Risk Management and the Minimum Viable Product

“...project risk is a good thing, a likely indicator of value. Projects that have real value but little or no risk were all done ages ago.”

Peopleware: Productive Projects and Teams

“Part of the challenge of programming (and for some people, the reason why programming is fun in the first place) is looking at the building blocks provided to you and deciding how to assemble them to build something new. After all, if everything you wanted a program to do already existed ready-made, it wouldn’t be called *programming* any more. It would be called *shopping*.”

Raymond Chen

“...the real reason for risk management: It’s not to make the risks go away, but to enable sensible mitigation should they occur. And mitigation may well have to be planned and provisioned well ahead of time.”

Peopleware: Productive Projects and Teams

What risks are you worried about?

What risks are you not worried about?

Categories of risk

Technical risk: bug in dependency, bug in other parts of the system, production environment differs from development environment, NP-hard subproblem

Entrepreneurial risk: you built the wrong thing, you built the right thing but the requirements changed, customers don’t really care, complementary assets not available

Social risk: your mentor is unresponsive, your users can’t/won’t make time to meet with you

Categories of risk

Technical risk: bug in dependency, bug in other parts of the system, production environment differs from development environment, NP-hard subproblem

Entrepreneurial risk: you built the wrong thing, you built the right thing but the requirements changed, customers don’t really care, complementary assets not available

Social risk: your mentor is unresponsive, your users can’t/won’t make time to meet with you

Personal risk: you get sick, you’re too busy with other classes, you don’t know the language/library/environment as well as you thought

Software Development Strategies

Software development strategies

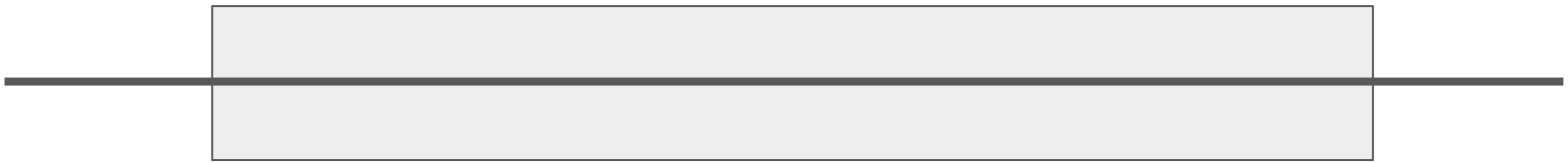
Software development strategies are strategies for managing risk.

Also called methodologies, processes, paradigms, …

Primarily describe interactions between humans (and associated artifacts), not projects or technologies.

