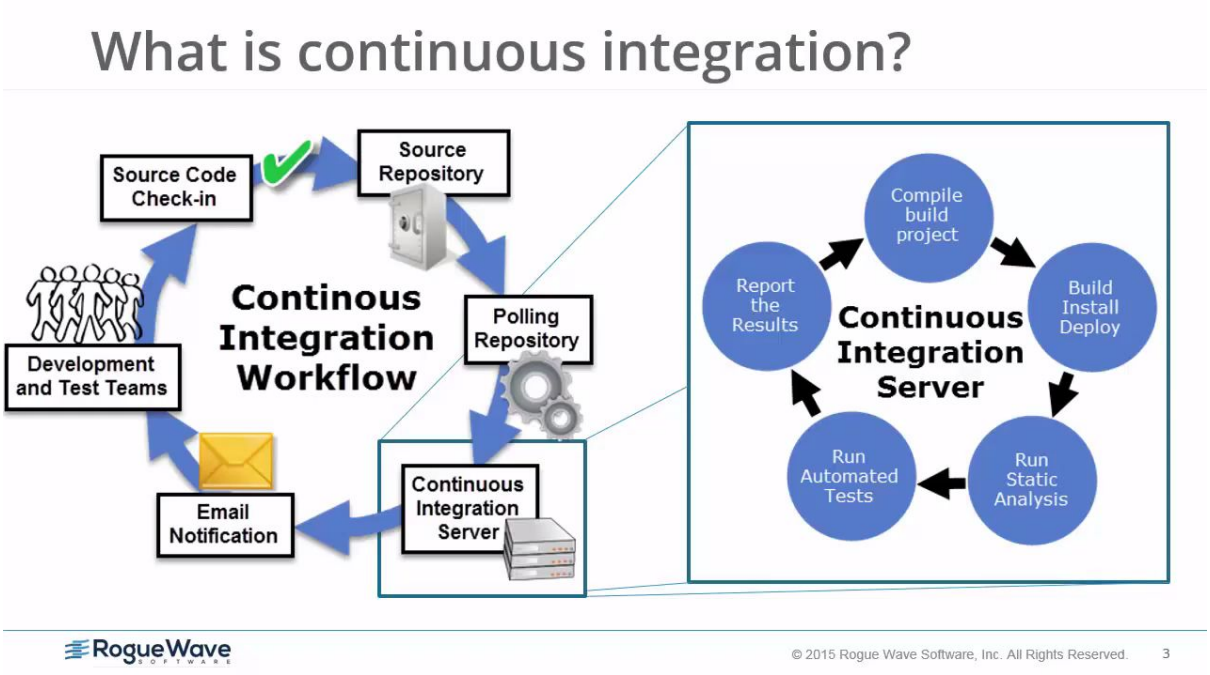
Lecture Notes
September 19, 2017

First team assignment (team formation) due tonight

Who still needs a team?

Second team assignment (project proposal) due next Tuesday September 26

Continuous Integration and Continuous Deployment



Integrates version control with “build” (may also integrate with IDE)

