Introduction II

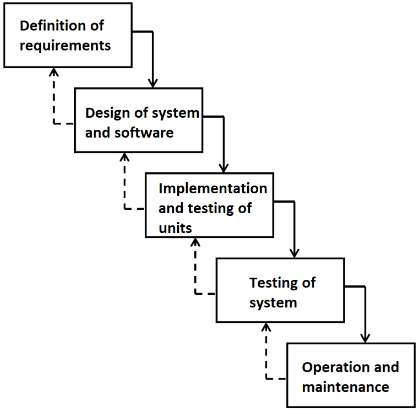
PA017 SW Engineering II $\rightarrow$ Aspects of SW Development Management

Jaroslav Ráček **Josef Spurný**

Faculty of Informatics, Masaryk University

September 27, 2022

Waterfall Lifecycle


Definition of requirements
Design of system and software
Implementation and testing of units
Testing of system
Operation and maintenance

- Integration and testing of the system at the same time
- Problems at later stages have great impact on price

Waterfall Lifecycle

Problems

- Real project do not follow predefined order of steps
- Users may not fully and clearly describe requirements at early stages of project
- Customer has to be patient
- Late discovery of issues may seriously jeopardize the whole project

Managers prefer this model.

Visibility of Waterfall Lifecycle

Activity	Output Document
Requirements Analysis	Feasibility Study General Requirements
Requirements Specification	Catalogue of Requirements
System Specification	Functional Specification Tests Plan User Manual Design
Architecture Design	Architecture Specification System Test Plan
Interface Design	Interface Specification Integration Tests Plan

Visibility of Waterfall Lifecycle

Activity	Output Document
Detailed Design	Units Specification Unit Tests Plan
Implementation	Source Code
Unit Testing	Unit Testing Protocol
Module Testing	Module Testing Protocol
Integration Testing	Integration Testing Protocol Final User Manual
System Testing	System Testing Protocol
Acceptance Testing	Final System and its Documentation

Implementation based on General Requirements

Very talented individuals are required

- Average team cannot be used for this type of development. Successful products were developed by small teams of highly-talented members

Systems are usually improperly structured

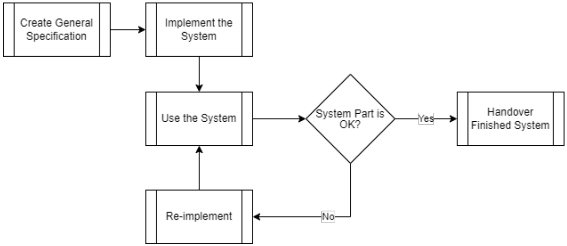
- Repeated changes damage system structure. Evolution is difficult and costly.

Process is invisible

- Manager needs regular outcomes to steer the process. For fast implementation, it is not efficient to produce documentation reflecting each version.

Incremental Lifecycle

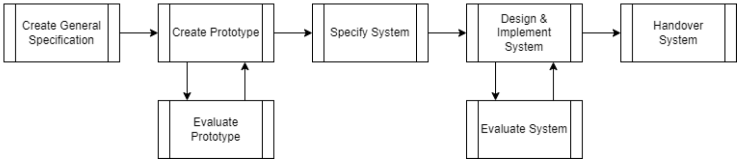
Development based on General Requirements



Problems

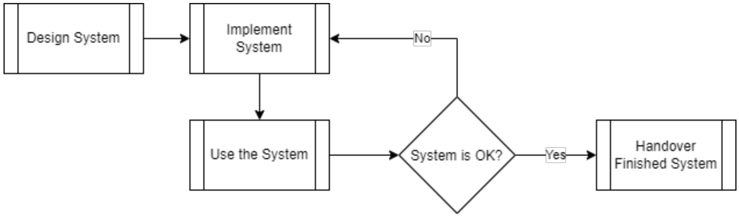
- General Requirements vs. Reality
- Program Documentation vs. Specification
- Maintenance increases entropy

Prototyping Lifecycle



- Suitable for smaller projects when general requirements are not clear
- People like to criticize
- Therefore, prototypes are used to collect knowledge
- Recommended 1-2 prototype iterations
- Prototypes are (mostly) discarded after specification – focus on low cost

Researcher Lifecycle



Problems

- Challenging to manage
- Trial-and-error approach
- Non-existent or invalid documentation
- Team members cannot be replaced

