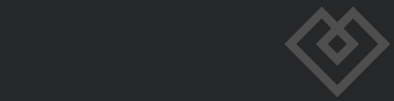
Software product development

Denis Sušac
http://www.mono-software.com

SSA 2014

“Natural” Phases of a Software Project by Scott Guthrie

- Enthusiasm
- Disillusionment
- Panic
- Search for the Guilty
- Punishment of the Innocent
- Praise and Honors for Non-Participants



Management by miracle

Creating an unrealistic plan, turning it prematurely into a promise, and when it becomes clear it isn't going to happen, praying for a miracle instead of updating the plan

It does not have to be that way...

Joel Spolsky's 12 steps to better code,
http://www.joelonsoftware.com/articles/fog0000000043.html

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

A bit of history

Waterfall software development manifesto

Software development can be equated to any other engineering task. We believe software development projects can be effectively managed by:

- Understanding and writing specifications that define how the software will look and what it will do

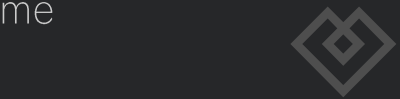
- Performing in-depth analysis and design work before estimating development costs

- Ensuring software developers follow the specifications

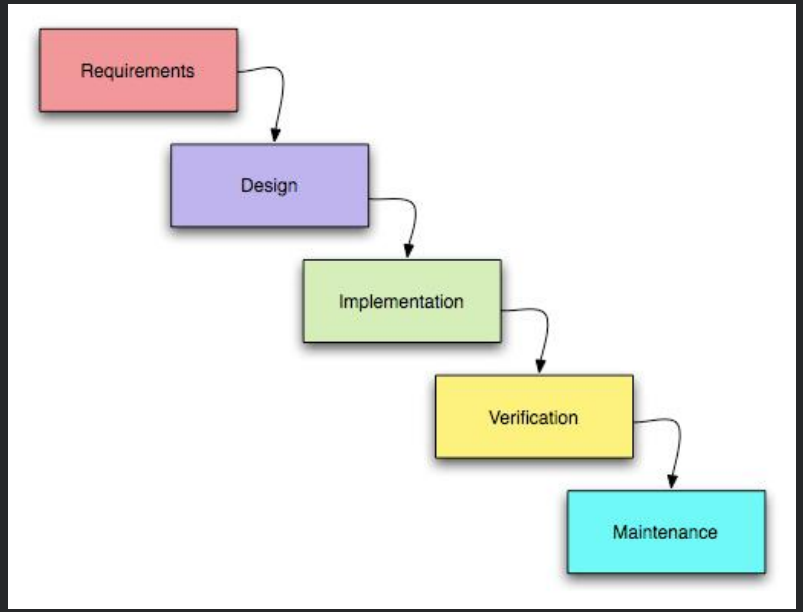
- Testing the software after implementation to make sure it works as specified, and delivering the finished result to the user

- That is, if the specification is of sufficient detail, then the software will be written such that it will satisfy the customer, will be within budget, and will be delivered on time

Will it?