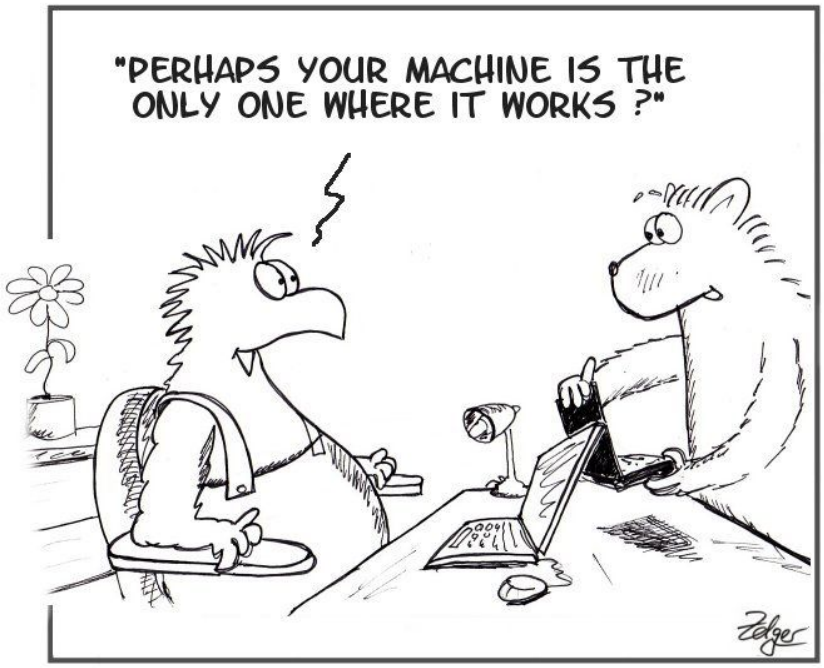
Does a full build every time anything is checked into the version control system (or at pre-designated times, such as 12midnight every day)

Pre-commit does a local build when the commit command is run, and doesn’t actually do the commit if the build doesn’t pass

From sample project:
https://github.com/COMSW4156Fall2017/coms4156_jumpstart/blob/master/bin/git-hooks/pre-commit

Post-commit does a remote build on the repository server (or some other designated server) after the commit finishes

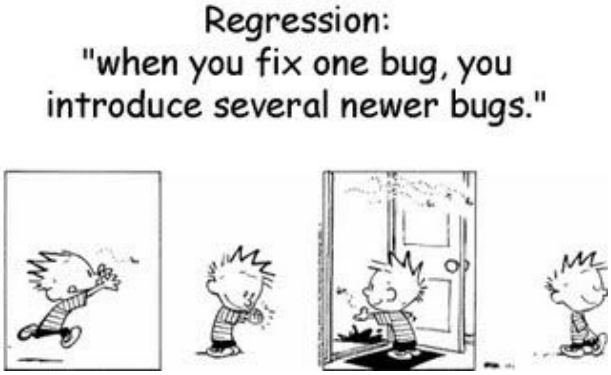
“Who broke the build?” - If the post-commit doesn’t “pass”, in most cases someone didn’t do a pre-commit build OR there is some mismatch between the environments for two or more developers



It works on my machine

For builds before/after every commit, typically only a subset of the full regression test suite is run because many full tests suites take hours to run - thus nightly builds

What is a “regression”? This is when fixing a bug or adding a new feature causes previously working code to break (or regress), sometimes code that seems completely unrelated



To detect regression, need to re-run *all* test cases that previously passed (and any tests that previously failed, in case they now fail in different ways or unexpectedly pass)

Continuous Deployment adds automatic deployment to continuous integration

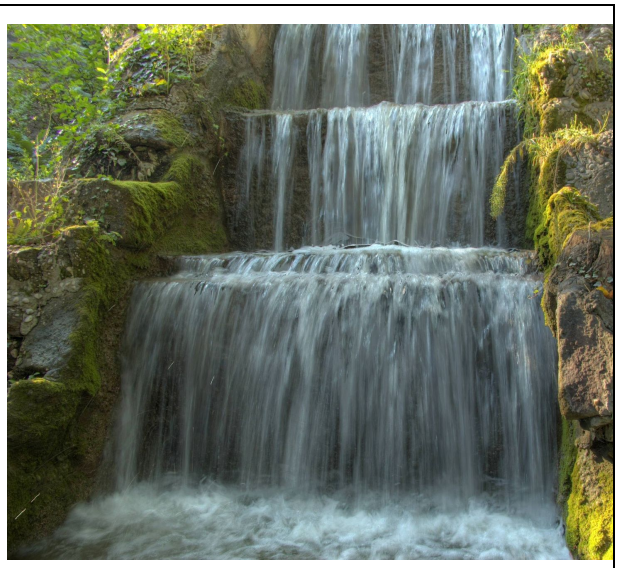
Installs on the production server(s) and/or produces a patch to push to customers

Examples: Jenkins, Travis CI

Plan-driven Processes

The term “agile” is used by many organizations to mean many different things, the main commonality is “not waterfall”

What is waterfall?



