Introduction to Agile Development

School of Electronics, Electrical Engineering & Computer Science

Queen's University, Belfast

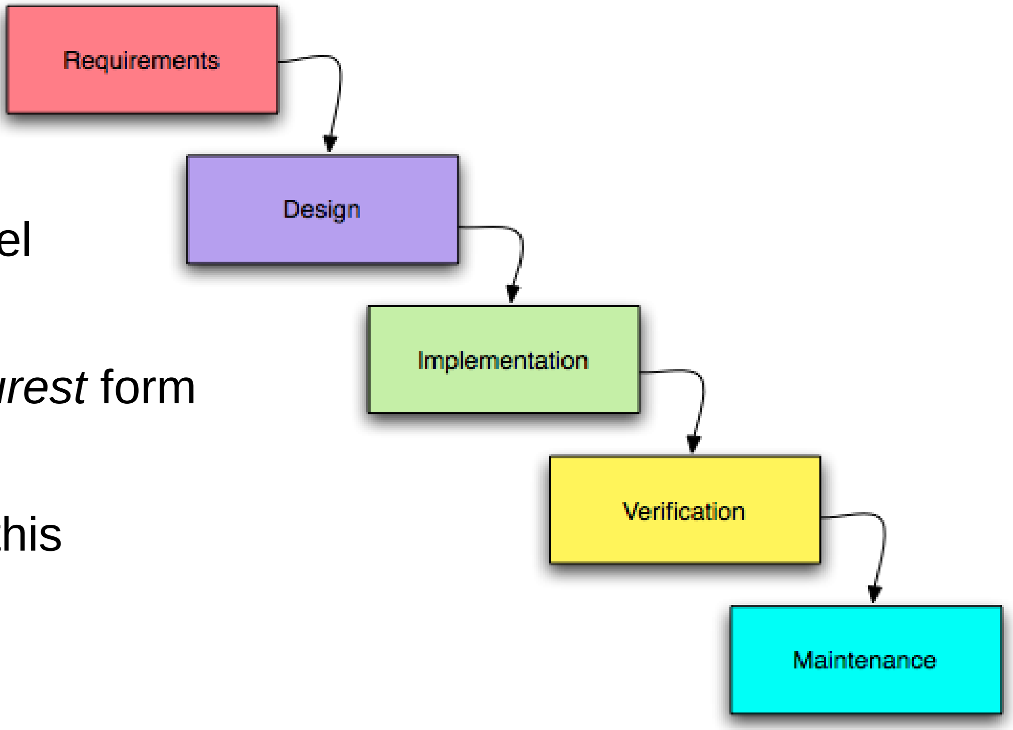
Dr Darryl Stewart

Outline

- Plan Driven Development
- Defined *vs* Empirical processes
- Agile Development
- Agile Manifesto
- Agile Principles
- Agile Methodologies

Plan-driven development approaches

- Typified by the **Waterfall** model
- This diagram shows it in its *purest* form
- There are several variants of this



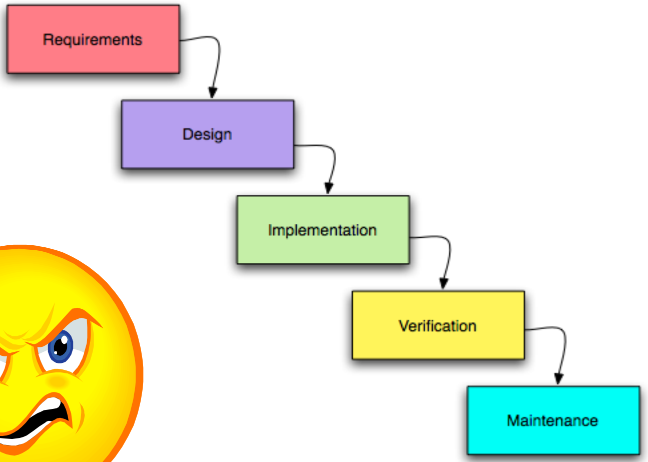
The origin of the term "waterfall" is often cited to be an article published in 1970 by Winston W. Royce, although Royce did not use the term "waterfall" in this article. Ironically, Royce was presenting this model as an example of a flawed, non-working model.

Wikipedia (!)

