Software and Software Engineering

Software Design
2DV608

Mauro Caporuscio
Department of Computer Science and Media Technology
mauro.caporuscio@lnu.se

The "Simple" Goal of Software Development

Construct software that creates *Value*

- **Functionality**
- of certain **Quality**

Why is it so difficult?

The Nature of Software

Software is intangible

- Hard to understand development **effort**

Software is easy to reproduce

- Cost is in its **development**
  - in other engineering products, **manufacturing** is the costly stage

The industry is labor-intensive

- Hard to **automate**

The Nature of Software

- Untrained people can hack something together
  - Quality **problems** are hard to notice

- Software is easy to modify
  - People make **changes** without fully understanding it

- Software doesn't 'wear out'
  - It **deteriorates** by having its design changed:
    - erroneously, or
    - in ways that were not anticipated, thus making it complex

The Nature of Software

- Demand for software is **high** and **rising**

- Much software has **poor design** and is getting worse

- We are in a perpetual **software crisis**
  - Some software are never delivered
  - Some are delivered but never put to use
  - Many are delivered late and over budget
  - Almost all **require modification** before they can be used

Types of Software

- *Custom*
  - For a **specific** customer
  - Typically developed **in-house** within the same organization that uses it
- *Generic*
  - Sold on **open market**
  - Often called **COTS** (Commercial Off The Shelf)
- *Embedded*
  - Built into **hardware**
  - **Hard** to change

Types of Software

- *Real time software*
  - Must react **immediately**
  - **Safety** often a concern
- *Data processing software*
  - Used to run **businesses**
  - **Accuracy** and **security** of data are key
- **Some software has both aspects**

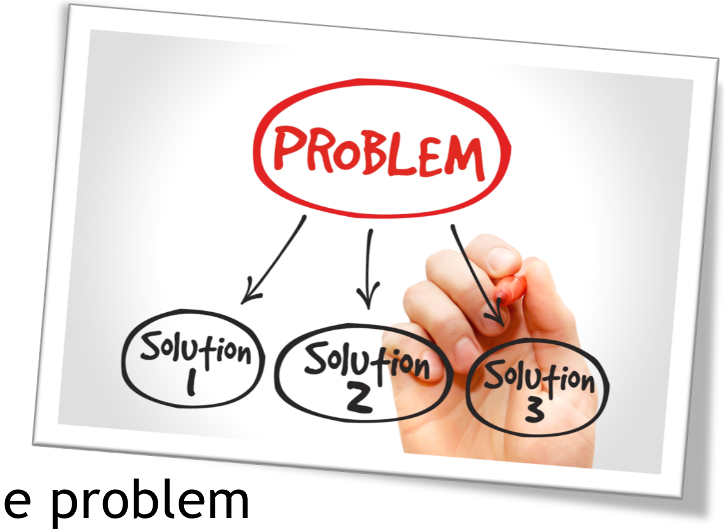
We have to learn to engineer software

What is Software Engineering?

- The process of ***solving customers' problems*** by the ***systematic development*** **and *evolution*** of ***large***, ***high-quality software systems*** within ***cost***, ***time and other constraints***

- Other definitions:
  - *IEEE*: the application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance of software; that is, the application of engineering to software.
  - *The Canadian Standards Association*: The systematic activities involved in the design, implementation and testing of software to optimize its production and support.

Solving customers' problems



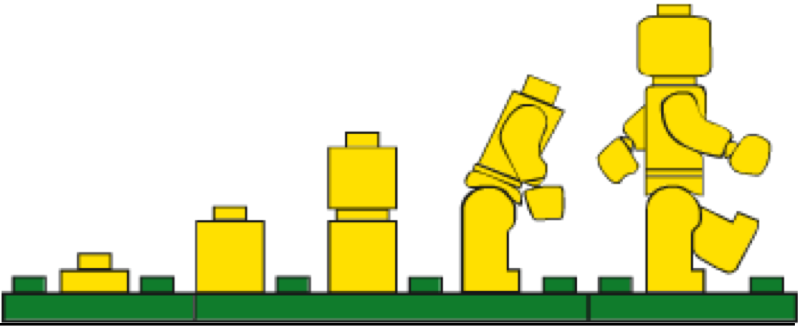
- This is the **goal** of software engineering
- Sometimes the solution is to **buy**, **not build**
- Adding **unnecessary features** does not help solve the problem
- Collective effort
  - Software engineers must **communicate effectively** to identify and understand the problem

Systematic development and evolution

- An engineering process involves applying **well understood techniques** in a organized and **disciplined** way

- Many well-accepted practices have been formally standardized
  - e.g., by the IEEE or ISO

- Most development work is **evolution**



Large, high quality software systems

- Software engineering techniques are needed because large systems **cannot be completely understood** by one person

- **Teamwork** and co-ordination are required

- Key challenge
  - **Dividing up the work** and ensuring that the parts of the system **work properly** together

