Agile Software Development

ENGI 5895: Software Design

Andrew Vardy

Faculty of Engineering & Applied Science
Memorial University of Newfoundland

March 5, 2018

Software Failures

Figures from "Why Software Fails" by Robert N. Charette, IEEE Spectrum, 2005.

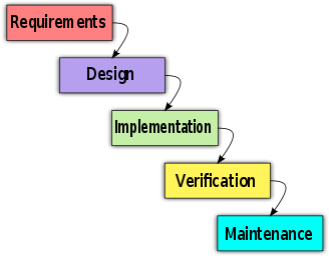
Software is ubiquitous and is constantly growing in size and complexity:

- `http://www.informationisbeautiful.net/visualizations/million-lines-of-code/`

Large software projects have a high failure rate. Estimated cost for failed software projects in the US: $60-70 Billion / year.
Software developers spend 40-50% of their time on "avoidable rework" as opposed to "work that's done right the first time".

The Waterfall Process

To mitigate failure, software engineers and managers establish **processes**, with scheduled outputs (documents, code, test results,...). The classic software development process is known as the waterfall process:



From `http://en.wikipedia.org/wiki/Waterfall_process`
The idea is to complete each phase in sequence before moving onto the next. **Adv.**: Encourages getting things right before moving on.

Problems with the Waterfall Process

- The waterfall process requires reports, meetings, and evaluations at every stage. Scheduling and executing all of this extra work is time-consuming.
- The artefacts and constraints of this process are not sufficient to prevent errors, so more artefacts and constraints are imposed.
  - Sounds paradoxical but this is how organizations often behave!
- As the development process becomes more and more cumbersome, the schedule slips.
- The customer may wish to modify the requirements, or it may be realized that the original requirements were poorly specified.
  - Leads to a massive cascade of changes through all subsequent stages of the process!

Agile Software Development

A set of values established by a group of industry experts in 2001 to allow software teams to work quickly and respond to change:

Manifesto for Agile Software Development

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.

Individuals and interactions over processes and tools

Some advice from "Agile Software Development" by Robert. C. Martin (Bob)

Individuals and interactions:

> *A team of average programmers who communicate well are more likely to succeed than a group of superstars who fail to interact as a team.*

Tools:

> *Don't assume you've outgrown a tool until you've tried it and found you can't use it. [...] Before you commit to the top-shelf behemoth database system, try flat files. Don't assume that bigger and better tools will automatically help you do better. Often they hinder more than help.*

Working software over comprehensive documentation

> *Huge software documents take a great deal of time to produce and even more time to keep in sync with the code. If they are not kept in sync, then they turn into* ***large, complicated lies*** *and become a significant source of misdirection.*
> *[...]*
>
> *It is always a good idea for the team to write and maintain a rationale and structure document, but that document needs to be short and salient. By "short" I mean one or two dozen pages at most. By "salient," I mean it should discuss the overall design rationale, and only the highest-level structures in the system.*

Customer collaboration over contract negotiation

> *Successful projects involve customer feedback on a regular and frequent basis. Rather than depending on a contract or a statement of work, the customer of the software works closely with the development team, providing frequent feedback on their efforts.*

Responding to change over following a plan

It is tempting for novice managers to create a [...] Gantt chart of the whole project [...] What really happens is that the structure of the chart degrades. As the team gains knowledge about the system, and as the customers gain knowledge about their needs, certain tasks on the chart become unnecessary. [...] In short, the plan undergoes changes in shape, not just changes in dates.

A better planning strategy is to make detailed plans for the next two weeks, very rough plans for the next three months, and extremely crude plans beyond that.