Computer Science Introductory Course MSC - Software engineering

Lecture 1: Software Management

Pablo Oliveira <pablo@sifflez.org>

ENST

13/10/2008

Outline

Lessons from the past

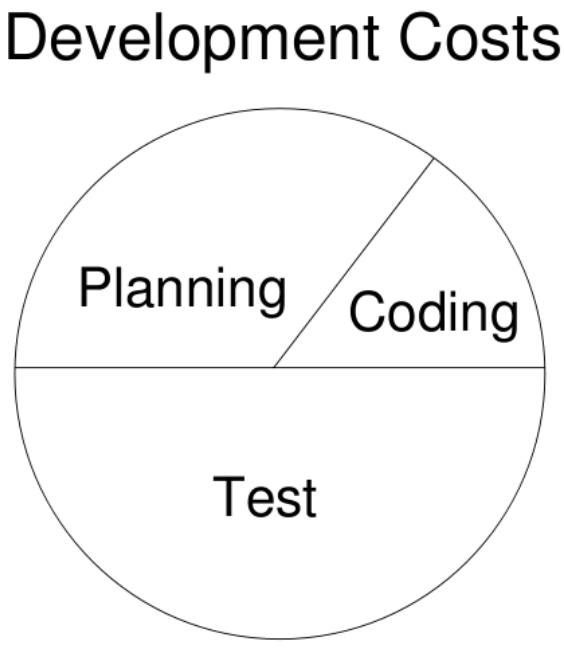
- 1996 Ariane-5 self-destructs, unhandled floating point exception, $500M lost.
- 1998 Mars Climate Orbiter is lost, navigation data expressed in imperial units, $327.6M lost.
- 1988-1994 FAA Advanced Automation System, project is abandoned, blame on management and over-ambitious specifications, $2.6B lost.
- 1985-1987 Therac-25 medical accelerator, a radiation therapy device malfunctions because of a race condition, 5 patients die, others are injured.

Common problems

- Amount of work is underestimated.
- Project specifications are vague.
- Lack of communication :
  - 'Communication overheads increase as the number of people increase (Brooks)
- Issues are not properly tracked.
- Teams botch the testing phase because of pressure from management.

Outline

Time distribution in a software project ?

