Software development processes: from the waterfall to the Unified Process

Nigel Goddard

School of Informatics
University of Edinburgh

The Waterfall Model

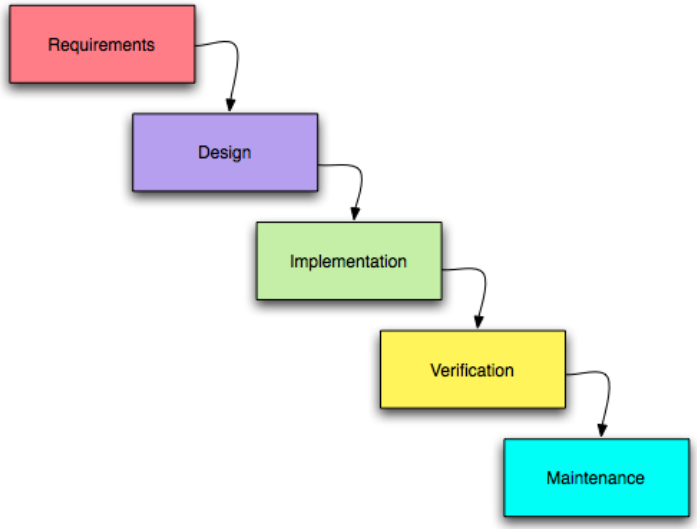


Image from Wikipedia

Pros, cons and history of the waterfall

+ better than no process at all – makes clear that requirements must be analysed, software must be tested, etc.
– inflexible and unrealistic – in practice, you cannot follow it: e.g., verification will show up problems with requirements capture.
slow and expensive – in an attempt to avoid problems later, end up "gold plating" early phases, e.g., designing something elaborate enough to support the requirements you suspect you've missed, so that functionality for them can be added in coding without revisiting Requirements.

Introduced by Winston W. Royce in a 1970 paper

as an obviously flawed idea!

Spiral models

Split project into controlled iteration: each iteration is a mini-waterfall.

+ Mitigate risk. E.g. check user requirements, try out technology, practice new techniques in an early iteration to catch errors before main cost of project starts.
– Cost: e.g., repeated testing and documentation. A few projects are so low risk that iteration isn't cost-effective.

In practice, need for rework: essential to allow time for refactoring.

Big projects need different approaches to different iterations.

Steps towards the Unified Process

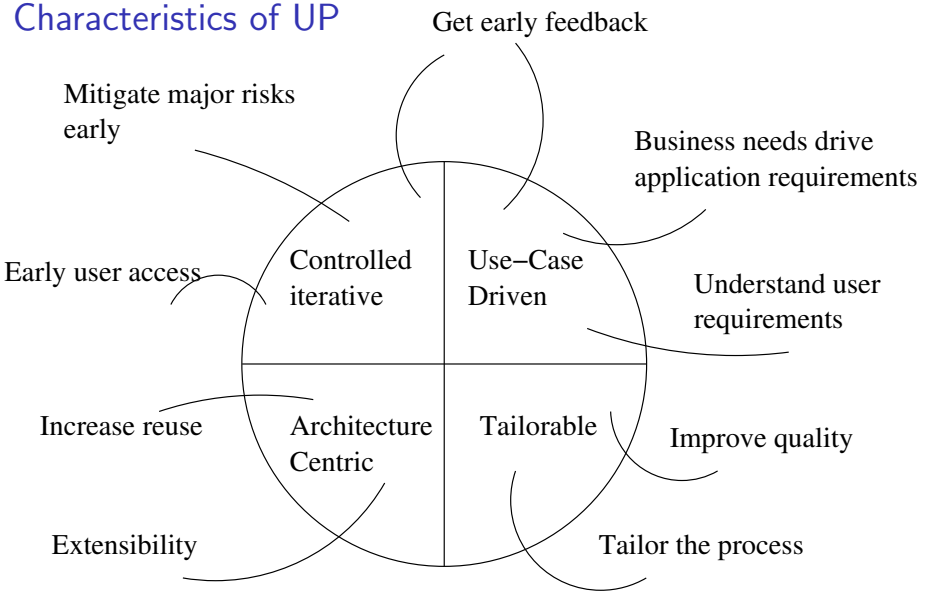
- 1960s - 1987: Ivar Jacobson at Ericsson: early component-based development, architectural block diagrams.
- 1987-1995: Jacobson founded Objectory (contraction of "Object factory"), added use cases
- 1995: Grady Booch, Jim Rumbaugh and Ivar Jacobson together at Rational, which bought Objectory. "The methods war is over – we won." First version of Unified Method produced. Controversial: quickly overshadowed by UML.
- 1995-1997: Rational Objectory: added controlled iteration
- 1998: (Rational) Unified Process

Unified process: the public domain, generic ideas

Rational unified process: more detailed, commercial. Now IBM.

Lots of variants, e.g. OpenUP, EnterpriseUP...

