Software Engineering 1
Introduction

Arkadiusz Chrobot

Department of Computer Science, Kielce University of Technology

Kielce, October 6, 2025

Outline

Bibliography

📕 Ian Sommerville,
Software Engineering,
Pearson Higher Education Inc., London, UK, 2015, on-line:
https://iansommerville.com.

📕 Pierre Bourque, Richard E. Fairley,
SWEBOK v3.0 Guide to the Software Engineering Body of Knowledge
IEEE Computer Society, USA, 2014.

📕 Sungdeok Cha, Richard N. Taylor, Kyochul Kang,
Handbook of Software Engineering,
Springer Nature Switzerland AG, Cham, 2019.

📕 Gerald O'Regan
Concise Guide to Software Engineering
Springer International Publishing AG, Cham, Switzerland, 2017.
...and many other will be presented in slides and on the website!

Motto

"If you look at software today, through the lens of the history of engineering, it's certainly engineering of a sort — but it's the kind of engineering that people without the concept of the arch did. Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousand of slaves."
Alan Kay

Introduction

IEEE 610.2 Definition

"Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software, and the study of such approaches."

Software

Software — computer programs and other artefacts created in the software development process.

David Farley "Working" Definition of Engineering

Engineering is the application of an empirical, scientific approach to finding efficient and economical solutions to practical problems.

📖 David Farley, *Modern Software Engineering: Doing What Works to Build Better Software Faster*, Pearson Higher Education Inc., London, UK, 2022.

Software Crisis

In the 60ies of last century the third generation of computer systems had been introduced to the market. That opened the possibility of developing complex software, but it soon had become apparent that it is very difficult. Nobody had known how to successfully build complex, efficient and reliable software. The majority of software projects ended with a disaster. The situation had become known as the *software crisis*. To overcome the problem in 1968 the NATO organized a conference in Garmisch, Germany. There the concept of software engineering was defined.

Results

The principles of developing software successfully are still unknown. The overall situation in the software industry hasn't improved significantly:

- Joel Mathis, „Starliner: What went wrong?" (2024),
- Gregory Travis, „How The Boeing 737 MAX Disaster Looks To a Software Developer (2024),
- Simon Sharwood, „CrowdStrike file update bricks Windows machines around the world" (2024),
- *The Risks Digest*,
- The Standish Group Chaos Report: "19% failed, 50% challenged, 31% successful" (2020).

Causes

Some of the causes:

- software projects are usually innovative,

Sources:

🌐 Terence Parr
Why Writing Software Is Not Like Engineering
http://www.cs.usfca.edu/~parrt/doc/software-not-engineering.html

🌐 Chuck Allison
Code Quality
[This source is no longer available]

Causes

Some of the causes:

- software projects are usually innovative,
- the product of software engineering is not physical,

Sources:

🌐 Terence Parr
Why Writing Software Is Not Like Engineering
http://www.cs.usfca.edu/~parrt/doc/software-not-engineering.html

🌐 Chuck Allison
Code Quality
[This source is no longer available]

Causes

Some of the causes:

- software projects are usually innovative,
- the product of software engineering is not physical,
- requirements in software projects change a lot,

Sources:

Terence Parr
Why Writing Software Is Not Like Engineering
http://www.cs.usfca.edu/~parrt/doc/software-not-engineering.html

Chuck Allison
Code Quality
[This source is no longer available]

Causes

Some of the causes:

- software projects are usually innovative,
- the product of software engineering is not physical,
- requirements in software projects change a lot,
- bad project management,

Sources:

🌐 Terence Parr
Why Writing Software Is Not Like Engineering
http://www.cs.usfca.edu/~parrt/doc/software-not-engineering.html

🌐 Chuck Allison
Code Quality
[This source is no longer available]

Causes

Some of the causes:

- software projects are usually innovative,
- the product of software engineering is not physical,
- requirements in software projects change a lot,
- bad project management,
- "the exponents war".

Sources:

Terence Parr
Why Writing Software Is Not Like Engineering
http://www.cs.usfca.edu/~parrt/doc/software-not-engineering.html

Chuck Allison
Code Quality
[This source is no longer available]

Computer Science vs. Software Engineering

Computer Science

How to create effective software? (algorithms, data structures, computational complexity, programming languages, programming paradigms).

Software Engineering

How to effectively create software? (project management, methods of copping with project complexity, software architecture, documentation, production costs, testing, reliability, maintenance).

Features of Software

Some features of high quality software:

- maintainability,

Features of Software

Some features of high quality software:

- maintainability,
- reliability,

Features of Software

Some features of high quality software:

- maintainability,
- reliability,
- effectiveness,

Features of Software

Some features of high quality software:

- maintainability,
- reliability,
- effectiveness,
- usability.

Programming Paradigms

Some of the main programming paradigms:

1. imperative
   1. procedural/structural (Pascal, C, Perl),
   2. object-oriented (C++, Java),
2. declarative
   1. functional programming (Erlang, Elixir, LISP, JavaScript),
   2. logic (Prolog).