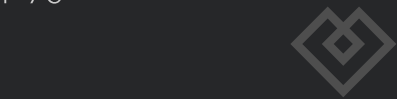


Waterfall explained



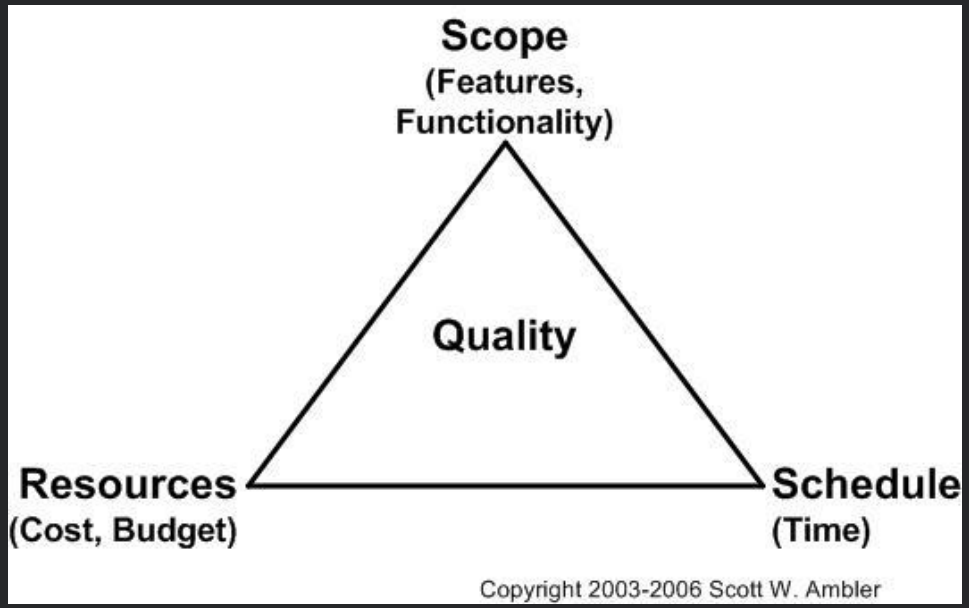
What's wrong with it?

- "I'll Know it When I See It"
- Software development is more like new product development than manufacturing
- Software development is far behind more traditional engineering fields in terms of development methods
- 70% of software projects using this methodology fail to meet their objectives.
- The fail rate of going over Niagara Falls in a barrel is 34%



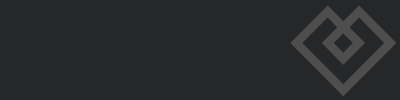
Project management triangle

Pick any two: fast, good, cheap



Three simple truths

1. It is impossible to gather all requirements at the beginning of the project
2. Whatever requirements you do gather is guaranteed to change
3. There will always be more to do than time and money will allow



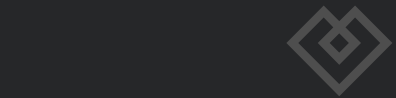
Agile manifesto

http://agilemanifesto.org/iso/hr/

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

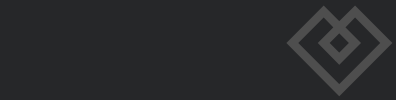
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

While there is value in the items on the right, we value the items on the left more.

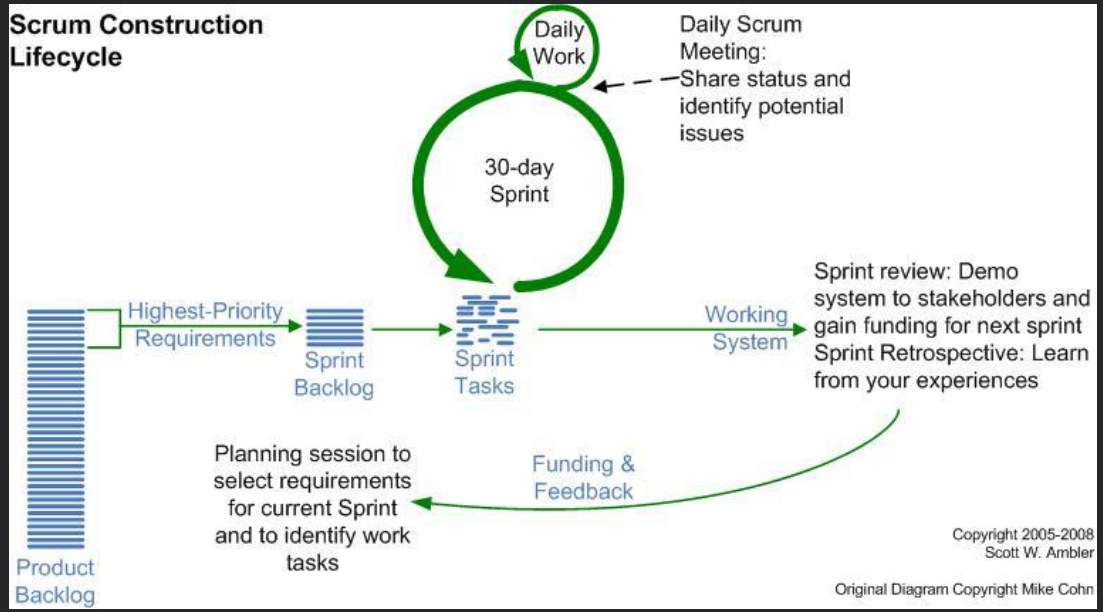


What is Agile?

Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end.



How it works?


Scrum Construction Lifecycle
Daily Work
Daily Scrum Meeting: Share status and identify potential issues
30-day Sprint
Highest-Priority Requirements
Sprint Backlog
Sprint Tasks
Working System
Sprint review: Demo system to stakeholders and gain funding for next sprint
Sprint Retrospective: Learn from your experiences
Planning session to select requirements for current Sprint and to identify work tasks
Funding & Feedback
Product Backlog
Copyright 2005-2008 Scott W. Ambler
Original Diagram Copyright Mike Cohn

User stories

User stories are short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

As a <type of user>, I want <some goal> so that <some reason>.

They strongly shift the focus from writing about features to discussing them. These discussions are more important than whatever text is written.

Examples

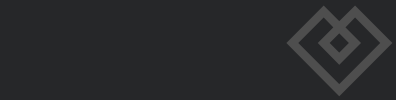
Epic

- As a user, I can backup my entire hard drive.

Breakdown

- As a power user, I can specify files or folders to backup based on file size, date created and date modified.

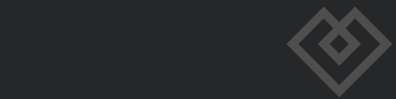
- As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.



Adding details

- splitting a user story into multiple, smaller user stories.
- adding “conditions of satisfaction.”

The conditions of satisfaction is simply a high-level acceptance test that will be true after the agile user story is complete.



Example

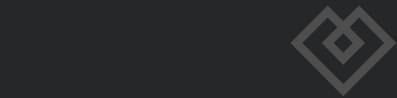
- As a vice president of marketing, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones.

Conditions of satisfaction:

- Make sure it works with major retail holidays: Christmas, Easter, President's Day, Mother's Day, Father's Day, Labor Day, New Year's Day.

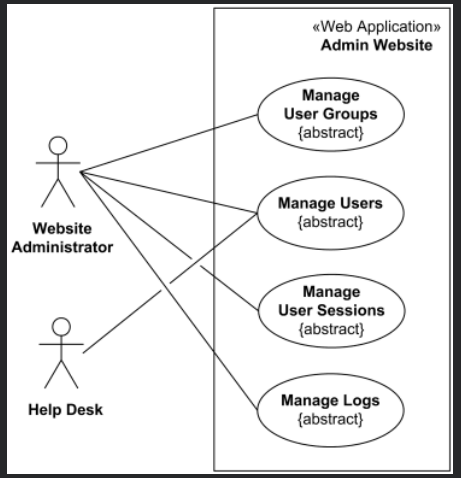
- Support holidays that span two calendar years (none span three).

- Holiday seasons can be set from one holiday to the next (such as Thanksgiving to Christmas).