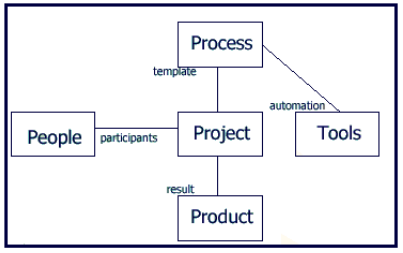
Characteristics of UP



(adapted from Rational slide)

The four Ps

Four equally-important aspects of the software engineering process

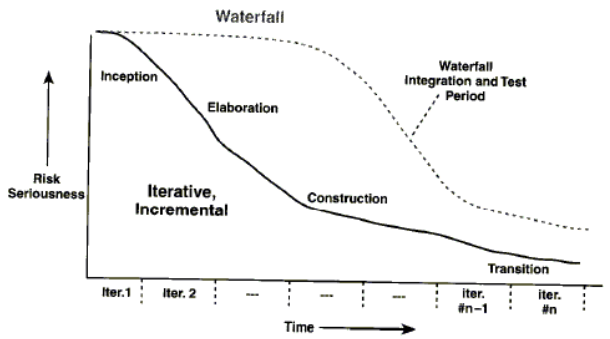


- People - do everything
- Project - make the product
- Product - not just code
- Process - organises the project
  - Tools - support process

UP phases (iterative: end with review)

- Inception ends with commitment from the project sponsor to go ahead: business case for the project and its basic feasibility and scope known.
- Elaboration ends with
  - basic architecture of the system in place,
  - a plan for construction agreed,
  - all significant risks identified,
  - major risks understood enough not to be too worried.
- Construction (definitely iterative) ends with a beta-release system .
- Transition is the process of introducing the system to its users.

UP phases: risk management



(adapted from Rational slide)

Workflows: 9 activites

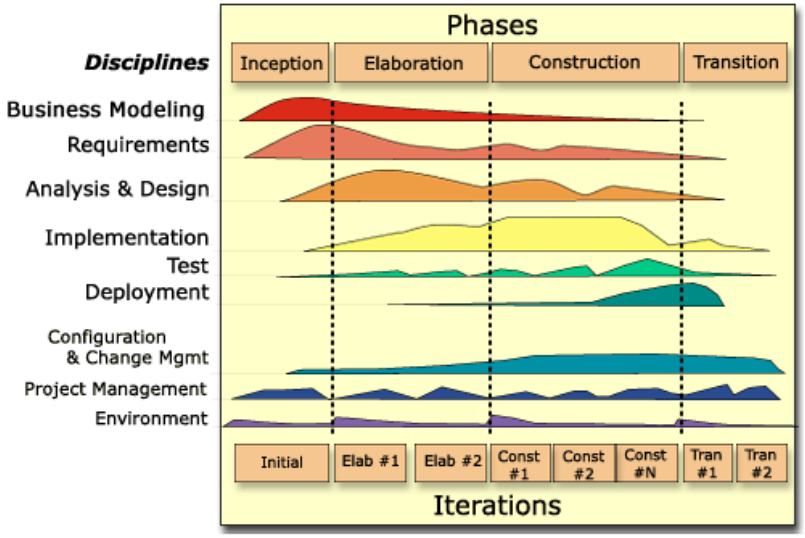
6 Engineering workflows:

- Business modelling
- Requirements
- Analysis and design
- Implementation
- Test
- Deployment

3 Supporting workflows:

- Configuration and change management
- Project management
- Environment (e.g. process and tools)

Workflows used in phases



(adapted from Rational slide)

UP best practices

Six fundamental best practices:

1. Develop software iteratively. Customer prioritisation, best first.
2. Manage requirements. Explicit documentation, analyze impact before adopting.
3. Use component-based architectures. Reuse, maintenance,...
4. Visually model software. UML...
5. Verify software quality. Testing...
6. Control change to software. Configuration management...

After a while, these should become automatic!

Personal Software Process I

Watts Humphrey, *A discipline for software engineering* p9:

"The following is the approach taken by the PSP:

- Identify those large-system software methods and practices that can be used by individuals.
- Define the subset of those methods and practices that can be applied while developing small programs.
- Structure those methods and practices so they can be gradually introduced.
- Provide exercises suitable for practising these methods in an educational setting."

Personal Software Process II

PSP provides a ladder of gradually more sophisticated practices.

Explicit phases of development, e.g. separate design from coding.

Lots of forms to fill in, e.g. time recording log, defect recording log.

Aim is to provide numerical data adequate for identifying weak areas and tracking improvements, in process and in own skills.

More info: `http://www.sei.cmu.edu/tsp/`

Tool support: `http://www.processdash.com/`

Where does PSP fit in?

PSP is a relatively high ceremony process, aimed at individuals and small projects. It's often used as a training process by people who expect to end up using a high ceremony process – such as UP – on large projects.

TSP, Team Software Process, is an intermediate.

Agile processes such as Extreme Programming take a very different approach – owing partly to deep philosophical differences, partly to different context assumptions. Next lecture.

A process maturity model such as CMMI (from SEI) can be used to help choose how to *improve* a software development process so as to fit the actual needs of the organisation.

Reading

Suggested: Browse the web to learn more about the processes mentioned:

- Waterfall
- Spiral
- (Rational) Unified Process
- Personal Software Process
- Capability Maturity Model