

# Introduction to Agile Software Development

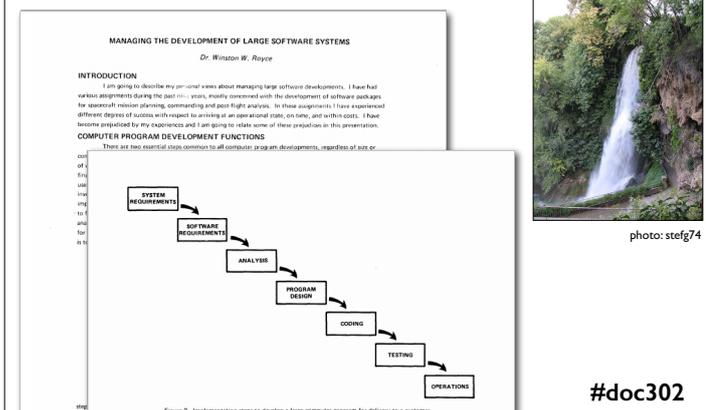
Dr Robert Chatley - [rbc@doc.ic.ac.uk](mailto:rbc@doc.ic.ac.uk)

 @rchatley #doc302

In this section we introduce a style of software development that is currently prevalent in the industry. Most teams developing software commercially follow these practices to some degree.

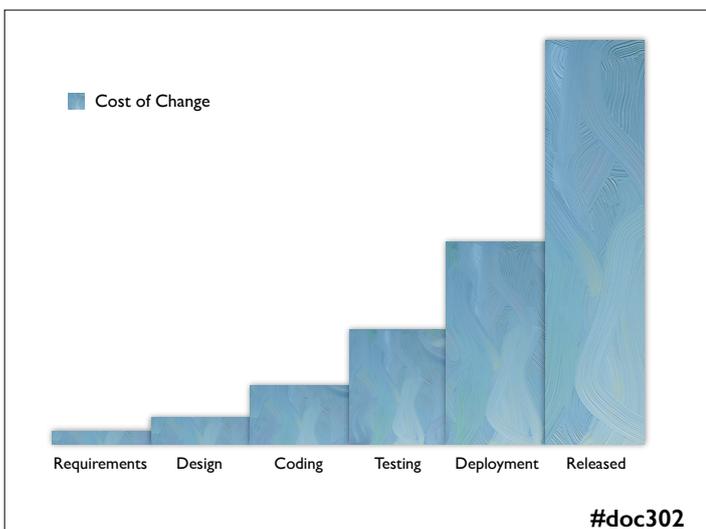
To effectively deliver your Group Projects, you will likely want to adopt some of these methods and techniques. We will illustrate different techniques, but you will want to read and research further to choose something that is appropriate for your project.

## Waterfall Development



Winston W. Royce wrote a paper in 1970 called "Managing the Development of Large Software Systems". Unfortunately this paper was somewhat misinterpreted by the industry at large, and what emerged was the Waterfall Model of software development. It set out a number of different phases of software design and implementation, which in the Waterfall model flow from one to the other sequentially.

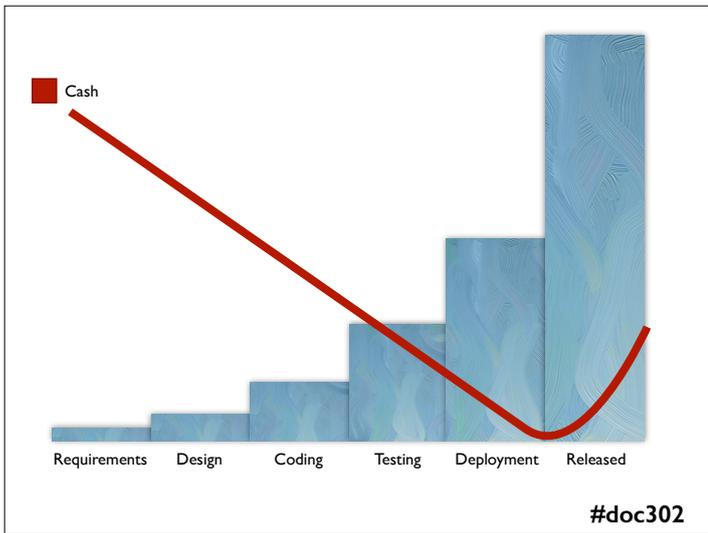
One of the problems with this is that we only get one shot at getting things right - there isn't much scope for going back and reworking things after a phase ends. As it happens, this wasn't actually what Royce meant when he wrote the original paper, but it was what people took from it and it stuck for quite a few years.