Alternatives

- Prototyping/wireframing
- Use cases



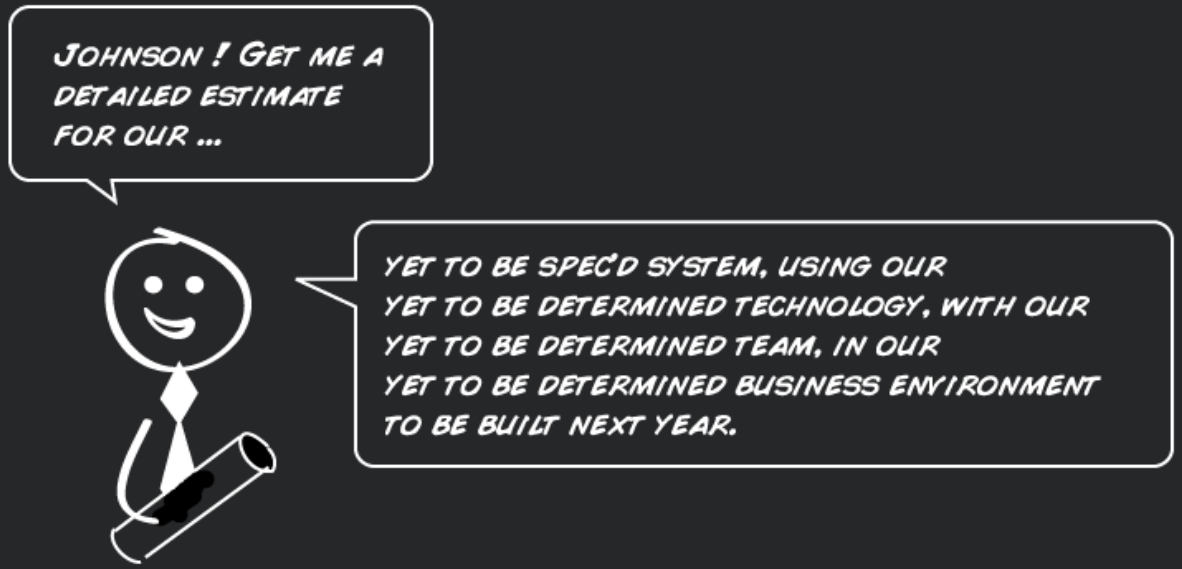
System requirements document

Q-Cash System Requirements Document

QCASH

Table of Contents

Estimation

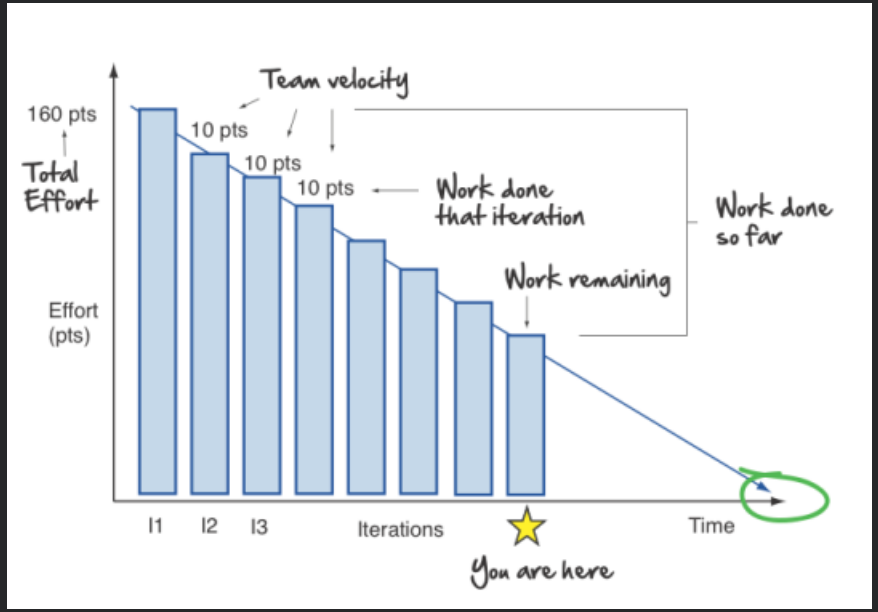

JOHNSON ! GET ME A DETAILED ESTIMATE FOR OUR ...
YET TO BE SPEC'D SYSTEM, USING OUR
YET TO BE DETERMINED TECHNOLOGY, WITH OUR
YET TO BE DETERMINED TEAM, IN OUR
YET TO BE DETERMINED BUSINESS ENVIRONMENT
TO BE BUILT NEXT YEAR.

