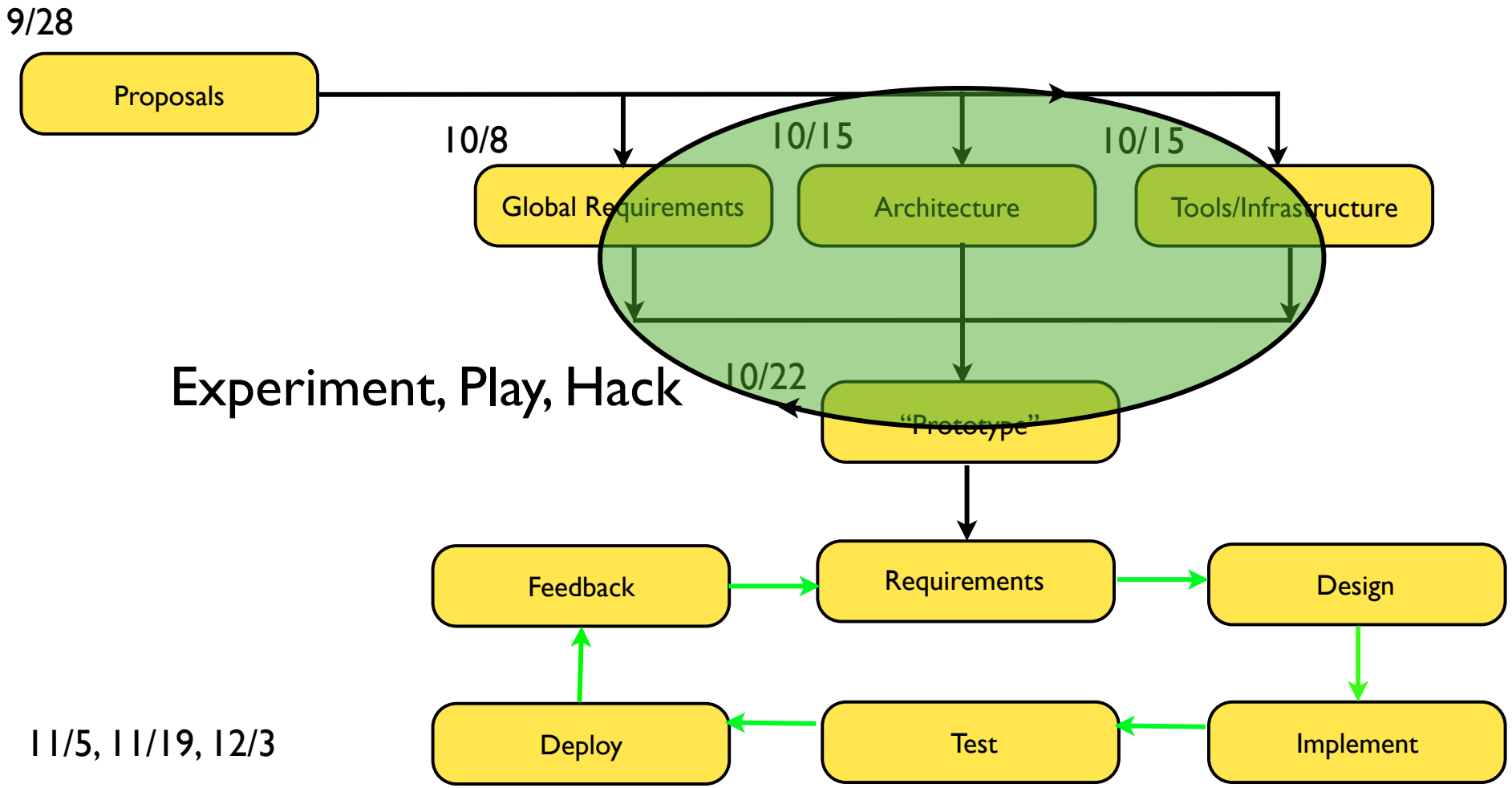
Lifecycle Methodologies

- Waterfall
- Staged delivery
- Evolutionary prototyping
- "Evolutionary" delivery
- Ad hoc

