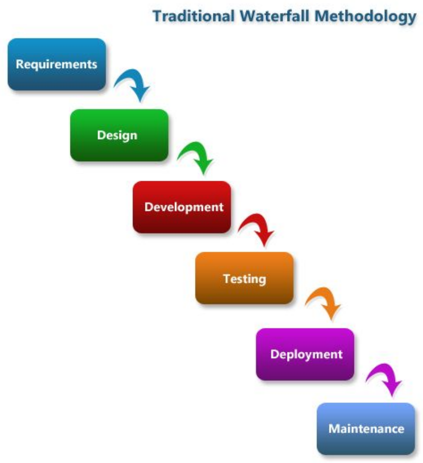
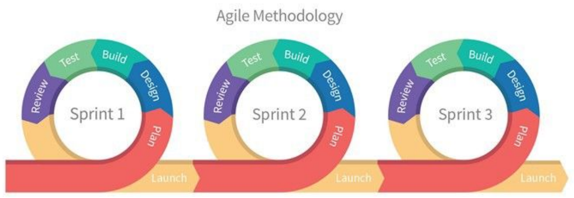
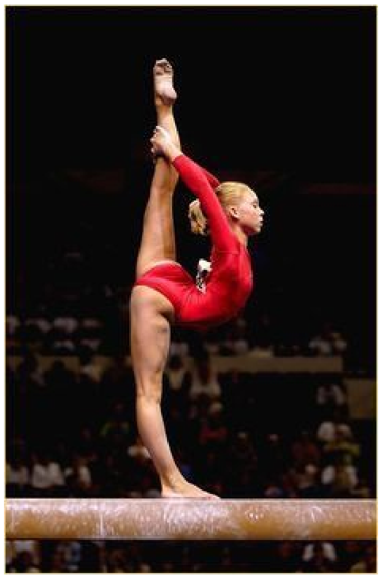
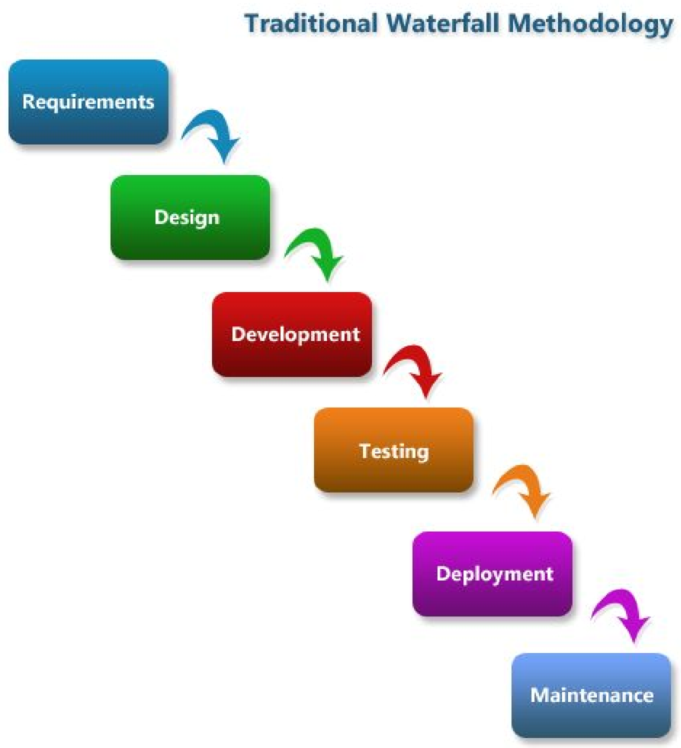
Lecture Notes
September 6, 2018

Software process: agile vs. waterfall

“Waterfall” and “agile” have become catch-all terms:
waterfall means not agile and agile means not waterfall

	 Traditional Waterfall Methodology Requirements Design Development Testing Deployment Maintenance
 Agile Methodology Sprint 1 Sprint 2 Sprint 3 Review Test Build Design Plan Launch	

For this course, waterfall = *phase-driven* process and agile = *adaptive/iterative* process



Waterfall is step by step sequential software development process that is simple to explain to beginning students, high-level (non-technical) management, and government procurement agents who review and signoff at each step

Do the entire first step: requirements analysts work with the customer to specify all requirements. Requirements analysts validate (review) the requirements specification with the customer to make sure this is what the customer wants. The requirements are not supposed to change later. But if they do change significantly, basically need to start over (with some reuse of previous work).