Waterfall is a *plan-driven* step by step sequential process that is simple to explain to beginning students, to high-level (non-technical) management and to government procurement agents

-> Review and signoff at each step

Only one chance to get it right: When problems are found during one step, they may be due to a flaw or omission at a previous step, but backtracking can cause > 100% schedule overrun

What preceded waterfall? Chaos - little or no structure, discipline, planning

- Didn’t scale to large projects with large numbers of developers
- Little predictability over costs and schedules

Waterfall was huge improvement!

- “Road map” to coordinate multiple developers
- Some control over scope of projects to improve predictability of costs and schedule

Waterfall, or some variant of waterfall, is necessary for some software projects, e.g., safety-critical and cyber-physical systems, with requirements fixed in advance and each step must be verified correct and complete before beginning the next step

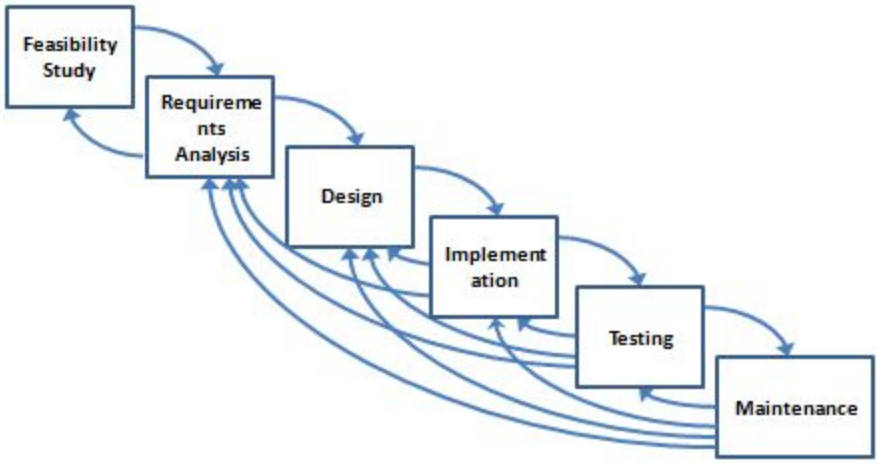
- Huge undertakings that have to be completed in all or nothing fashion, every little detail must be designed up front
- Embedded systems that cannot easily be changed after deployment
- Examples: software controlling airplane flight, automobile engine, nuclear power plant, medical device implanted into human body

But the ultimate user of the software didn't normally even see the software until all of the development and testing was complete and by that point in time it was very difficult, if not impossible, to go back and make any significant changes

- The emphasis on control of scope made the process very inflexible to any changes that might be needed to meet user needs and business goals in an uncertain environment

Basic waterfall extended by Win Royce and others since ~1970

- Iteration, rapid prototyping ("do it twice"), involve customer
- But still plan-driven and document-intensive, cannot exit a phase (the first time) until the documentation required for that phase has been completed, reviewed, and approved