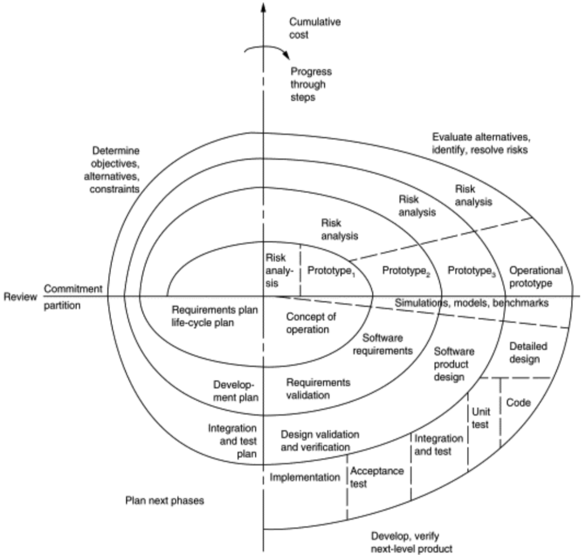
Experimenting without predictable outcome.

Researcher Lifecycle

For each project, a suitable lifecycle has to be chosen.

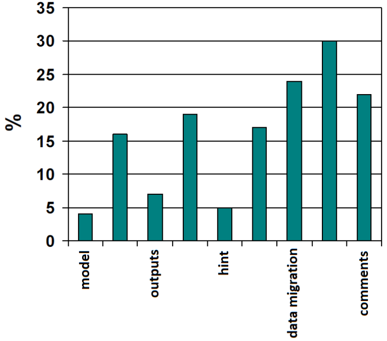
Is Researcher Lifecycle suitable for critical infrastructure?

Spiral Lifecycle (Boehm,1988)



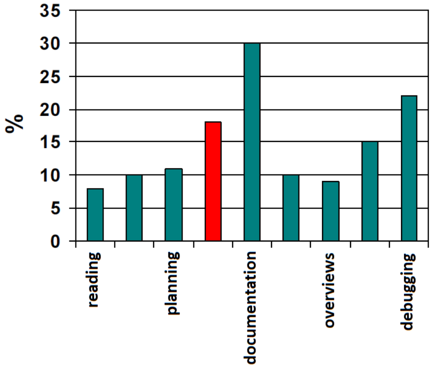
Composition of Application

- Model Calculations
- User Inputs
- User Outputs
- Management
- Hints / Help
- Errors Processing
- Internal Data Migration
- Data Declaration
- Comments



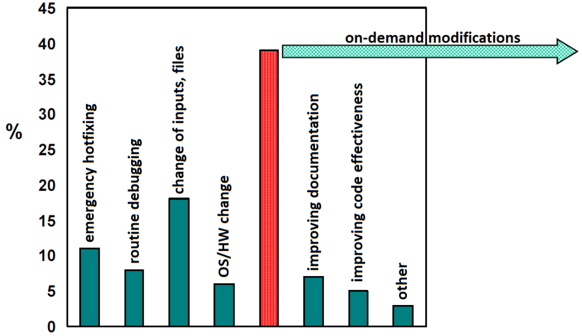
Typical Programmers' Activities

- Reading knowledge base
- Designing app, components, documentation
- Planning approach, tasks, time
- Programming
- Documentation
- Testing
- Overviews
- Meetings
- Debugging

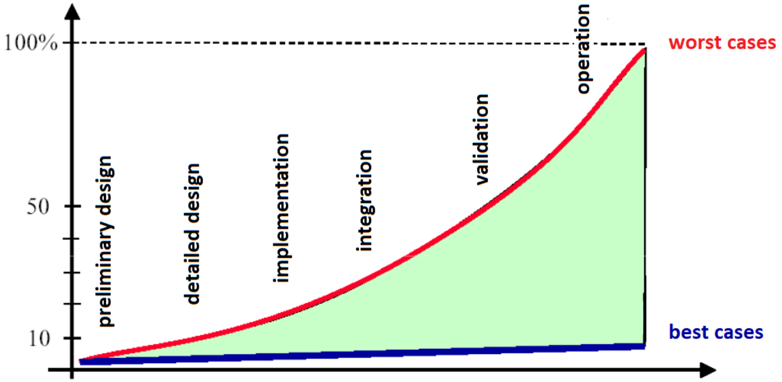


Maintenance

Maintenance is software product modification after handover to the Customer with the aim of removing errors, improving performance, or adaptation to the changing environment.