Software Development Paradigms

There are several models of developing software. Some of them are:

- waterfall,
- iterative development,
- formal methods,
- integration and configuration,
- agile methods.

Software Development

The software development is a process of creating a product which is, basically, a computer program (or computer programs). The course of this process depends on the type of the created software, but in each case four main phases can be found:

1. specification — easier for *generic software* and much harder in case of *custom software*,

2. development,

3. validation,

4. evolution.

Model of the Software Development Process

A model of the process (or a paradigm) is a simplified description of the process from a particular perspective. The models can be classified as follows:

- workflow model,
- data flow model,
- role activity model.

Waterfall Model

The Waterfall Paradigm is the oldest Software Development Life Cycle model applied in Software Engineering, proposed in 1970 by dr. Winston W. Royce. It was adopted from other engineering disciplines. In this model there are five main activities that are performed only once in the whole process: *requirements analysis and definition*, *system and software design*, *implementation and unit testing*, *integration and system testing* and *operation and maintenance*. This model organizes the work on software, but any change in requirements generates a lot of costs — the process has to start again from the beginning. It is appropriate for software projects that are a part of a larger engineering projects, like for example space probe programs.
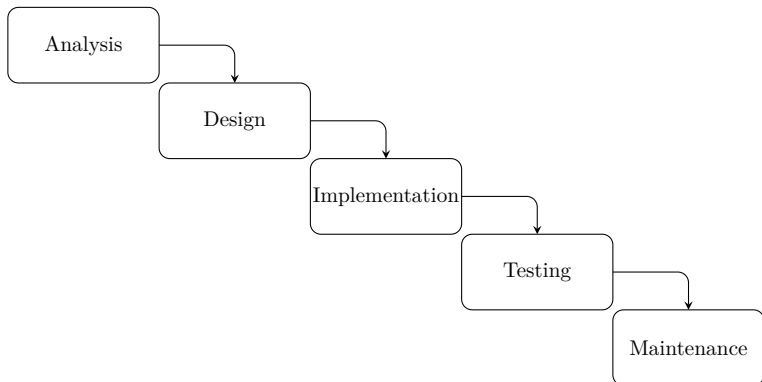
Waterfall Model

Figure: Simplified Waterfall Model
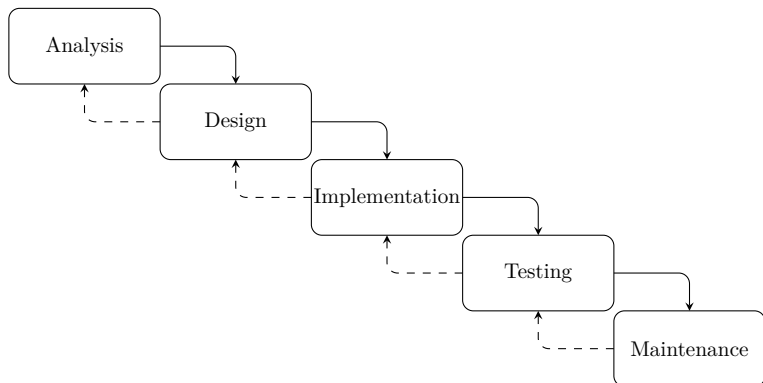
Waterfall Model

Figure: Simplified Waterfall Model

Iterative Development

In the Iterative Development the product (software) is build gradually. First an initial version based on early specification is build and delivered to users. After getting feedback from the users the development team improves the product and then sends it back to them for further evaluation. The process is repeated until a required system is build. The activities that are common for all development models are interleaved in the Iterative Development rather than separate. This approach to software building can be combined with prototyping. The prototype can be crated as an experimental implementation of new functionalities and then incorporated to the main product or abandoned. In the latter case the prototype is build only to refine or discover requirements. The Iterative Development is suitable for projects in which requirements frequently change (in fact it is true for most of the software projects). The resulting product better responds to customer needs, but it internal structure may not be optimal. The progress of development may be hard to follow.

Formal Methods

Formal Methods incorporate mathematical techniques into the software development process. Requirements for the product are written in a formal, strict specification language. The software is *derived* from its specification and its correctness is verified with the use of mathematical proofs. The resulting product is reliable and of high quality. Unfortunately, not any type of software can be developed like this. Moreover, the development is not always cost-effective. Nowadays Formal Methods are usually used for safety-critical products.

Further readings:

📕 Gerald O'Regan
Concise Guide to Formal Methods
Springer International Publishing AG, Cham, Switzerland, 2017

Formal Methods

"Ten years ago, researchers into formal methods (and I was the most mistaken among them) predicted that the programming world would embrace with gratitude every assistance promised by formalisation to solve the problems of reliability that arise when programs get large and more safety-critical. Programs have now got very large and very critical — well beyond the scale which can be comfortably tackled by formal methods. There have been many problems and failures, but these have nearly always been attributable to inadequate analysis of requirements or inadequate management control. It has turned out that the world just does not suffer significantly from the kind of problem that our research was originally intended to solve."