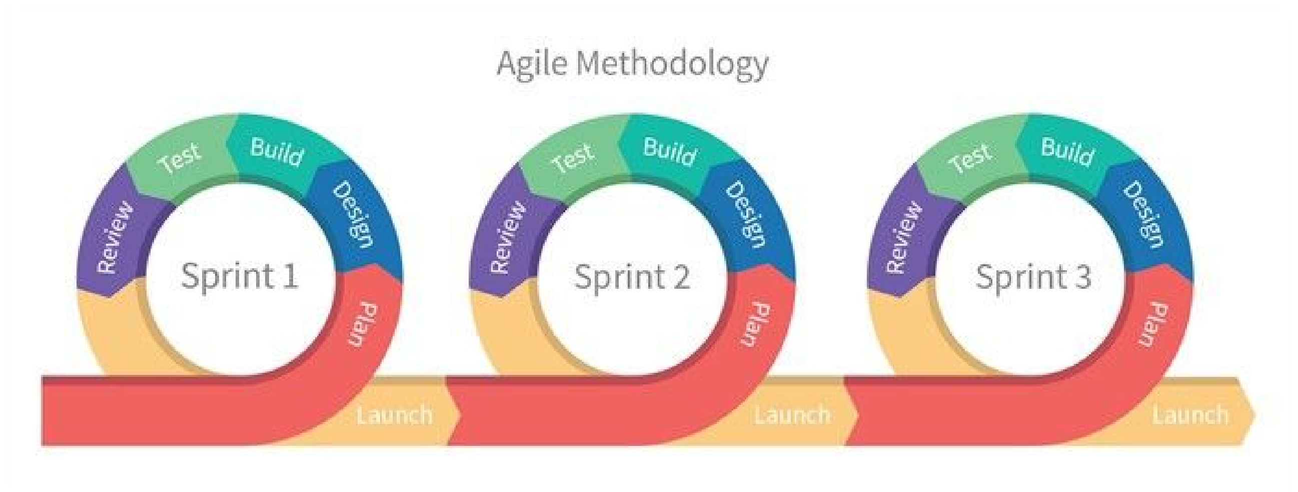
Do the entire second step: designers develop entire design, planning all the modules, data structures, subroutines, etc. Designers validate (review) the design with requirements analysts to make sure it fulfills the requirements. Afterwards, the design is not supposed to change.

Do the entire third step: programmers write all the code, and make sure it compiles and builds. Programmers validate (review) the code with the designers to make sure it fulfills the design. Afterwards, the structure of the code (modules, classes, data structures, methods) is not supposed to change - assumes bug fixes will be small and localized.

Testers plan the test suite, and validate (review) the test suite with requirements analysts, designers and programmers to make sure it covers 1. the original requirements, 2. the design, 3. the code. Portions of test planning may run in parallel with these stages.

Do the entire fourth step: testers repeatedly run the full test suite until all tests pass. When bugs are found, testing cycles between programmers patching the code and testers rerunning tests. The patches are supposed to be modest bug fixes, not major rewrites, with <u>no</u> changes to design or requirements.

Finally deploy to customer. All changes after that considered “maintenance”, which is not really addressed by waterfall - except by repeating the entire waterfall process for version two.



In contrast, agile considers a small piece of the software project at a time in a "sprint" (or iteration), typically two weeks, with deployment to customer after a short series of sprints or even after every sprint.

An agile sprint goes through all the waterfall-like stages, but for only that small piece of the project, and trying to parallelize as much effort as possible - the roles of requirements analysts, designers, programmers, testers are not necessarily separated.

Since so little time and effort has been invested in an individual sprint, it is feasible to "change everything" during the following sprint.