Projects developed according to the waterfall model proceed through phases, from requirements analysis, through design, implementation and testing, until they are released. The later a defect is detected and fixed, the more expensive it is, so much effort is put in to analysis phases early on. Unfortunately, Waterfall has not proved a good model for software projects, with many being delivered late or going over budget.

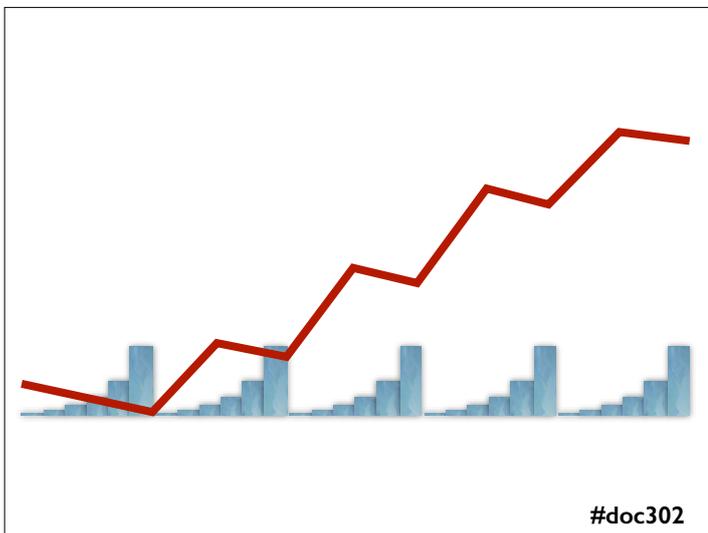
This graph shows the Cost of Change curve by Barry Boehm, described in "Software Economics", written in the 1980s. It shows how if a change of requirements (or a bug) is detected early then it is relatively cheap to make this change (maybe the code is not written yet). But if the change comes up late in the project, it can be very expensive to rework things.

Following this model, engineers spent a lot of time on analysis and requirements early in the project to try and



One problem with releasing at the end of the project, once everything is finished, is that no value is returned by the project until the end. For the majority of the life of the project, we are spending money, but not making any money back.

In terms of projects like your Group Projects, if we only have one delivery point at the end, it makes the project very risky. How will we know whether or not we are building the right thing and if people are happy with it? What if we miss the deadline and do not deliver a working system?

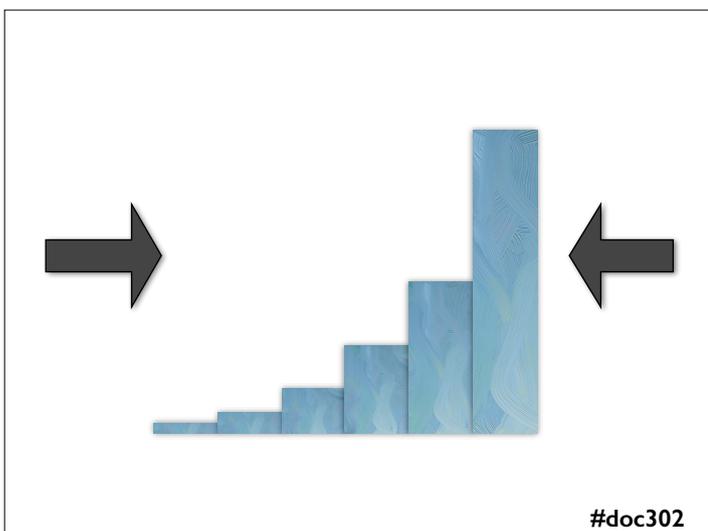


Agile methods favour an iterative approach. Rather than proceeding in phases, we iteratively design, build and release small sets of features. We aim to deliver value from the first release, which should be early in the project.