Relative vs absolute

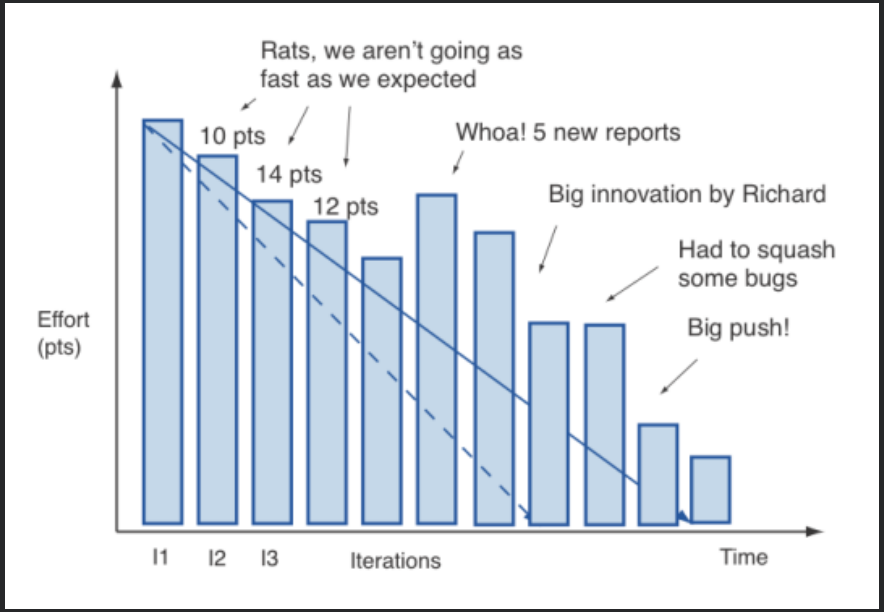
We aren't very good at estimating things absolutely, it turns out we are pretty good at estimating things relatively.

Story points

How fast are we going?

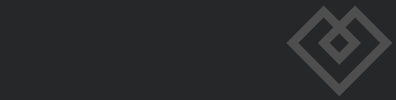


More realistically...



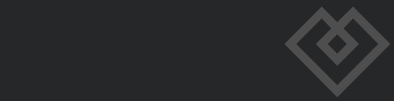
Engineering

- unit testing
- refactoring
- continous integration
- test-drived development



Important practices

- coding standards
- code reviews
- static analysis
- dynamic analysis
- testing: unit testing, functional testing, performance testing, penetration testing...



Tools

- version control
- bug and issue tracking
- time tracking
- wireframing
- continuous integration

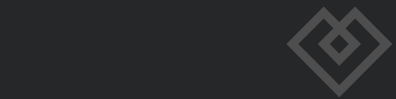
Issues with the pure engineering approach

Engineers often seem to believe that they can completely streamline the process of building software, with complete control and predictability. And, this belief seems to be process independent.

In other words, even engineers who are strong advocates of agile often believe that if you do a, b, and c, you will get the results you want.

Some engineers seem to think that best practices apply everywhere or in every situation.

More experienced engineers are aware that different contexts require different best practices.



Craftmanship

Craftsmen see software development more as an art that emerges and less as a science that can be controlled, so they let time and experience shape up their skills in successfully completing software projects - similar to traditional medieval craftsmanship where an apprentice learns a craft under the guidance of a master gradually climbing his way into becoming a skilled journeyman

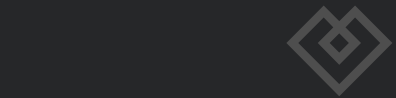
Even more importantly...

Control is ultimately illusory on software development projects. If you want to move your project forward, the only reliable way to do that is to cultivate a deep sense of software craftsmanship and professionalism around it.

The guys and gals who show up every day eager to hone their craft, who are passionate about building stuff that matters to them, and perhaps in some small way, to the rest of the world – those are the people and projects that will ultimately succeed.

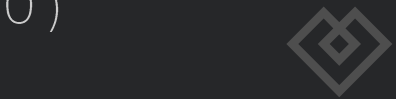
Everything else is just noise.

Jeff Atwood



A couple of tips

- Outsource everything that is not core to the business. The leanest companies will identify as few core competencies as possible, and outsource everything.
- 85% of all developer time should be spent on business-specific logic. Do not reinvent the wheel! (https://medium.com/on-management/4578569760e8)
- Know your stuff! (http://www.informationweek.com/architecture/why-your-software-development-process-is-broken/d/d-id/1111510)



Thank you!