Plan-driven development approaches

- Write down **advantages/good things** about this model

- Write down **disadvantages/bad things** about this model



Plan-driven development approaches

- Classes **advantages/good things** about this model

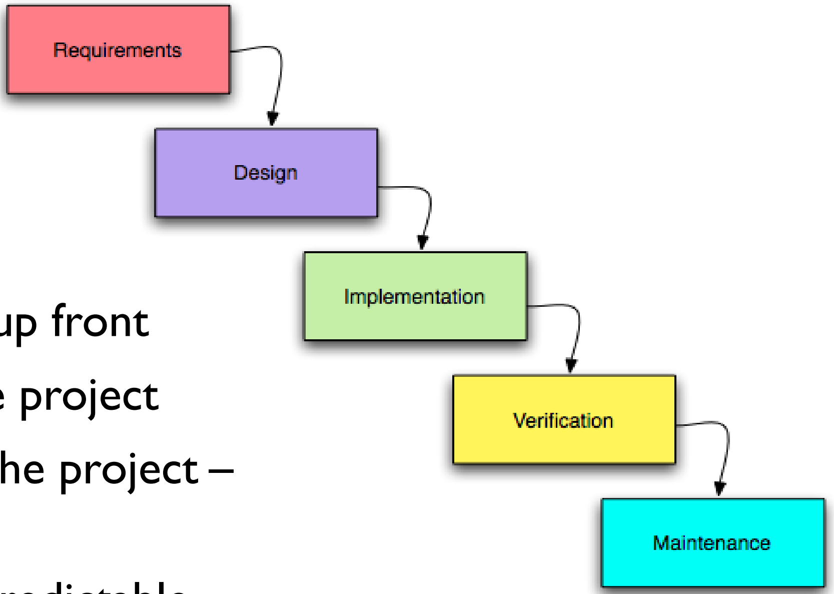
•Client gets a clear schedule and delivery time
•Documentation is great (developers, management)
•Specialists on each phase (can plan and do what they enjoy most)
•If requirements are stable then very efficient
•It is easy to see where you are (developer)
•Encourage clarity in requirements
•Easier to management scope creep (Team)

Plan-driven development approaches

- Classes **disadvantages/bad things** about this model
- Forced to move onwards cant go back
- Hard to prioritise the work
- Client not involved enough
- Specialists waiting for work downstream
- Slow to get working system
- Changes are expensive and difficult to make
- TDD is not possible

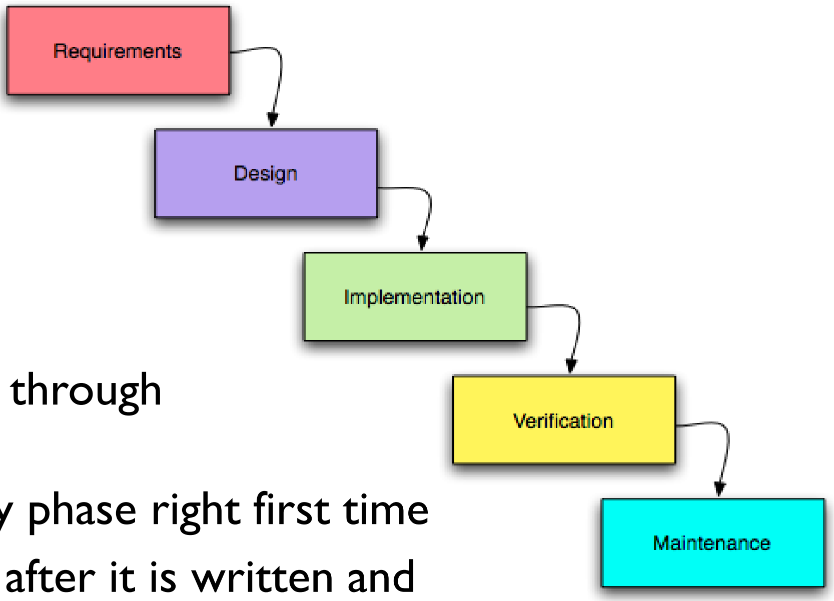
Plan-driven development approaches

Some often quoted arguments for



- Allows management to develop a large plan up front
- Each phase marks a discrete milestone in the project
- Each phase produces documentation about the project – usually in a formalised format
- Can be viewed as disciplined approach and predictable for the developers
- "Better to measure twice and cut once"
- Independent teams/individuals can work on each phase
- The development company can fix the contract after the requirements phase

Plan-driven development approaches



Some often quoted arguments against

- Many details only become clear as you progress through implementation
- Considered by some to be impossible to get any phase right first time
- Formal documentation is often redundant soon after it is written and provides little (if any) value to the Customer
- Effort used to specify and design for low priority requirements up front delays delivery of high priority requirements
- Difficult for changes to be requested in later stages
- Independent teams/individuals can work on each phase!
- The contract can be fixed after the requirements phase!