Tony Hoare in 1995

Integration and Configuration

The main idea behind this approach to software development is to reuse components that have been created in previous projects, bought or are available as open-source software. The development can be fast and cheap, but usually some compromises regarding the software functionality have to be made. It means that the resulting product may not exactly be what the customer needs.

Agile Methods

In 2001 several software professionals created and signed the Agile Manifesto:

"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."*

Source:

🌐 https://agilemanifesto.org/

The Manifesto is accompanied by Twelve Principles of Agile Software.

Agile Methods

The Agile Manifesto and Agile Principles led to the creation of many software development *methodologies* that are know as Agile Methods. Also some already existing methods that fulfill the requirements of the Agile Manifesto and Agile Principles were classified as Agile. Among them are: Lean Software Development, PDCA (Plan — Do — Check — Act), Kanban, Extreme Programming (XP), Feature-Driven Development, Dynamic System Development Method, Crystal, Scrum. Those methods can be wholly or partially combined with others, also those "non-agile". The main goal of Agile Methods is to provide the customer with a working product and reduce the amount of any additional work in the project. The Agile Methods seem to be suitable for small projects with small teams and with vague requirements. They do not scale well for bigger projects.

Beyond Agile

Recently the Agile Methods are supplemented with three other development models:

CI/CD Continuous Integration/Continuous Delivery — Continuous Integration is when software engineers are developing the software in small increments that are frequently, even several times a day, integrated, build and tested in an automated fashion. If the software is also deployed or prepared for deployment at the end of the process than this is Continues Delivery.

Beyond Agile

Recently the Agile Methods are supplemented with three other development models:

CI/CD   Continuous Integration/Continuous Delivery — Continuous Integration is when software engineers are developing the software in small increments that are frequently, even several times a day, integrated, build and tested in an automated fashion. If the software is also deployed or prepared for deployment at the end of the process than this is Continues Delivery.

DevOps   DevOps integrates software development and operations, making it easier for the software engineers to get feedback on the current state and issues related to their software after it was deployed. It requires even more automation that the CI/CD, regarding the deployment, monitoring and configuration of the software. DevOps is especially useful in Cloud Computing Environments.

Beyond Agile

Recently the Agile Methods are supplemented with three other development models:

CI/CD Continuous Integration/Continuous Delivery — Continuous Integration is when software engineers are developing the software in small increments that are frequently, even several times a day, integrated, build and tested in an automated fashion. If the software is also deployed or prepared for deployment at the end of the process than this is Continues Delivery.

DevOps DevOps integrates software development and operations, making it easier for the software engineers to get feedback on the current state and issues related to their software after it was deployed. It requires even more automation that the CI/CD, regarding the deployment, monitoring and configuration of the software. DevOps is especially useful in Cloud Computing Environments.

DevSecOps DevSecOps is the same as DevOps, but with security concerns taken into account. It means cooperation with security experts during development and maintenance of the software and automation of processes related to the software security.

Software Engineering Methods and Tools

The Software Engineering not only defines the models of the software development process but also provides means (called Software Engineering Methods) and tools that help to follow them. The methods allow software engineers to build models of computer programs that form the specification of software. The UML (Unified Modeling Language), Object-Oriented or Structural Analysis are examples of such means.

The tools originally were named Computer-Aided Software Engineering or CASE for short. Nowadays this term is rarely used. They are used for requirements analysis and designing (so-called Upper CASE tools) or implementation and testing (Lower CASE). Contemporary examples are GitLab, GitHub, JUnit.

Software Development Costs

The overall costs of developing software depend on its type and the kind of the development process applied. Generally, the most expensive phase is the verification (testing), which takes up to 40% of the project budget and in case of safety-critical software it can even reach 50% of total costs. But more expensive than the development is the maintenance of the software (its modification after the deployment).

Challenges

The most important challenges of the modern software engineering are:

- legacy software,
- diversity of systems,
- delivery times,
- software quality (security, reliability, etc.),
- software impact on society.

Summary

The computer software becomes more and more complex. Also more and more aspects of human life depends on it. We (as humanity) do not know exactly how to develop reliable, efficient software. There are many failures in the short history of Software Engineering (recent example: MCAS software in Boeing 737-MAX 8), but also many successes (for example: the Voyager space probes). The recent Chaos Report (2020) suggests that the software in the future will be developed in pipelined fashion, similar to DevOps/DevSecOps. It seems that the Software Engineering is finally heading in the right direction. The conclusion is that adhering to the principles of Software Engineering doesn't alone guarantee the success of a software project, but neglecting them is the recipe for assured failure.

For entertainment:

🌐 Aaron Cummings,
Uptime 15,364 days - The Computers of Voyager,
https://www.youtube.com/watch?v=H62hZJVqs2o

Questions

?

THE END

Thank You for Your attention!