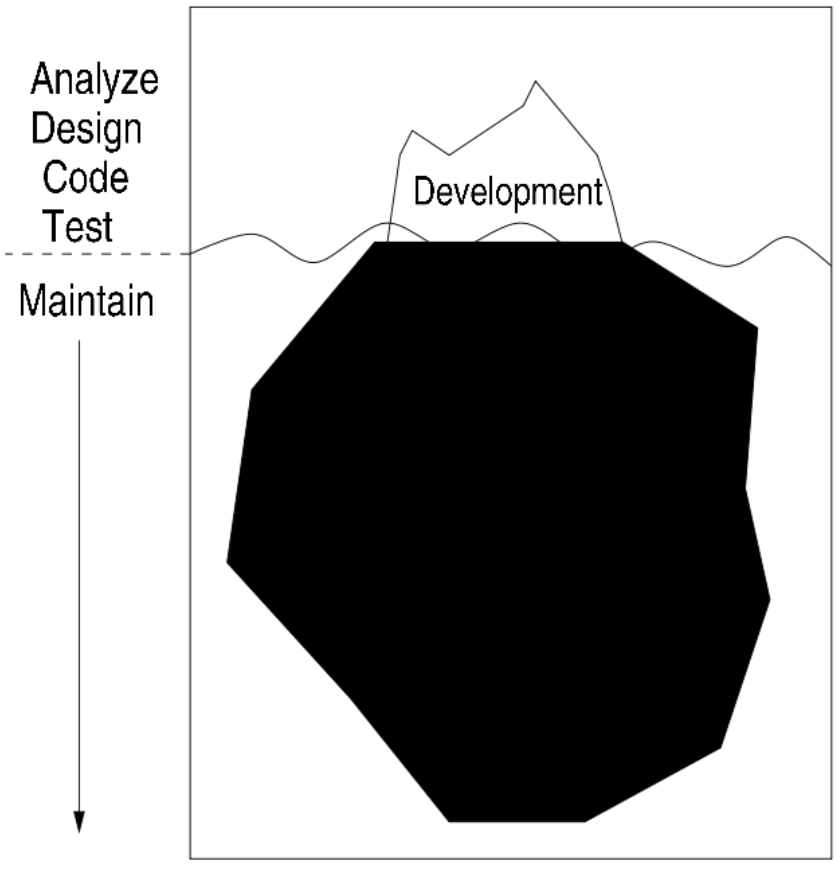


1/3 planning

1/6 coding

1/4 component test

1/4 system test

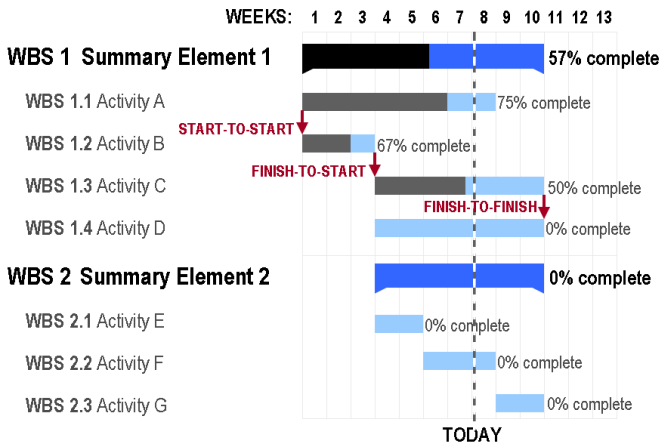


Development costs are only the tip of the iceberg.

(source : Nancy Leveson)

Planning

- Plan time carefully : “adding manpower to a late software project makes it later” (Brooks).
- You only control what you can measure : use metrics.
- Model dependencies and deadline, analyse risk.
- Keep track of deadlines and critical tasks, Gantt chart.



The phases of software development

- Analysis (Requirements capture and specification)
- Design
- Implementation
- Integration
- Testing
- Deployment
- Maintenance

Keeping track : Document !

Each software phase should be documented : each component life should be traceable

- Requirements $\longrightarrow$ use-cases, requirements formal document.
- Specifications $\longrightarrow$ specifications formal document.
- Code $\longrightarrow$ Comments / Revision Control System.
- Bugs $\longrightarrow$ Issues tracker / Regressions tests.
- User Documentation.

Outline

Requirements capture

- Objective : Understand the problem, so you can build the system the client needs instead of the system he thinks he needs.
- Hard because :
  - the client may have strong preconceptions about the system.
  - the client may be vague abouts its needs.
- Requirements specify : 'What' a system does and not 'How' it should be done.
- Requirements should be expressed in a language understandable by the client.
- Requirements should be traceable.

Requirements analysis

- Interviewing
  - Lots of work
  - Not necessarily precise
- User stories
  - Clients write down user stories
  - Use cases
  - Each user story has acceptance tests
- Straw-men
  - Sketch the product
  - Use anything ; napkins, storyboards, HTML, flowcharts
  - Anything to convey ideas without writing code !
- Rapid prototyping
  - Create one for client to validate
  - Major functionality, superficially implemented

(Source : Irfan Hamid)

Functional Requirements

The functions of a system : what should a system do ?

- mapping from input to output
- control sequencing
- timing of functions
- handling of exceptional situations
- formats of input and output data
- real world entities and relationships modeled by the system
- ...

(Source : Steve Easterbrook, University of Toronto)

Non-Functional Requirements

Constraints and quality goals

- interoperability
- portability
- availability
- safety
- ...

Requirements specifications

- At the end of the requirements gathering phase, the team must produce a specification document.
- The problem must be explained.
- Functional and Non-Functional requirements must be stated, and numbered.
- Some exemplary use cases that illustrate the product's functions should be given.

Requirements must be testable

An untestable requirement

The system shall be easy to use by experienced controllers and shall be organized in such a way that user errors are minimized.

A testable requirement

Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

(Source : Nancy Leveson)

Example of requirements specifications

- 3.3.4 Intute Repository Search response data
  - 3.3.4.1 Description The web service must provide responses to name query requests with a list of possible matches including the name authority record identifier (URI). The service may also need to return all other forms of an entity's name and affiliations for further disambiguation.
  - 3.3.4.2 Related requirements 3.3.1, 3.2, 3.2.1, 3.2.2.
  - 3.3.4.3 Source Introduced in Stakeholders' Requirements for the Names project prototype Intute Repository Search, Page 7.

(Source : Software Requirements Specification for the Names project prototype

by Daniel Needham, Amanda Hill, Alan Danskin & Stephen Andrews)

Outline

Specification

- An abstract description of the software that serves as a basis for (or describes) detailed design and implementation
- Describes how the requirements will be achieved.
- Primary readers will be software designers and implementers rather than users or management.
- Goals and constraints specified in requirements document should be traceable to the design specification (and from there to the code.

