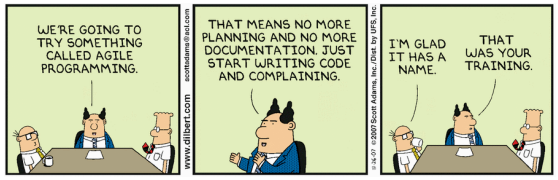
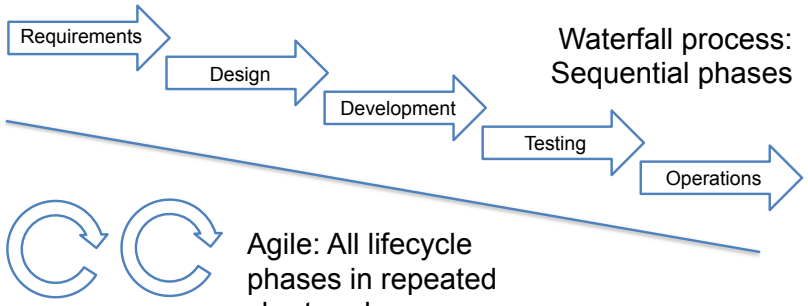
Plan & Document ⇒ Agile



Dilbert 11/26/17



Waterfall process: Sequential phases

Agile: All lifecycle phases in repeated short cycles

Agile Manifesto (2001)

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

- **Individuals and interactions** over processes and 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.

http://agilemanifesto.org

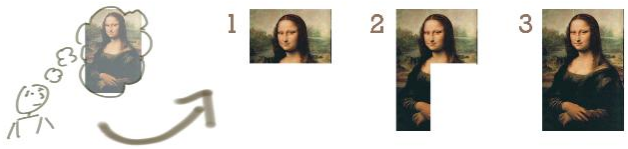
Agile vs. agility

1. Find out where you are,
2. Take a small step towards your goal,
3. Adjust your understanding based on what you learned, and
4. Repeat

When faced with two or more alternatives that deliver roughly the same value, choose the path that makes future change easier

Do you want to increment or iterate?

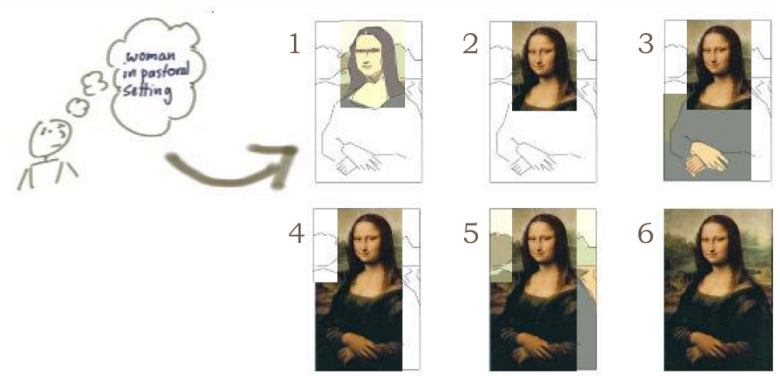
Incremental



Iterative

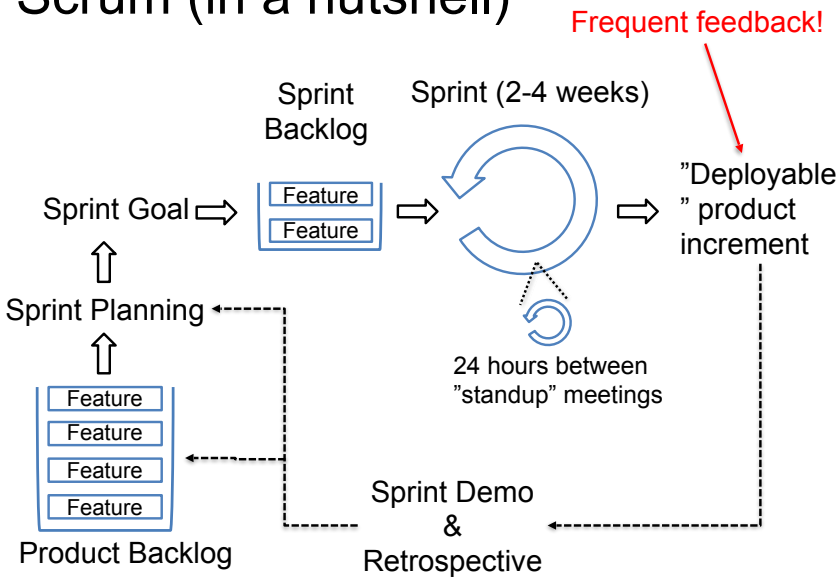
https://jpattonassociates.com/dont_know_what_i_want/