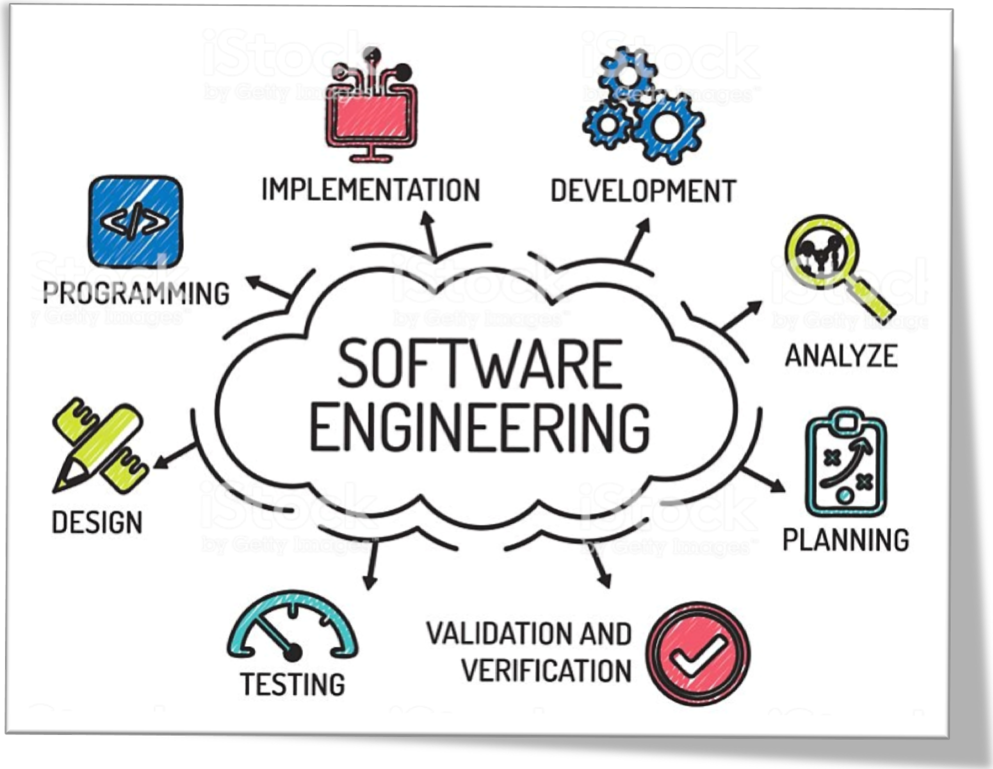
- The end-product must be of **sufficient quality**

Cost, time and other constraints

- **Finite** resources
- The **benefit** must outweigh the cost
- **Competitors** do the job cheaper and faster
- Inaccurate **estimates** of cost and time have caused many project failures

Software Engineer

Software engineers **design** artifacts following best practices



Stakeholders in Software Engineering

1. Users
   - Those who **use** the software

1. Customers
   - Those who **pay** for the software

2. Software developers

3. Development Managers

All four roles can be fulfilled by the **same person**

Beauty is in the eyes of the beholder

How the customer explained it

Like beauty, quality is in the eye of the beholder

Customer:
solves problems at
an acceptable cost in
terms of money paid and
resources used

User:
easy to learn;
efficient to use;
helps get work done

Developer:
easy to design;
easy to maintain;
easy to reuse its parts

Development manager:
sells more and
pleases customers
while costing less
to develop and maintain

Software Quality: Attributes

- Usability
  - Users can **learn it fast** and get their job done easily
- Efficiency
  - It doesn't **waste resources** such as CPU time and memory
- Reliability
  - It does what it is required to do without **failing**
- Maintainability
  - It can be easily **changed**
- Reusability
  - Its parts can be **reused** in other projects, so reprogramming is not needed

Software Quality: Conflicts and Objectives

- The different qualities can **conflict**
  - Increasing efficiency can reduce maintainability or reusability
  - Increasing usability can reduce efficiency

- Setting **objectives for quality** is a key engineering activity
  - You then **design to meet the objectives**
  - Avoids ‘over-engineering’ which wastes money

- **Optimizing** is also sometimes necessary
  - E.g. obtain the highest possible reliability using a fixed budget

External Quality Criteria

The system as observed from the **user's** point-of-view

- Examples
  - Functionality
  - Usability
  - Efficiency
  - Reliability

Internal Quality Criteria

The system as observed from the **developer's** point-of-view

- Characterize **aspects of the design** of the software

- Have an effect on the external quality attributes

- Examples
  - Maintainability
  - Reusability

Short Term Vs. Long Term Quality

- Short term:
  - Does the software **meet the customer's immediate needs**?
  - Is it sufficiently efficient for the volume of data we have **today**?

- Long term:
  - Maintainability
  - Customer's **future** needs
  - Scalability: Can the software handle larger volumes of data?

Software Engineering Projects

Most projects are **evolutionary** or **maintenance** projects, involving work on **legacy** systems

- Corrective projects: fixing defects
- Adaptive projects: changing the system in response to changes in
  - Operating system
  - Database
  - Rules and regulations
- Enhancement projects: adding new features for users
- Re-engineering or perfective projects: changing the system internally so to improve it

Activities Common to Software Projects

Requirements and specification

- *Domain analysis*
  - Understanding the **background**
- *Defining the problem*
  - Narrowing down the **scope** of the system
- *Requirements gathering*
  - Obtaining **input** from as many sources as possible
- *Requirements analysis*
  - **Organizing** the information
- *Requirements specification*
  - Writing detailed instructions about how the software should **behave**

Activities Common to Software Projects

Design: How the requirements should be implemented

- *Systems engineering*
  - Deciding what should be in **hardware** and what in **software**
- *Software architecture*
  - Dividing the system into **subsystems** and deciding how the subsystems will **interact**
- *Detailed design*
  - Deciding how to **construct** the details of each subsystem
- *User interface design*
  - Deciding in detail how the user is to **interact** with the system
- *Design of databases*
  - Deciding how the **data** will be stored

Activities Common to Software Projects

Modeling

- Creating **representations** of the domain or the software
  - Use case modeling
  - Structural modeling
  - Dynamic and behavioral modeling
- Performed by means of semi-formal notations (e.g., UML)

Programming

Quality assurance

- Encompasses all the processes needed to **ensure** that the quality objectives are met

Deployment

- **Distributing** and **installing** the software and any other components of the system