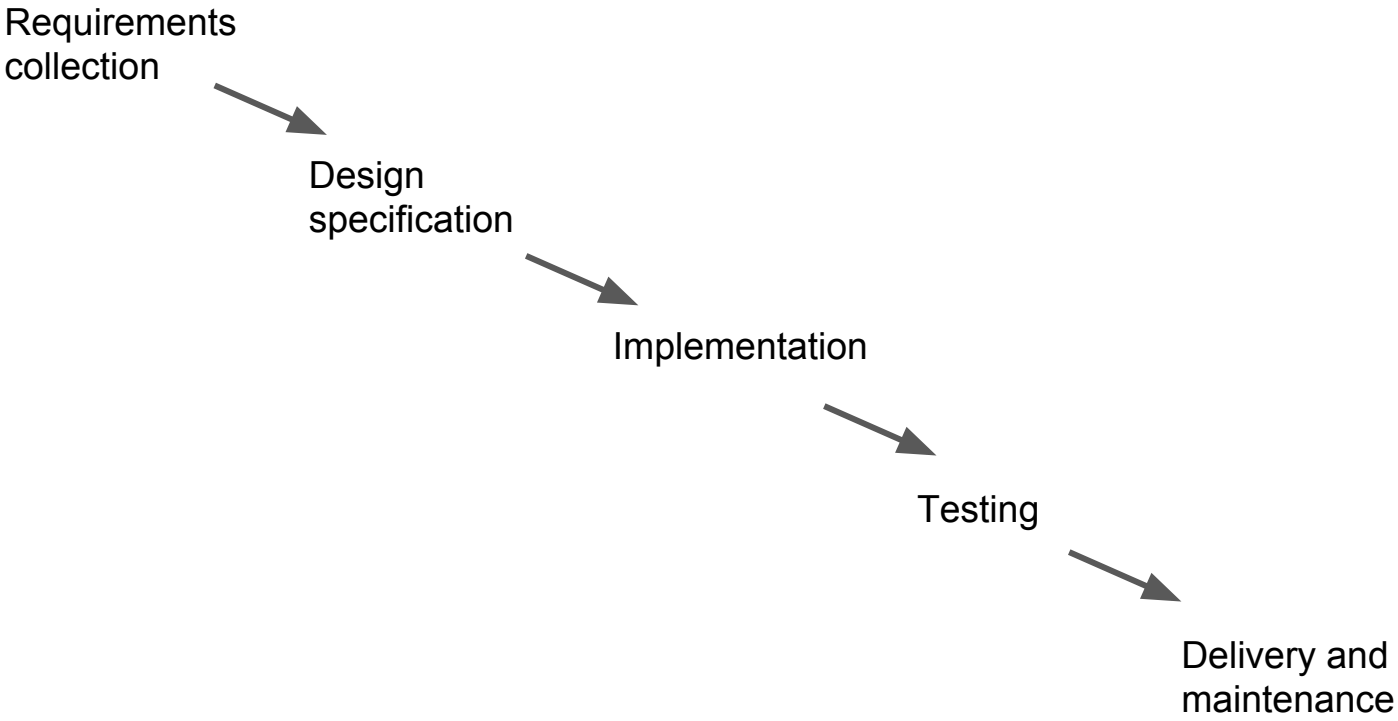
Waterfall

Agile



Both sides are strawmen. Every real project is somewhere in between.

The waterfall model



The waterfall model: pros and cons

Manage risk by getting certainty early.

Producing a design specification helps detect technical challenges earlier, when they're cheaper to mitigate or avoid.

Producing a design specification before implementation begins helps document and retain overall knowledge of the system even after people leave the team.

Dedicated testing phase stabilizes the product before release. Even if a bug isn't fixed, it's at least not a customer surprise.

Customers may not know what they want (or don't want) until they see it. If requirements change, hard to correct course.

Product becomes available all at once at the very end of the process. In the meantime, the partial product is not useful.

The Agile Manifesto

“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

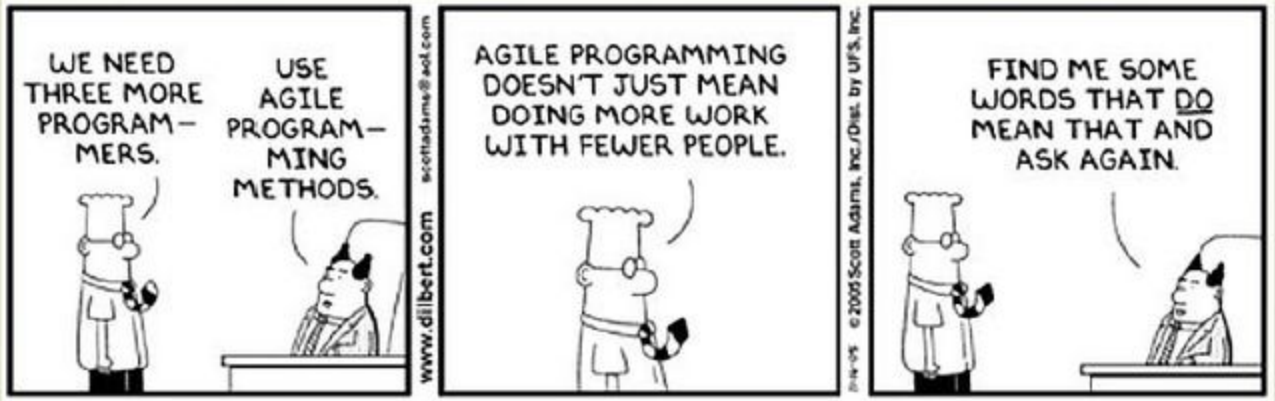
Responding to change over Following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

From Wikipedia

Popular agile software development frameworks include (but are not limited to):

- Adaptive software development (ASD)
- Agile modeling
- Agile Unified Process (AUP)
- Crystal Clear methods
- Disciplined agile delivery
- Dynamic systems development method (DSDM)
- Extreme programming (XP)
- Feature-driven development (FDD)
- Lean software development
- Kanban
- Rapid application development (RAD)
- Scrum
- Scrumban



Backlog: a list of user stories, features, bugs, issues, or other tasks that might eventually be worked on. Size often estimated in arbitrary *points*.

Timebox (or sprint): a fixed length of time in which some amount of work will be done, resulting in a usable/shippable product. 2-4 weeks is typical.

Project owner: orders the backlog according to risk or return. Ideally the customer.