Relative Cost of Maintenance

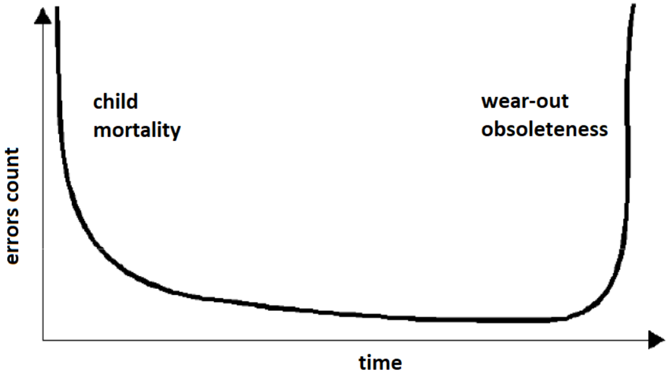


If you think-through the structure of the future SW well and design it properly, the cost of implementation will be slightly higher, but the cost increased this way will return in the SW operation phase when maintenance is provided and customer on-demand modifications are delivered

Hypotheses about Errors

- The later the phase in which an error is detected, the more costly is the cost of fixing it
- Many errors remain hidden and will be revealed only after the phase in which the error was made has ended
- There are many errors in the requirements
- Errors in requirements mostly consists of wrong assumptions, forgotten facts, conflicting or ambiguous information
- Errors in requirements can be detected

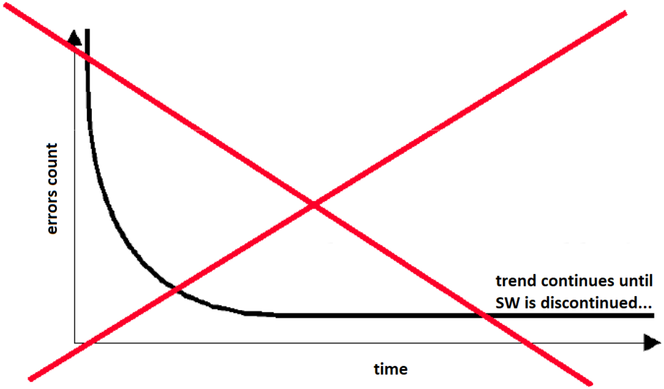
Errors and HW Wear



- At the beginning, the HW performance is sufficient, but there are errors that can be fixed over time
- At the end, the wearing begins to manifest, and the HW is falling behind its more powerful surroundings. It is time to replace it

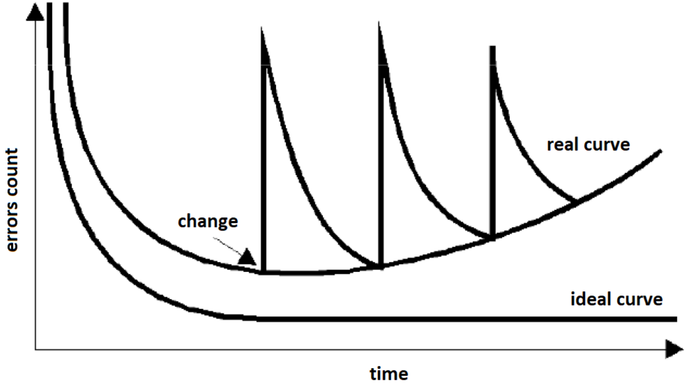
Errors and SW Wear

It is naive to think that, as time passes by, you will remove all SW errors and all troubles will disappear :-)



Errors and SW Wear

In reality, the Customer asks for new modifications repeatedly during operation. This makes the SW more complex and introduces new errors. One day, it will be better to develop the system from scratch (Lehman’s Law).



Lehman Laws (1974)

Law of Continuing Change

System used in real environment is continuously changing, until it becomes cheaper to re-structure the system, or to completely replace it by a newer version.

Law of Increasing Complexity

During evolutionary changes, the programs becomes increasingly less structured and internal complexity becomes higher. Removing increased complexity requires additional effort.

Law of Self-regulation

The pace of global system attributes change may appear random over time. From long-term perspective, it is a self-regulated process which can be statistically described and predicted.

Lehman Laws (1978)

Law of Invariant Work Rate

The overall advancement in development is statistically invariant. In other words, the development pace is approx. constant and does not correlate with invested resources.

Law of Conservation of Familiarity

Users must update their familiarity with the system to efficiently handle it. Fast growth hinders the handling mastery. As a consequence, average increment growth remains invariant as the system evolves.

Programming in Team

LOC = Lines of Code - program size
E = Effort - time in months
PP = Programmer's Productivity
GPP = Group Programmers' Productivity

$PP = \frac{LOC}{E}$ (lines of code per month)
N programmers $\rightarrow \frac{N(N-1)}{2} \approx N^2$ interactions
λN^2 – effort per communication (each communicates with everyone else)
$GPP = \frac{LOC}{(E+\lambda N^2)}$ - group productivity
$\frac{GPP}{PP} = \frac{E}{(E+\lambda N^2)}$

Brooks' Law

Adding a team member to delayed project may cause an increased delay

Costs for inclusion of a new team member are usually higher than his/her benefit.

MUNI
FACULTY
OF INFORMATICS