

What is Object-Oriented Modelling?



This section explains the reasons for modelling software and explains what is so special about object-orientation.

First Things First

Welcome to the **Object-Oriented Modelling and UML** course. The objective is to provide an informative and interactive learning environment that allows you to acquire the principles of Object-Oriented Modelling Techniques and UML using Gentleware's Poseidon.

The Idea behind Modelling

Modelling is a concept that has been around for years. Any project of significant size is usually planned using some sort of model. This can be an architect's blueprint of a building, a solicitor's text template for a legal document or an engineer's drawing of a car. The main **advantages** of a modelling stage preceding implementation are the **better planning, reduction of risk & cost** and **enhanced communication** between a client and contractor.

The same holds for the development of software. It increases communication among all stakeholders, guarantees better planning of the system to be built and therefore reduces risk and cost. In addition to the generic advantages, modelling has additional benefits in a software engineering environment:

- Modelling helps to find and reveal abstractions
- Modelling provides a platform independent means to express software designs
- Modelling provides a means to express and plan software architectures

The importance of software architectures has been expressed in the following quote by the Carnegie Mellon Software Engineering Institute:

„Software architecture forms the backbone for building successful software systems. An architecture largely permits or precludes a system's quality attributes such as performance or reliability. Architecture represents a capitalized investment, an abstract reusable model that can be transferred from one system to the next. The right architecture is the linchpin for software project success. The wrong one is a recipe for disaster.“

Roles

There is a wide range of roles in any software development process which will make use of modelling facilities.

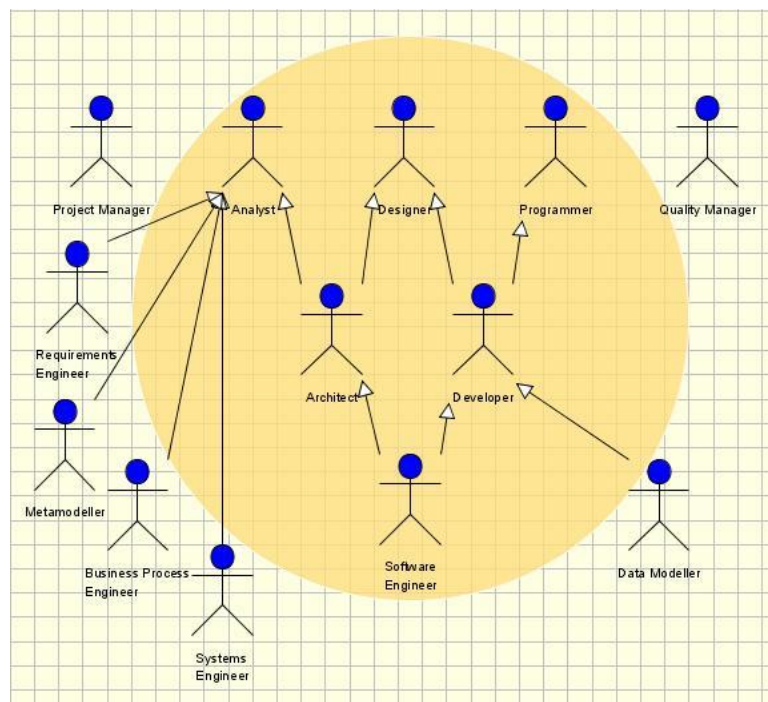


Figure 1: Roles of Software Engineering

There exists a wide range of software engineering roles in a project. On smaller projects certain roles are taken on by a single person. For example, a software architect might be covering the roles of the analyst and the designer.

Different roles are involved at different stages in the software engineering life-cycle and therefore make use of different (object-oriented) modelling functionality. As a consequence, different roles will make use of different UML functionality.

What's the Story about OO?

There have been different paradigms of software development. Approximately every ten years a new approach of how to develop software has been adopted. In principle, the newer style improved productivity, reduced risks, increased quality and allowed building systems with increased complexity.

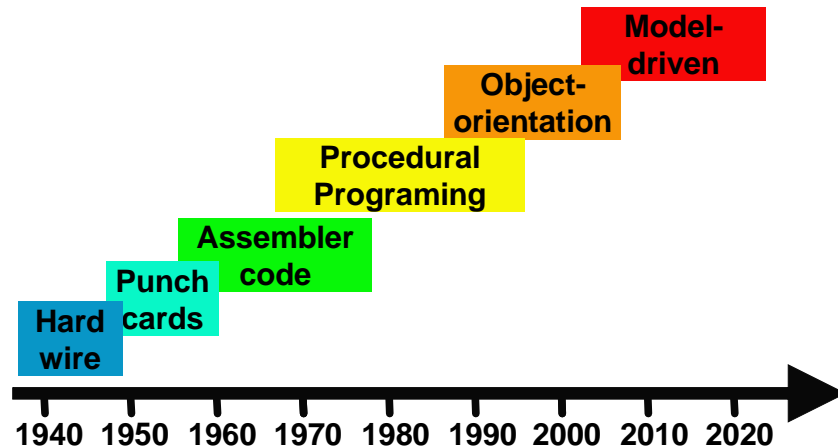


Figure 2: Paradigms of Software Development

The very first systems were **hard wired** circuit boards performing very simple computations, compared to today's standards. **Punch cards** followed allowing the storage of programs. **Assembler code** represented machine readable instructions which were computed a low level, close to processor and memory.

Procedural programming has dominated the software engineering scene for almost two decades. As the name suggests, it is based on the concept of the procedure call. Procedures, also known as routines or functions, contain a series of computational steps to be carried out. Any given procedure can be called at any point during a program's execution. The benefits are as follows:

- The ability to re-use the same code at different places in the program without copying it.
- An easier way to keep track of program flow than a collection of "GOTO" statements (which turn programs into so-called "spaghetti code").
- The ability to be strongly modular or structured.

However, there exist a range of drawbacks, which have been overcome by **object-orientation**:

- Reusability of code is hard
- Building of large systems in a team requires a large amount of communication and is prone to errors
- System maintenance is hard since elements are not independent
- Procedural systems are hard to understand for non-technical people

The idea behind object-orientation is that a system is seen as a collection of individual units, or **objects**, that act on each other, as opposed to a traditional view in which a program may be seen as a collection of procedures. Each object is capable of receiving messages, processing data and sending messages to other objects.

The next generation of software engineering paradigm will be model-driven. The approach focuses first on *what* is being achieved in a computing system and then on *how* it is being achieved. It's the responsibility of a software subsystem to translate the *what* to *how* using code generation. However, these systems are immature and it will take some more time until these systems will become mainstream.



We have shed some light behind the objectives of software modelling and have shown the advantages of object-orientation.