(Source : Nancy Leveson)

Views of Specifications

- Developer
  - Must be detailed enough to aid implementation
  - Must not be ambiguous
  - Must be traceable
- Client
  - Must be comprehensible
  - Must be readable by non-computer specialists
- Legal
  - A binding document.
  - Must contain acceptance (testable) criteria.

(Source : Adapted from Irfan Hamid course 2005)

Format of Specifications

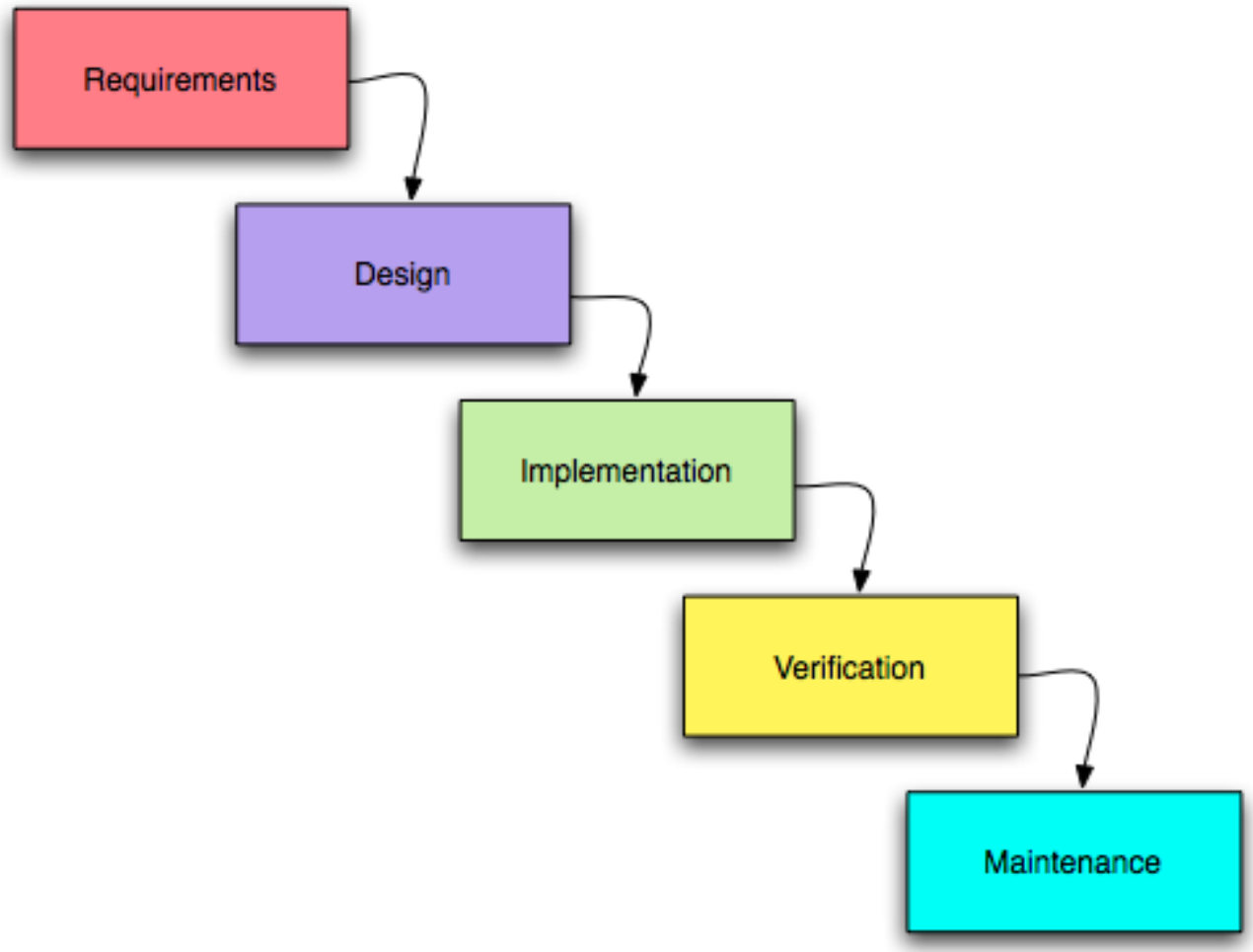
- Natural language (must be as unambiguous as possible)
- Semi-formal specifications (UML)
- Formal specifications (DFA, Z language, B language, math ...)