Defined *vs* Empirical Processes

In Manufacturing Theory

- A **Defined** process control model assumes that the process will start, run until completion and produce predictable results each time
- For example building *things* on an assembly line from component parts
- A Defined process control model is appropriate if the process requires little creativity to achieve the results and there is very little potential variance (change) during the process

- Using a Defined approach to control software development, means you have to rely heavily on **assumptions of stability** in an attempt to plan sufficiently
- The waterfall model (plan driven approach) could be described as a defined process control model
- This can provide a false sense of security… things tend to change

Defined *vs* Empirical Processes

- What are the "sources of change" or "things that could change" during a software development project?

Class suggestions

- Clients budget
- Reacting to competition
- New technology
- Wrong requirements
- Legislation
- Leaked info
- Staff change – new devs

Defined *vs* Empirical Processes

- The software development process can be subject to a lot of change
  - ***Customer requirements***
  - *Development team – growing, shrinking etc.*
  - *Organisation – management changes, reporting mechanisms etc.*
  - ***Market** – competitors etc.*
  - *Technology – languages, platforms*
  - …
- Any non-trivial project requires considerable creativity, investigation, research and problem solving

- These are characteristics of something that requires an **Empirical** process control model <u>not</u> a **Defined** process control model

Defined *vs* Empirical Processes

An **Empirical process control model** involves managing a project using:

- **Short "inspect and adapt" cycles**
- **Frequent, short feedback loops**

These are the core common characteristics of **Agile** methods for software development

Defined *vs* Empirical Processes

Is Predictability Impossible?

In general, no. There are some software developments where predictability is possible. Organizations such as NASA's space shuttle software group were a prime example of where software development can be predictable. It requires a lot of ceremony, plenty of time, a large team, and **stable requirements**. There are projects out there that are space shuttles. However I don't think much business software fits into that category. For this you need a different kind of process.

"The New Methodology", Martin Fowler

Agile Development

- Agile development is a form of **iterative development**
- Where each iteration could be viewed as a mini project with :
  - Requirements analysis
  - Design
  - Implementation
  - Testing

- Constantly integrating all of the newly developed software with that which was developed in previous iterations
- The final system evolves over a series of iterations
- **How does this differ from prototyping?**

Agile Development

- Methods considered to be 'Agile' emphasise **short iterations** (1 - 4 weeks)
- Feedback from each previous iteration and new information allow frequent refinement and adaptation for future iterations
- Gives more frequent feedback to the customer
  - Can observe the true rate of progress and plan accordingly
  - Can get working code earlier with potential business value

Pareto's principle (AKA: the 80/20 rule)

- Getting the last 20% of the requirements implemented will take 80% of the time
- 20% of the features are used 80% of the time
- Etc.

- Allows more frequent refinement of requirements by customers
- Small iterations mean lower complexity in each and therefore less risk

