Software Engineering Theory

What’s in a theory?
Software Engineering

- A systematic way of managing large programming projects

- A “long” historic evolution
  - From spaghetti programming
  - To Iterative development and RUP

- Many different ways
Small programs

- Simple code - simple structures
  - Assembler -> pseudo code

- Larger code chunks - spagetti structures
  - Requires better structuring
Start programming,
while I go and find out what the customers want!
Problems

- Complexity
- Errors
- Changeability
- Delays
Mistakes are expensive!

American Air Board
Upgraded traffic control system
Abandoned in 1994, no result
Cost: $2.6 billion

Sydney Water Corporation
Automatisation of customer information and billing.
Abandoned in 1992, no result
Cost: $33.2 million
Solution

- Planning
- Structured development
- Strict management of design process
- Strict management of programming
Waterfall Model

1. Requirement Specification
2. Analysis
3. Design
4. Implementation and module testing
5. Integration
6. System testing
7. Maintenance
V model

What?

Why?

How?

Testing

System

Subsystem

Module
Spiral model (Boehm)

Analysis

Design

Requirement specification

Implementation and module testing

Integration and system testing

Version 1  V.2  V.3

Continuous risk analysis

Maintenance
Incremental Development

Towards original goal

Usable versions
For the user

Towards revised goal

Revised goal

Functions set 1
Version 1

Functions set 2
Version 2

Functions set N
Final version

Original goal
Incremental Development

- Probably the "first" development method
- Requires structuring and planning (!)
- Small increments
- Stable system
- (aka. evolutionary development)
Prototyping

- Preliminary requirement specification
- Prototype
- Requirement specification/design
- Other model of development
A comment on prototyping

- Prototyping is not just making a prototype!
- Requirement analysis and specification still needed
- Project management and planning still needed
- Documentation essential
Agile development

- Rapid development cycle

- Suitable for projects with:
  - Loose specifications
  - Changing (dynamic) requirements

- Based on test specifications
  - Specifying testing before development
Iterative methods

- Needs to be controlled!
  - Extra functions should be screened!
  -

- High demands on documentation
  - Undocumented changes are dangerous

- High need for flexible structure
  - Iteration adds change
Project planning

Step 1. Define the goals
- What are the goals?
- What is to be delivered?

Step 2. Planning
- Which tasks?
- Who does what?

Step 3. Test the plan
- Everything considered?
- Does it work together?

Step 4. Risk analysis
- Weaknesses?
- What can be done about them?

Step 5. Define control points
- Minimize the risk of "surprises"
Problems in Software Engineering

- Optimistic time schedule
  - Quick fixes
  - Bad planning

- Changing needs
  - Changed circumstances -> different product

- Bad project planning

- New requirements are added

- New innovations are added on the fly
Requirement specification

- Detailed description of the external characteristics of the final product
- Functional benchmarks
- Compatibility
- User interface
- Installation/Maintenance
- Verification measures and delivery goals (milestones)
- Short descriptions of desired functions or problems to be solved
- End user expectation
Focus on Technology

- System solutions
- Beautiful representations
- Difficult to use
- Unnatural tool
- Data decides on functions
Focus on Users

- Task solutions
- Presentation more important than representation
- Suitable for context of usage
- "Natural" tool
- Usage decides on functions