Some sprints, particularly the first sprint of a new product, or a new release of an existing product, may be preceded by one-week "design sprint"

Waterfall is not “bad”, agile is not “good”

What preceded waterfall? Chaos (1940s and 1950s)

- Little or no structure, discipline, planning
- Little predictability over schedules and costs
- Didn’t scale to large projects with many developers

Waterfall was huge improvement! (1960s - 1980s)

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

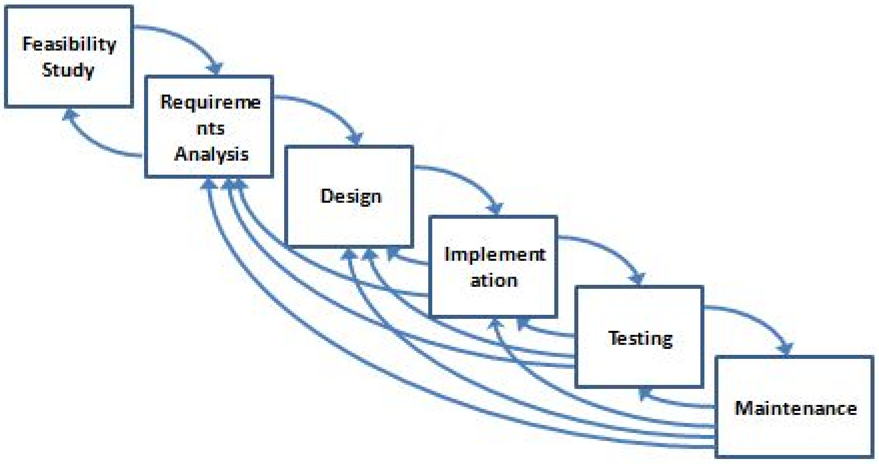
Waterfall, or some variant of waterfall, is still necessary for certain kinds of software projects, e.g., safety-critical / cyber-physical systems

- Cannot easily be changed after deployment so requirements must be fixed in advance
- Each step must be carefully verified correct and complete before beginning the next step
- Examples: software controlling airplane flight, automobile engine, implanted medical device

But waterfall is unnecessarily restrictive (and expensive!) for most business and consumer software, even more so since ~2000 when it became possible to update software via downloads

Basic waterfall extended by Win Royce and others since ~1970

- Iterative/incremental, rapid prototyping (“do it twice”), involve customer
- But still phase-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, continuing introduction of new features, relatively easy to re-deploy - most business and consumer applications

So today we have agile (since 1990s, although the term "agile" was coined in 2001)

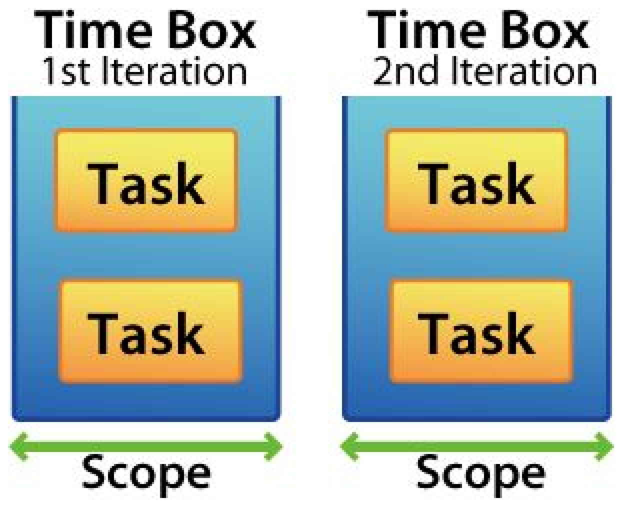
- Many organizations still use waterfall or a variant of waterfall
- Many organizations are "transitioning" to agile
- Many organizations claim to follow an agile software development process
- If you eventually work for two or more different "agile" organizations, you will learn there is not a single well-defined agile process used by everyone

Some well-known agile variants:

- Scrum
- Kanban
- Extreme Programming

This class will focus on major concepts that cut across most variants of agile

One major agile concept is Time box vs. Scope box



Fixed-length sprint means the scope is limited by what can be accomplished during that time period

If more features (scope) are added to a iteration, then the length of the iteration necessarily increases

If less time is allocated to an iteration, then fewer features (scope) can be completed

If it turns out that the originally planned features will take longer than scheduled, which happens often on new projects, then the scope must be reduced with some features pushed off to later sprint (or dropped)

Choosing time boxing over scope boxing controls costs, because management knows how much the iteration will cost ~= how much the staff is paid for the time-period

One limitation of waterfall is it is essentially scope boxed: the requirements determined up front define the full set of features (scope), but cannot address how long it will take to complete that scope - So costs cannot be predicted!

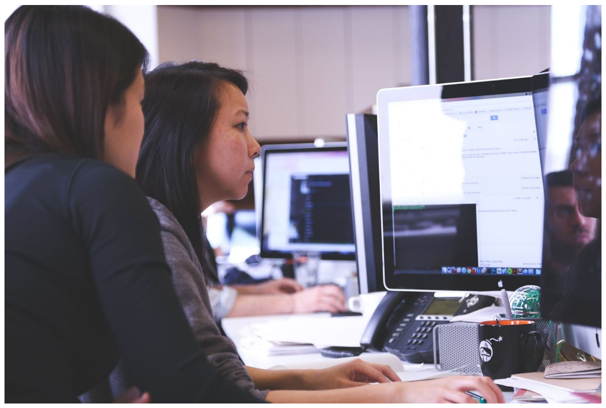
Another major concept often associated with agile is daily standup meetings

- Why standup? Keeps short, 5-10 minutes
- Why daily? Detect and remove blockages quickly
- Some software teams instead have meetings 2-3 times per week, sitting down for 10-20 minutes, possibly including remote participants

"Pair programming" is also a common agile practice

Many software organizations have adopted pairing for some or all software engineering activities, not just programming

Some use pair-programming interviews between a job candidate and an evaluator (current employee) even if their employees do not pair-program routinely



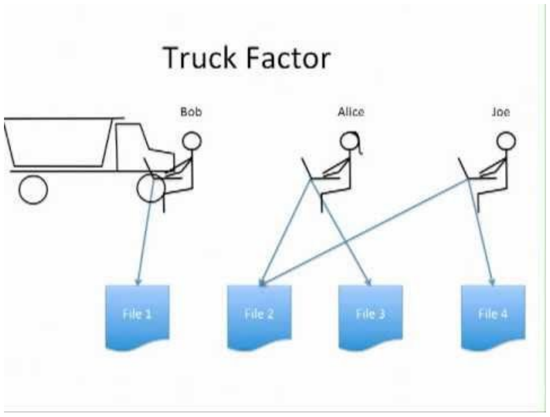
Pair programming is when two people sit side by side at same computer - or use remote desktop sharing

- Take turns "driving" (typing) vs. "navigating" (continuous review of code or whatever working on)
- Slide keyboard and mouse back and forth
- Switch roles at frequent intervals (e.g. 25 minutes)
- The pair does **not** divide up the work, they do everything **together**
- If necessary to work separately on something, e.g., when one partner is absent, they review together

Studies show higher quality, but mixed productivity - often takes more total time to do the initial work (when you consider time-spent x 2), but less time later on debugging and bug repair

Organizations that use pair programming typically switch pairs often to reduce risk

- *Collective code ownership*
- Maximize "**truck factor**" = number of engineers that would have to disappear before project would be in serious jeopardy



Other benefits of pair programming

- Shared responsibility to complete tasks on time
- Stay focused and on task -
  Shared time treated as more valuable
  Less likely to read email, surf web, etc.
  Less likely to be interrupted
- Partners expect "best practices" from each other
- Two people can solve problems that one couldn't do alone and/or produce better solutions
- Pool knowledge resources, improve skills

Read for Tuesday:

"Knightmare: A DevOps Cautionary Tale"

and

Code Smells