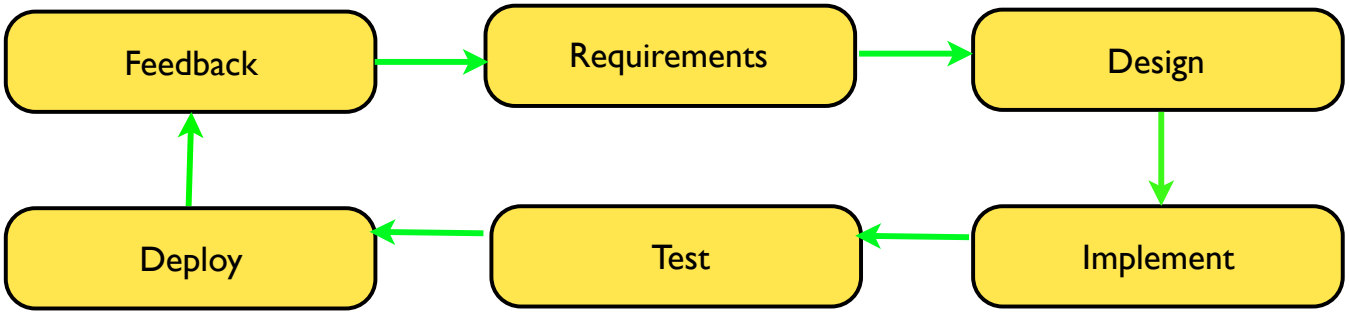
Release Cycles

- Feature based
- Train-based
- Continuous

CSE 403 Projects



What happens in the two week cycle?



- Review feedback (Day 1)
- Finalize/Review Requirements (Day 1-2)
- Decompose features, Design, Implement, Unit Test (Day 3-10)
- Feature complete/Code Free (Day 11)
- Fix bugs (Day 12-13)
- Deploy (Day 14)

What could go wrong?

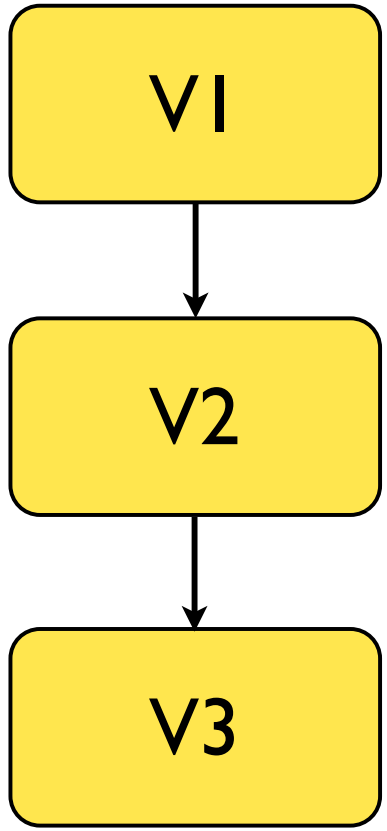
- Underestimated time to implement a feature
- Underlying infrastructure not stable
- Deployment environment not ready
- Where's the data?/Where are the users?
- Not enough time to write/review requirements
- Code doesn't work
- No time to review what went wrong, so next cycle is jeopardized

Remedies

- Drop features (until the next release or forever?)
- Scale way back on the next release to get infrastructure worked out
- “Pipeline” overlapping releases
- Wrong process to begin with?

Serialized releases (non-overlapping)