Example of specification using : Pre-conditions, Post-conditions, Invariants

```
class Dictionnary
    put (x: ELEMENT; key: STRING)
                require (pre-condition)
                  count <= capacity
                  not key.empty
                ensure  (post-condition)
                  has (x)
                  item (key) = x
                  count = old count + 1
invariant
                0 <= count
                count <= capacity
```

Outline

Waterfall (1/2)



(source : Paul Hoadley)

Waterfall (2/2) : pros & cons

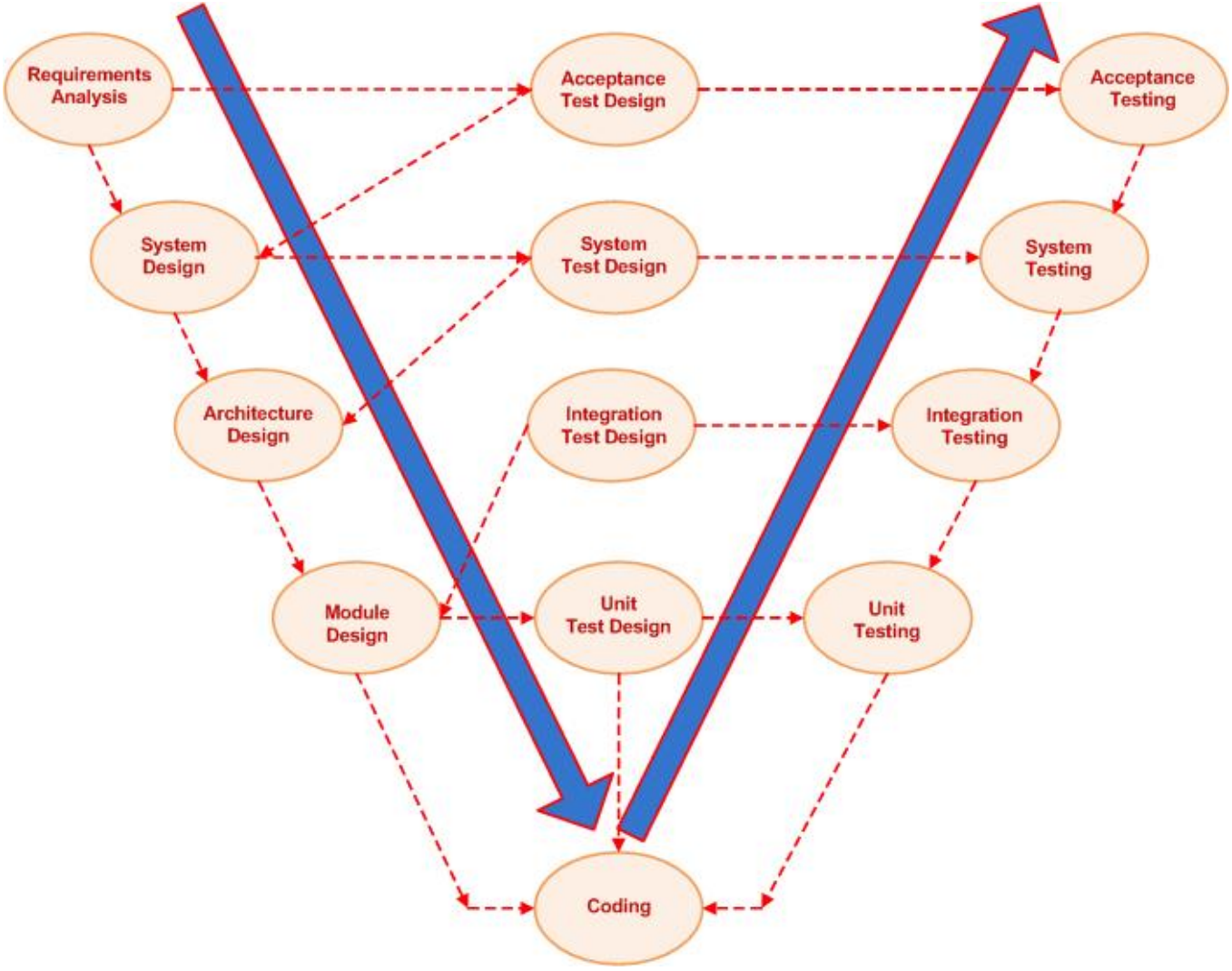
Pros

- Disciplined approach.
- Big Design Up Front.
- Document driven.