Agile Development

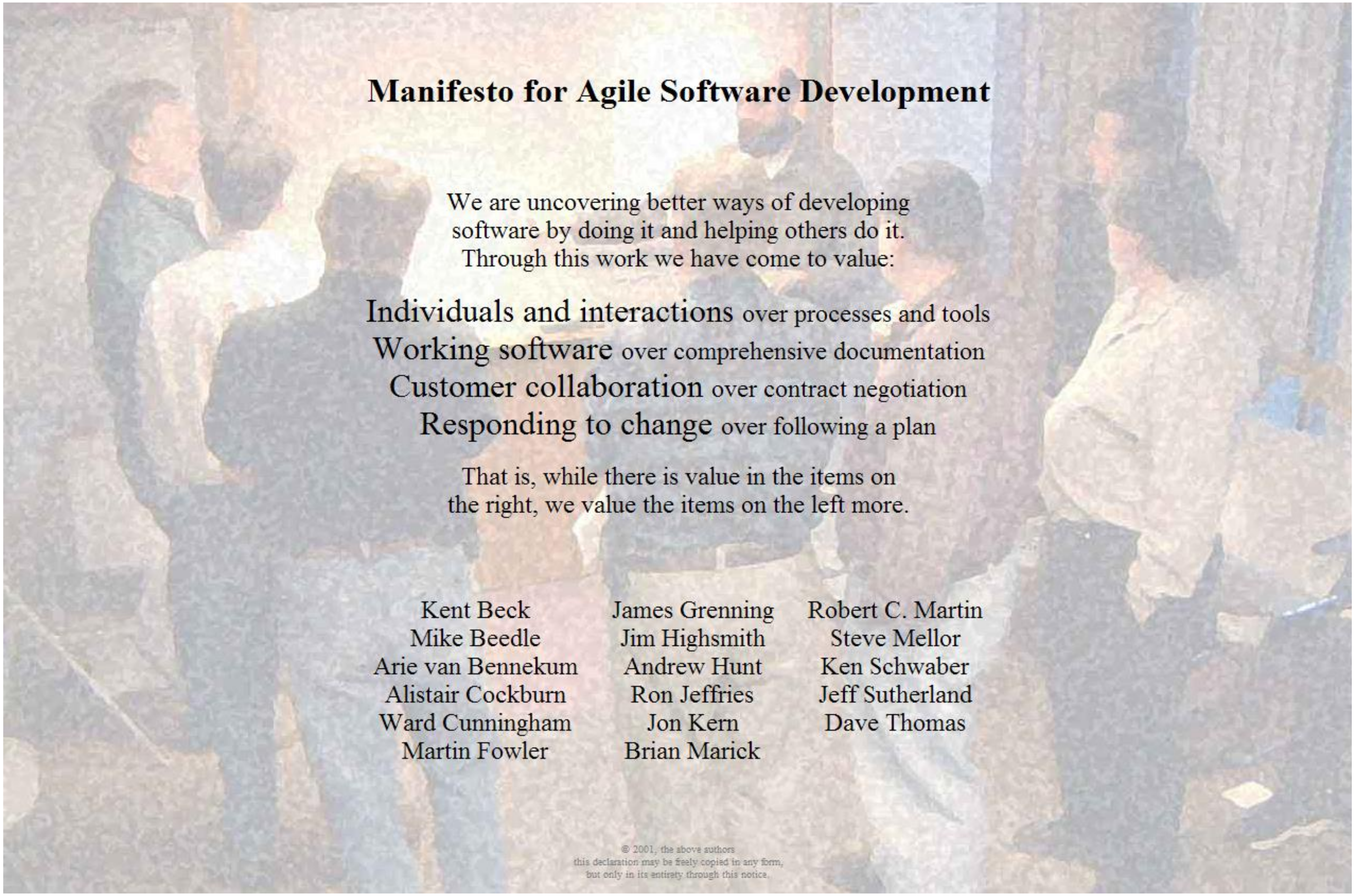
Timeboxing of work into short iterations has several benefits

- The development teams have a close range motivating goal to work for

"… work expands to fill the time allocated for it" [Parkinson 1954] (http://www.economist.com/node/14116121)

So … restrict the time available and efficiency/productivity expands

- The work of the development team gets a rhythm which promotes sustainable working practices
  - Planning becomes easier
  - There is less risk of missing targets
  - If targets are missed it is detected quickly and the consequences are not overly severe



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	

Principles behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through **early and continuous delivery** of valuable software.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a **preference to the shorter timescale**.
4. **Business people and developers must work together** daily throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

Principles behind the Agile Manifesto

7. **Working software is the primary measure of progress**.
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to **maintain a constant pace** indefinitely.
9. Continuous attention to **technical excellence** and **good design** enhances agility.
10. **Simplicity--**the art of maximizing the amount of work not done**--**is essential.
11. The best architectures, requirements, and designs emerge from **self-organizing teams.**
12. At regular intervals, the **team reflects** on how to become more effective, then **tunes** and **adjusts** its **behaviour accordingly**.

Agile Methodologies

Some of the most well known methodologies are:

- eXtreme Programming (XP)
- Scrum

Probably the best known methods

XP@Scrum

- Crystal (various colours)
- Dynamic Systems Development Methodology (DSDM)
- Feature Driven Development (FDD)
- Adaptive Software Development (ASD)
- Agile Modelling
- Lean Development (Some people hotly debate if this falls under the agile umbrella)
- We will be looking primarily at **XP and Scrum** in this module

Take home messages

- Agile is an empirical approach to software development
- There are pros and cons of each approach
- Agile methods use **time boxing**
- There is a **Manifesto** and a set of **Principles**
- We are going to focus on **Scrum** and **XP**