Agile tries to reduce the time between someone having an idea and its implementation in software that can be used. We compress the development waterfall to a succession of small cycles, which we perform iteratively over and over again.

The first release happens sooner, meaning that our software can start generating revenue earlier. This is cumulative, so if we prioritise high value features first, we should quickly get a return on investment. If we don't, perhaps the best outcome is to cancel the project - early!

We also increase the speed and frequency with which we learn/validate something about what the customer (really)



We want to make these cycles small, so that we can get feedback as early and often as possible. But we do not want to sacrifice any of the engineering rigour that goes into making a reliable, well-tested, product.

To do this, we compress the analysis-code-test-release cycle, and perform it many times during the project. Each of these iterations lasts a short period of time, typically a week, perhaps a few weeks on large projects, and increasingly commonly, much less than a week in some highly dynamic environments.

**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.

|                   |                |                  |
|-------------------|----------------|------------------|
| Kent Beck         | James Grenning | Robert C. Martin |
| Mike Beedle       | Jim Highsmith  | Steve Mellor     |
| Arie van Bennekum | Andrew Hunt    | Ken Schwaber     |
| Alistair Cockburn | Ron Jeffries   | Jeff Sutherland  |
| Ward Cunningham   | Jon Kern       | Dave Thomas      |
| Martin Fowler     | Brian Marick   |                  |

<http://agilemanifesto.org/>

The Agile Manifesto (<http://agilemanifesto.org>) was drawn up by some pioneers in the field, in 2001. It does not define a particular method, but gives some overriding principles that the authors though would lead to effective software development practice.

They wanted to focus on improving the way that people interacted and communicated to build the right software, rather than necessarily building processes and tools, or setting out the specification as a long document at the beginning of the project that was hard to change later. They wanted to favour delivering working software over writing complex design documents. They acknowledged that having some sort of a plan for a project was a good idea, but that to deliver the best and most valuable software, we should be happy to change that plan when we get new information.

**Extreme Programming Explained**  
EMBRACE CHANGE  
KENT BECK  
with CYNTHIA ANDRES  
Foreword by Erich Gamma  
Second Edition

**Agile Software Development with Scrum**  
red  
yellow  
green  
blue  
red  
blue  
yellow  
green  
blue  
Ken Schwaber • Mike Beedle

**KANBAN**  
Successful Evolutionary Change for Your Technology Business  
David J. Anderson  
Foreword by Thomas H. Davenport

#doc302

There are several different development methods or processes that all come under the banner of agile. Here are three:

Extreme Programming (XP) is one of the original agile methods, and works well for software projects. The method includes project management techniques, as well as technical practices to help us deliver reliable software quickly.

Scrum concentrates more on the project management methods, and does not talk specifically about building software.

Kanban is a more recent method in software circles, that gets rid of the timeboxed iterations from Scrum and XP, and aims for a continuous flow of work. It is influenced by Japanese manufacturing techniques, particularly from companies like Toyota.

Watch this Video

<http://www.theguardian.com/technology/video/2013/jun/13/geeks-opened-up-government-video>

#doc302

This video shows how agile practices have been put to use by the Government Digital Service, a group working in the UK civil service since 2011 re-engineering the technology that allows the general public to interface with government. The civil service has traditionally been a slow-moving establishment, so it is interesting to hear the benefits they have found from embracing quicker iteration and modern methods.

<http://www.theguardian.com/technology/video/2013/jun/13/geeks-opened-up-government-video>