Cons

- Does not adapt to change :
  - Changing requirements.
  - Problems discovered during the implementation phase.

V Model (1/2)


Requirements Analysis
Acceptance Test Design
Acceptance Testing
System Design
System Test Design
System Testing
Architecture Design
Integration Test Design
Integration Testing
Module Design
Unit Test Design
Unit Testing
Coding

V Model (2/2)

- Extension of the waterfall model :
  - Takes in account the V&V and defines acceptance tests for each step.
- Better correspondence between design & tests.
- Makes V&V a central part of the process.

Evolutionnary

Plan to throw one away

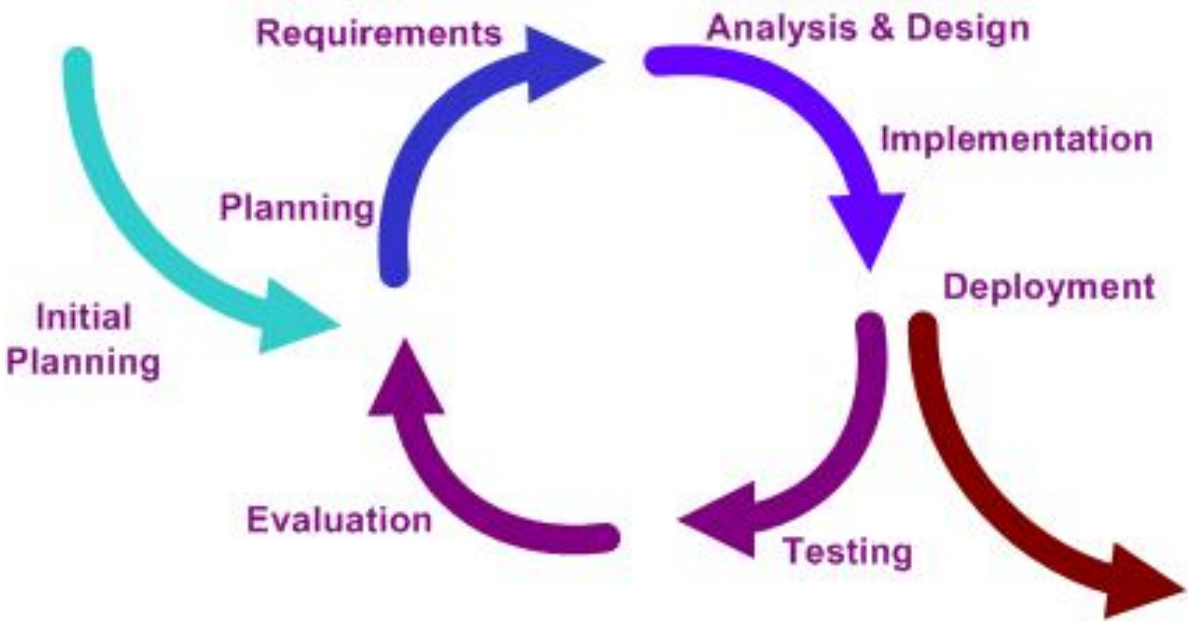
Pros

- Test feasibility.
- Check requirements.
- Discover problems early.
- Get user feedback early.

Cons

- Prototype gets a life of its own.
- Less though out designs.
- Less robustness.

Iterative (1/2)