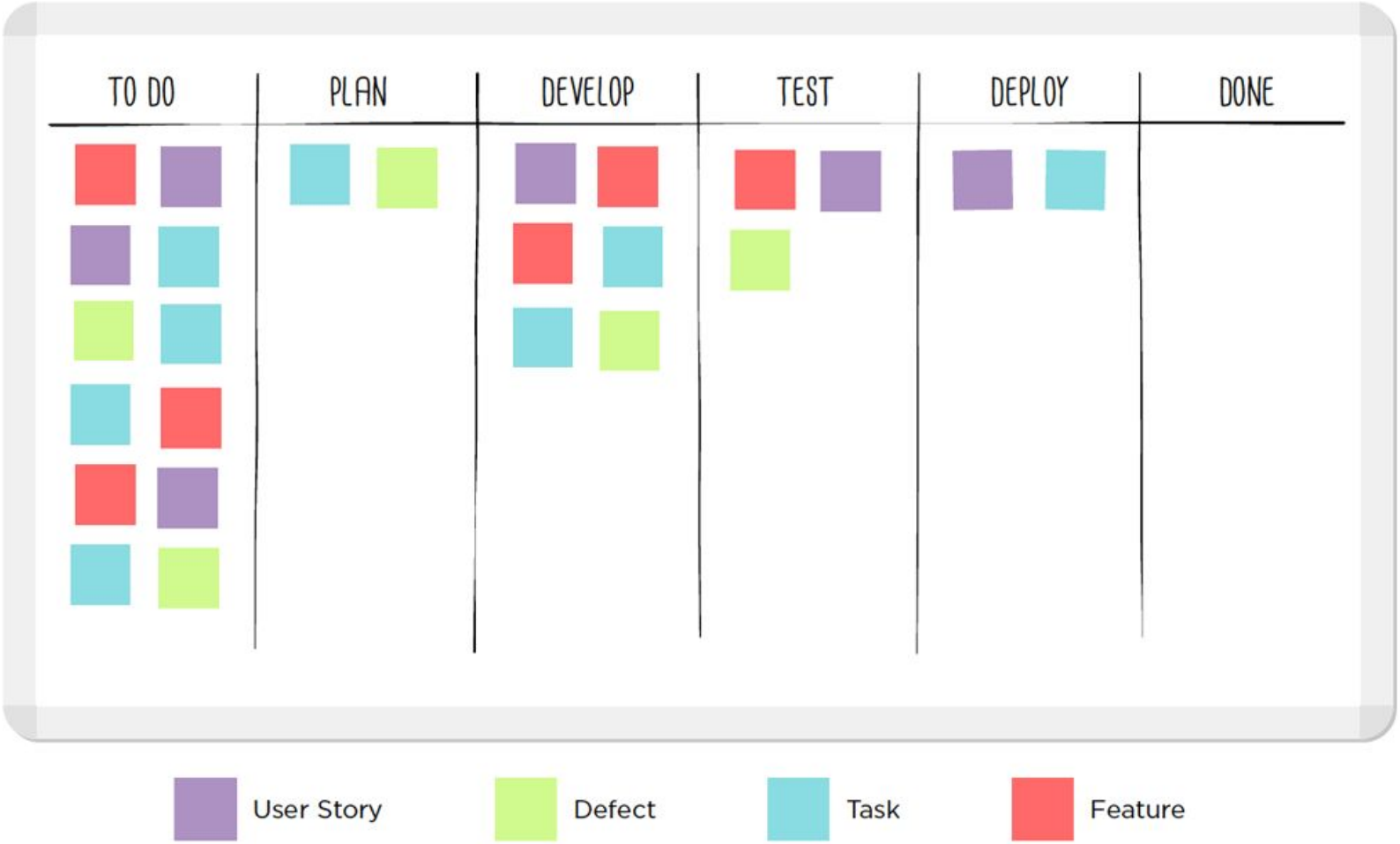
Stand-up meeting

- What did I do yesterday?
- What will I do today?
- What's blocking me?

Meant for general awareness and initiating follow-up meetings between team members. *Not* a problem-solving meeting.

Brief enough to be conducted standing up.

Information radiators: Kanban board



Release trains

Linux kernel: release trains branch every ten weeks. Release candidates are made available for testing. Only bug fixes are added to the train after branching.

Firefox: Aurora (two releases ahead), Beta (one release ahead) and main release channels. Uplift requests to merge to Aurora and Beta require approval based on risk and user impact if declined. Chemspill releases for showstopper bugs or security issues discovered after release.

Windows: internal betas, community technology previews, consumer preview, release to manufacturing, general availability, service packs.

Windows 10: fast ring, slow ring, release preview

Minimum Viable Product

From Disciplined Entrepreneurship p237

There are three core elements necessary to have a Minimum Viable Product. All three must be present for this step to be successfully completed.

- The customer gets value out of use of the product.
- The customer pays the time and effort necessary to gain use of the product.
- The product is sufficient to start the customer feedback loop, where the customer can help you iterate toward an increasingly better product.

Your odds of success are higher if you limit the number of variables in your initial product, getting something that works into the customer’s hands quickly, even if it does not have all the functionality you would like to provide.