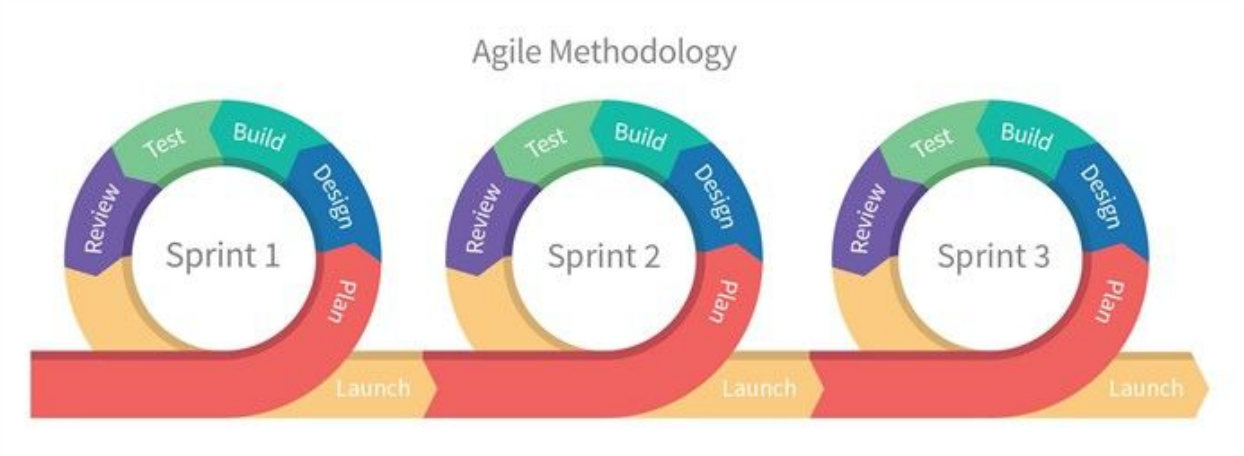
Contrast to projects that can or should evolve gradually, to allow for mistakes and mid-course corrections, relatively easy to re-deploy - pretty much any web application

Adaptive Processes

Transition to *adaptive* rather than plan-driven has been relatively slow, invention of term "agile" ~2001 combined with internet access to open-source tool chains may have sped things up

There are many different special cases of agile, such as scrum, kanban, extreme programming - we will consider agile more generally, any process that is adaptive rather than plan-driven

> But what you're teaching in class does not match what I did on my summer internship or on a prior full-time job! - Right, every work environment has their own peculiarities



Agile concepts: iteration, involve customer (and other stakeholders), change, iterative and incremental (do it twice -> do part of it many times), delivery more important that documentation

Stakeholders = customers, end-users inside the development organization, end-users outside the development organization - may be inside or outside customer organization, operators and administrators, technical support, the developers themselves, government regulators, society

- Ideally all stakeholders would be involved throughout the software process

What does every (paying) customer want to know?

- How long will it take?
- How much will it cost?
- Will it do what I need it to do?

Unhappy customers (and other stakeholders) are “bad for business”

- Ask what customers want
- Show customer what you think they asked for - and possibly find out that you’re wrong
- Rinse repeat

In the waterfall model, or any other plan-driven model without frequent customer involvement, developers may find out they’re wrong far too late

- This is one reason why agile should not be used for safety critical/cyber physical systems - developers cannot afford to be wrong, customers cannot change their minds

Change - customers can change their minds at any time, before or after delivery

- ➢ Customers may not even know what they want until they see it (or see something that is not it)

Why not do a full requirements analysis up front, as in plan-driven/waterfall process?

- ➢ Customers do not always know what they want,
- ➢ or what they think they want is not what they need

Incremental - pick one part of an application, e.g., prioritized feature set (considering dependencies), and develop it first, then the next part, then the next …

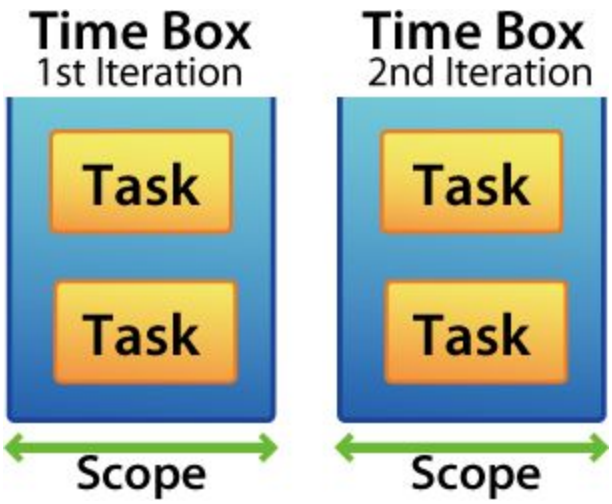
Iterative / iteration - Short project, e.g., one week, two weeks, one month, to develop and deliver an increment