Iterative Incremental

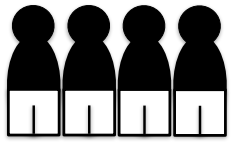


http://itsadeliverything.com/revisiting-the-iterative-incremental-mona-lisa

Scrum (in a nutshell)



Scrum team



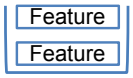
Development Team
- Self-organizing ← Critical!!
- Cross-functional
- No hierarchy of specific titles
- A single team without sub-teams
- Accountable as a group

Product Owner
- Represents the customer
- Responsible for prioritizing the product backlog

Scrum Master
- Servant-leader for team
- Facilitate SCRUM process

Scrum artifacts: Product backlog

Product Backlog

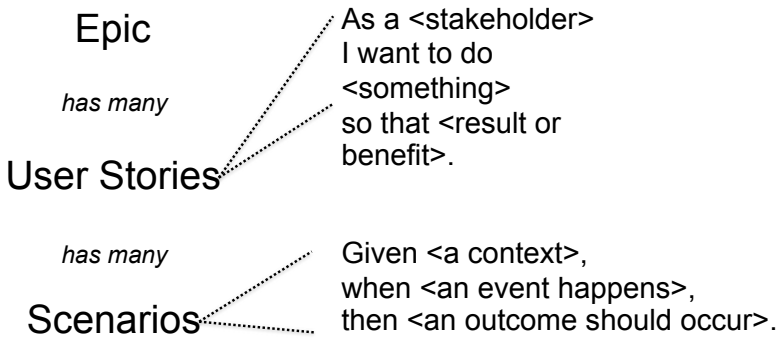


Sprint Backlog

- A *prioritized* list of user stories (and other tasks) maintained by the product owner
- Evolves as you learn more (stories are added, removed, re-prioritized)
- A subset of stories are chosen for each sprint (Spring Backlog)
- Should be readily accessible to everyone on the team (and me!)

Relevant tools: GitHub issues, Google Doc, Trello, Pivotal Tracker, …

Recall: Epics, User stories, Scenarios



Effort estimation and velocity

- Not all stories count equally, need to know how much work we are taking on
- Assign each story (and bug) points

  Recommend: 1, 2, 4, 8 (8 is rare and should be split)

  Vote independently, high/low explain their vote

  Iterate until convergence OR take high vote
- Aim for constant velocity

  velocity := points per week

For last 3 iterations, Team Blue's (#003F84) average velocity is 8, Team White's is 4. Which, if any, of the following comparisons between the Blue and White teams is valid?

A. Blue has more developers than White
B. Blue is twice as productive as White
C. Blue has completed more stories than White
D. None of the above

Student Advice: Scrum/Stand-ups

- "5-minute daily standups really helped us stay on track, and share knowledge when stuck"
- "Biggest challenge for us was team communication/coordination"
- "Have a scrum leader each time, rotate the position"
- "1 meeting per week isn't enough"

Adapted from Berkeley CS169

Pair programming

- Driver types and thinks tactically about current task, explaining thoughts while typing
- Observer reviews each line of code as typed, and acts as safety net for the driver
- Observer thinking strategically about future problems, makes suggestions to driver

Should be lots of talking and concentration
Frequently switch roles

Pair programming evaluation

- Small increase in developer time (15%)
- Decrease in defects, i.e. higher quality
- Transfers knowledge between pair
  Programming idioms, tool tricks, company processes, latest technologies, …
- Programmers often report increased job satisfaction

Williams et al. IEEE Software, 2000

Thinking about pairing: Dreyfus squared for skills

	Novice	Adv. Beginner	Competent	Proficient	Expert
Novice			✔		✘
Adv. Beginner		Crazy learning!			
Competent					
Proficient					✔
Expert					

Novice: Needs rules
Advanced Beginner: Tests the rules
Competent: Applies rules
Proficient: Falls back on rules
Expert: Transcends rules

Student Advice: Pair programming

- "Helped avoid silly mistakes that could take a long time to debug"
- "Changing partners frequently made team more cohesive"

Adapted from Berkeley CS169

Resolving conflicts (e.g. different views on the technical direction)

1. Remember there is no “winning”, most questions don’t have “right answers” just tradeoffs
2. List all items on which you agree
   Instead of starting with a list of disagreements
   Maybe you agree more than you realize
3. Articulate the other side’s argument, even though you don’t agree
   Avoids confusions about terms or assumptions (often the root cause of the conflict)
4. Constructive confrontation (Intel)
   If you have a strong opinion that a proposal is technically wrong, you are obligated to speak up and seek a conclusion
5. Disagree and commit (Intel)
   Once a decision is made, embrace it and move ahead

See also: K Matsudaira, Resolving Conflict. Don’t “win.” Resolve. ACM Queue 14(5) 2016