Feedback, Requirements, Design, Implementation, Test, Deploy



11/05

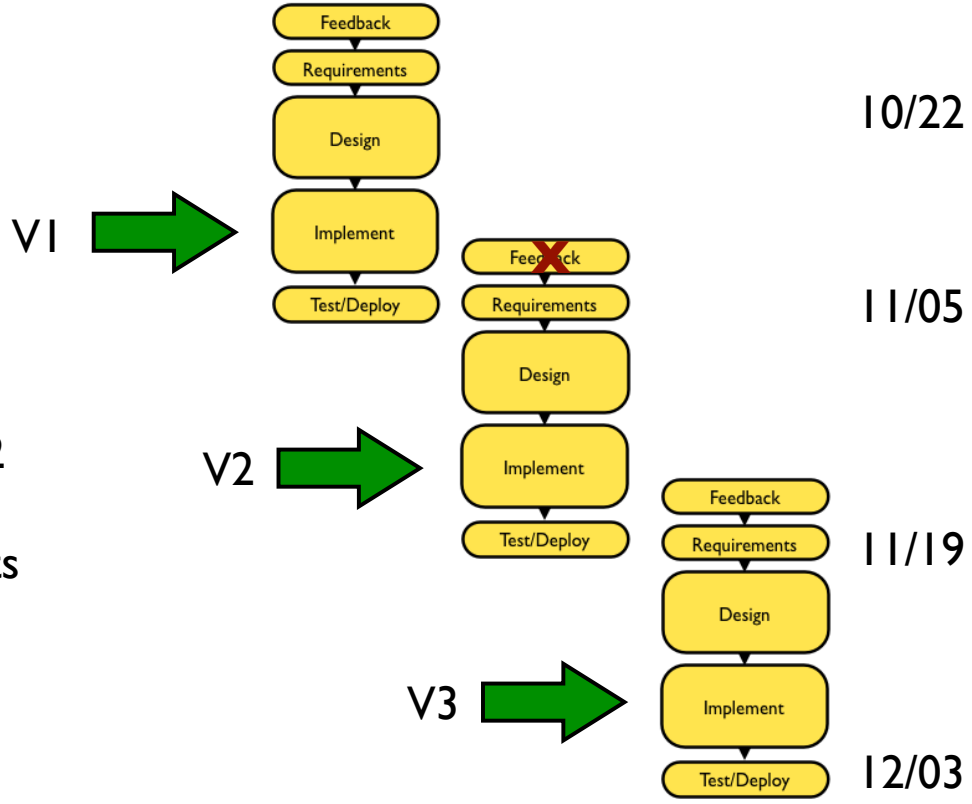
Feedback, Requirements, Design, Implementation, Test, Deploy

11/19

Feedback, Requirements, Design, Implementation, Test, Deploy

12/03

Pipelining releases



- Release cycle is 3 weeks w/ ~2 weeks of development
- Overlap feedback/requirements of next release with implementation of previous release
- 3 releases in 6 weeks

Obviously more complicated

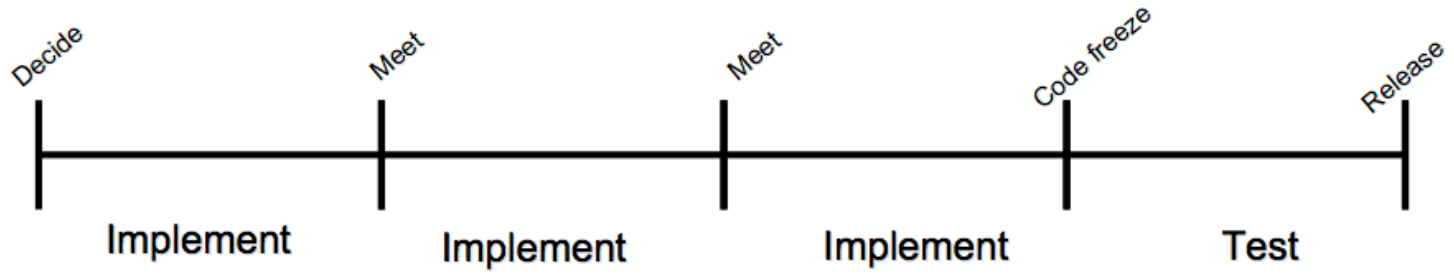
- Parallelism through pipelining
- Working on two releases at once
- But carefully structured it so that only implementing one release at a time
- Divide and conquer necessary: requirements separated from implementation
- Aside: What happens if you have a big feature that takes longer than 1 cycle to implement?

Where else have we seen pipelining/ divide and conquer in computer science?

- Basic computer architecture
- fetch, decode, load, execute, and store cycle
- Instruction execution broken down into pieces
- Can we pipeline and execute in parallel?
- Are there cross cycle dependencies?

Related example: A monthly release cycle

- Start of month: decide/prioritize what you are going to build and release for the month
- Mondays: Decide what you'll build this week
- Code freeze 7 days before the end of the month – test, test, test!
- End of month: Release!



Take aways

- Many different kinds of software life cycles
- “Agile” with short release cycles are in vogue (at least in consumer web dev)
- Process is a necessary overhead so you can move forward as you grow
- Recommend your projects take an agile approach, with short release cycles -- hybrid feature/train releases (continuous if you are brave)
- Pipelining -- will it be necessary?

Warning: We've arbitrarily added more process in CSE 403