Iterative (2/2) : pros & cons

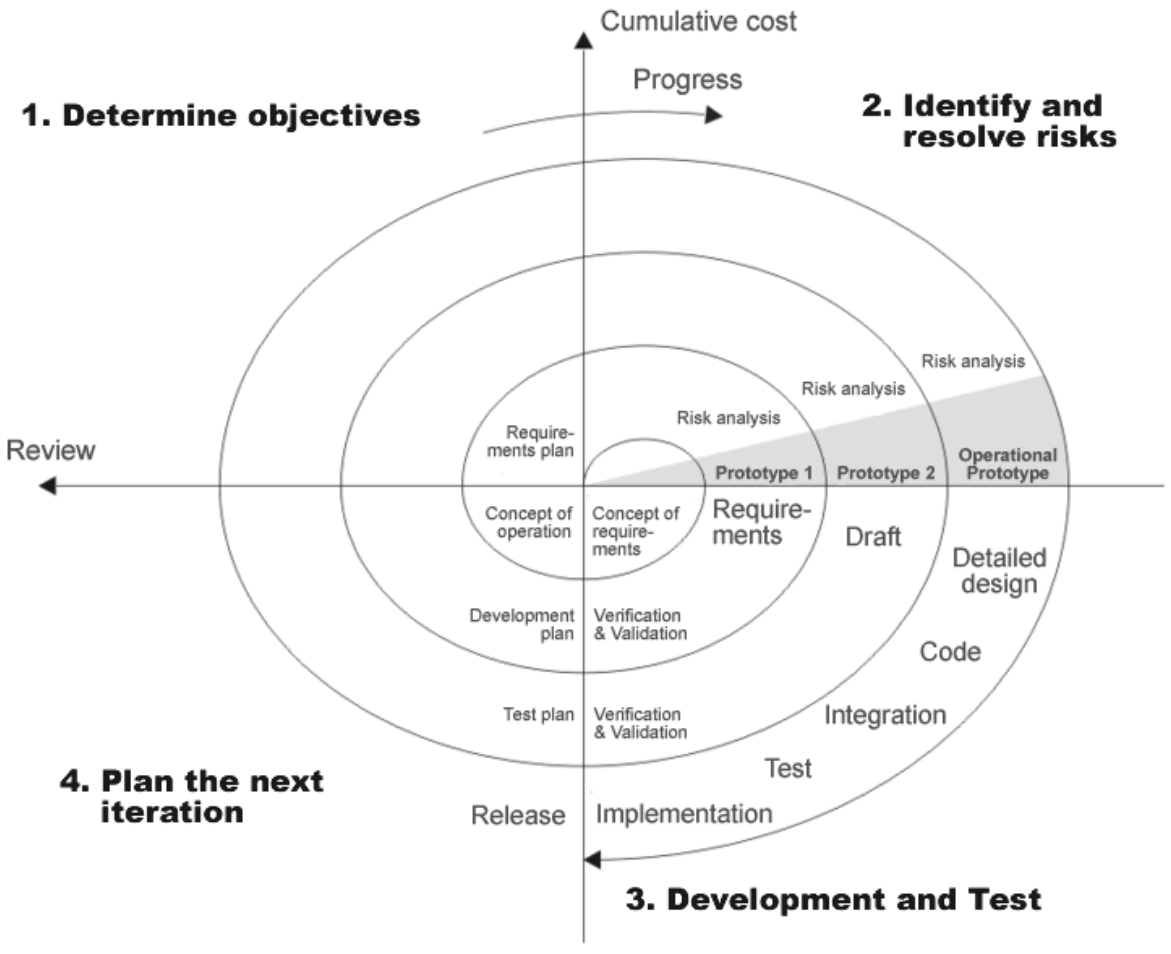
Pros

- Adapts well to change.
- Learning from the errors in the previous steps.
- Each step produces a finished product.

Cons

- Hard to recover from bad design choices in early steps.

Spiral(1/2)



(source : Conrad Nutschan)

Spiral(2/2) : pros & cons

Pros

- A form of iterative development.
- Tries to combines all the previous models.
- Risk based.

Cons

- Very costly for small projects.

Agile ?

- A buzzword that describes an emerging practice of software development.
- Encourages adaptation, inspection, communication, customer involvement, time-boxed development steps.
- Still very new, it is hard to evaluate the benefits from agile methods :
  - pair programming.
  - stand-up meetings.
  - time-boxed development steps.
- Yet it seems mainly effective with small teams of experienced developers.

Which is best ?

- SE is a young discipline, we lack perspective and objective studies to validate its methods.
- Depends on the project, the team size and competences, the enterprise culture.
- Act of faith.
- Choose something that works for you, but try to be rigorous and keep track of things.

Outline

Software development tools

- Tools will never replace team communication.
- Tools will never replace well though design.
- BUT...
- Tools can help in keeping track of code, bugs and issues.
- If used wisely, they can ease project documentation, management and communication.

Testing frameworks

- JUnit, ...
- Eases writing of unit, functional and regression tests.
- Allow automatic execution of the tests.

Version control system

- SVN(centralized), GIT(distributed), ...
- Keep track of changes in code.
- Each change is tagged with a commit message that explains which problem the code is going to solve.
- Developers have the complete history of a line code at their fingertips.
- When coupled with regression testings, can help finding the exact change that introduced a bug.

Issues tracker

- Roundup, Trac, etc ...
- Keep track of issues in a project.
- Allow easy bug reporting from users.
- Each issue is assigned a ticket which traces :
  - the discussion surrounding the issue.
  - the state of the issue.
  - the proposed patches to solve the issue.
  - the code commit that solves the issue.