- Earliest iterations might just demo, not deliver, until minimum viable product (MVP)
- Often called “sprint”
- Includes full range of process activities - planning, requirements, design, development (coding and unit testing), system testing, release/deployment, tracking/monitoring

Emphasis on delivery - Good developers develop, great developers ship

- This is another reason why agile cannot be used for safety critical/cyber physical systems - incremental delivery infeasible

Time box vs. scope box



You can add more features (scope) to an iteration, but then the iteration will take longer
You can reduce the time for an iteration, but then fewer features will be completed

You may find that the originally planned features for an iteration take longer than scheduled, so you need to reduce scope or increase time

You cannot both increase scope and decrease (or hold constant) time

Choosing time boxing over scope boxing controls costs, you know how much your iteration will cost (e.g., what you pay your developers)

And you retain some predictability, you know you will complete *something* each iteration

One of the most common causes of job dissatisfaction (and job loss) among software engineers is that management and customers often don't understand this, so insist that engineers complete more work in less time

One activity often associated with adaptive/agile processes is daily standup meetings

- Why standup? Short, around ten minutes
- Why daily? Update task board and burndown chart, detect and remove blockages

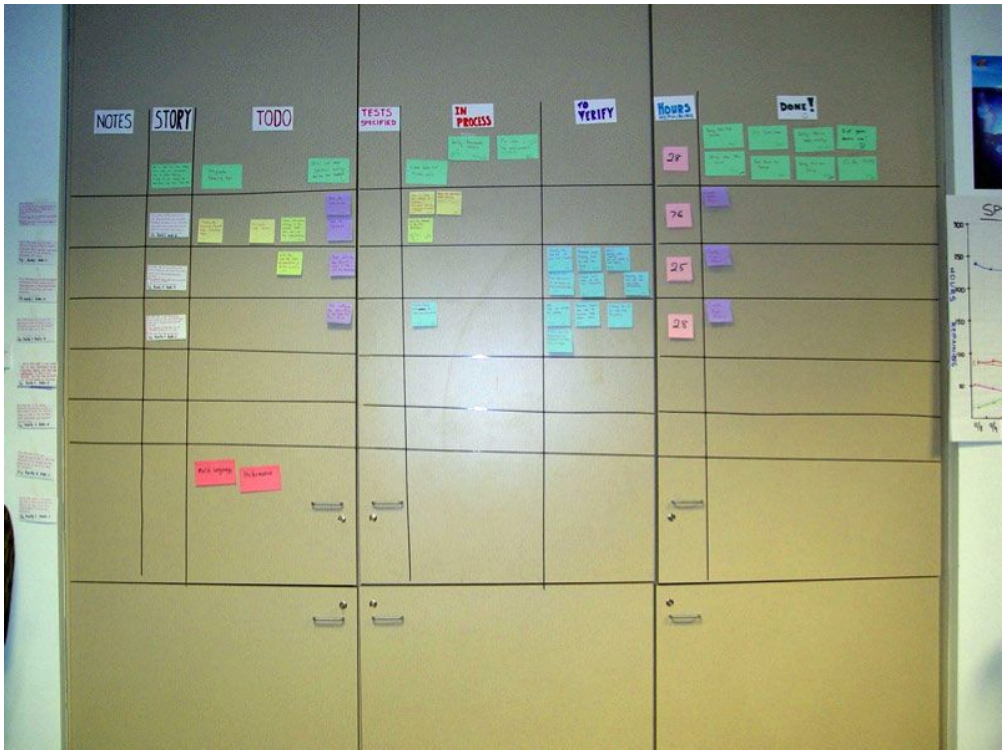
Task Boards and Burndown Charts

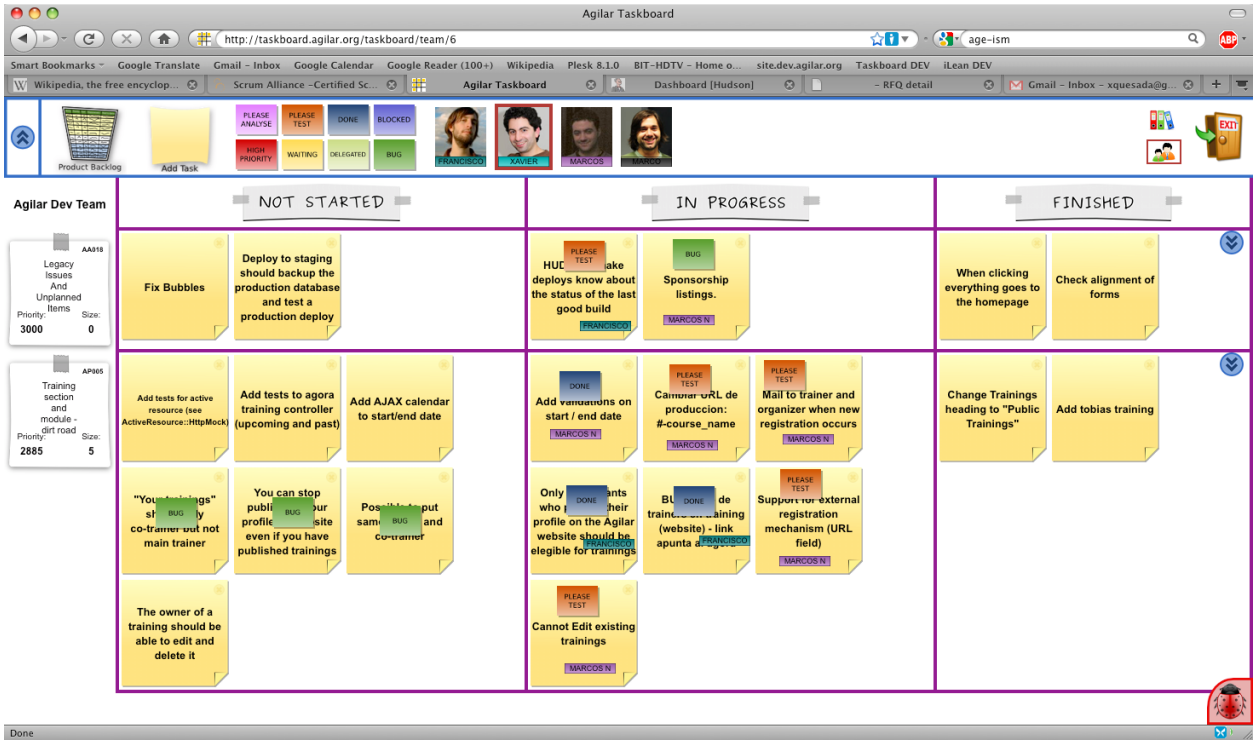
Two artifacts often associated with agile processes are task boards and burndown charts, used for team communication as part of incremental/iterative planning

Use whatever you like… or not: for your team project you will need to use github (or equivalent) “issues” and, possibly, “wiki” but you do not need to manage task boards or burndown charts

- ★ However, there are very likely to be task board and/or burn chart questions on exams

Task boards can be physical or virtual





Github provides a very simple form of virtual task board, called “Projects”

- Project corresponds more to a single sprint than a multi-sprint project
- No way to assign a task to a specific team member, unstructured text
- But conveniently integrated with github “Issues”, which can be assigned

[Sample]

Trello has somewhat more elaborate task boards, Slack and Google Hangouts also useful for team communication/collaboration

Burndown charts also physical or virtual