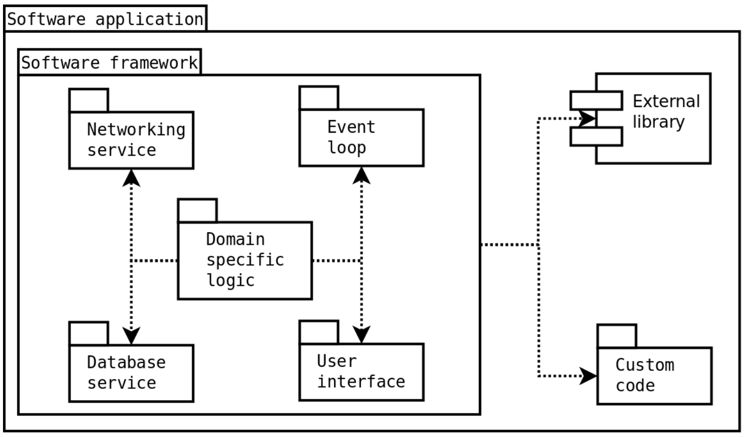

Figure 10: The framework architecture of a program.

3.2 Software Engineering

The term *software engineering* was introduced into public circulation for the first time at conferences organized by NATO in 1968 and 1969. The software crisis, that has been previously mentioned, was discussed. The term was introduced to clearly suggest that software developers should structure and manage a process of software development similar to others industrial sectors. It was a suggestion to move away from an art and get closer to the craft (business) of programming. An engineering approach to software development needs to reduce the costs of software development and lead to software that is more reliable [29]. Today, software engineering (SE) is a scientific discipline that deals with the development, deployment and maintenance of computer software. It is a part of computer science. SE promotes the methodical, disciplined, and quantifiable approach to the development of software, i.e. a well defined development process consisting of traceable actions and tasks with predictable size. Software engineers have to take into consideration a lot of circumstances, e.g. the type of hardware the software will be used on, operation systems by which the software will be handled, and also the specificity of the company in which the software will be deployed or the qualification level of the end users who will operate the software.

The primary objective of SE is the production of software systems in accordance with a requirement specification; it should be done on time, and within budget. One can say that software engineering is an instance of systems engineering (an interdisciplinary field). However, carrying out its own tasks, software engineering utilizes its own development processes and design methods for software.

3.2.1 Software Development Process

Software development teams follow a specific theoretical life cycle in order to guarantee the best quality of created software product. This life cycle was build around the detailed study of the experiences of many software teams. It assumes similar knowledge from general engineering as well. Normally, the *life cycle of software* is comprised of the following phases:

1. Requirements gathering, analysis and specification
2. Software design
3. Implementation (coding)
4. Testing (validation) and integration
5. Deployment (installation)
6. Maintenance
7. Retirement (withdrawal)

Each phase produces deliverables required by the next phase in the life cycle, e.g. requirements are translated into software design, code is produced in accordance with blue prints, testing verifies the code taking into account the requirements, and so on. Even the withdrawal from circulation of old software generates useful information for the requirements and design phases of the next version of software.

In the first phase, business requirements are gathered. This means interviewing a stakeholder about the software that is planned and its business goals. The answers to the following questions will be significant: who is going to use the software, how will they use the software, what data will be inputted into the programs, what data should be outputted, and which action or data should be restricted? These answers constitute requirements. They are analyzed for their validity, cohesion and the possibility of realization in the planned software. The analysis ends with a document called software *requirements specification* (SRS). This document is an essential part of the contract for the software and should clearly determine “what the software should be.”

The second phase is about blue prints. At the time, the software design is arranged in accordance with the requirement specification. One of the most important tasks is to determine the architecture of the planned of software. The designer should describe a high-level software architecture that outlines the functions, relationships, and interfaces for major components. This means that the designer should decide on modularity of software, on the possible need to use libraries, frameworks, design patterns, and the ready to use components. Based on these decisions, the third phase will be organized – coding (implementation). This is the longest period of the software development life cycle. Software design documents (blue prints) are guidelines for programmers in the process of writing code. Coding is done in collaboration with designers, because planned modules or other units of code may vary with respect to those documented in the preceding phase.

In the fourth phase, ready-made units of code are integrated (combined one with another so that they become a whole) and tested against the requirements. Testers have to make sure that the product is really solving the needs formulated by stakeholders. At a further stage other tests are performed to ensure a high quality of product. After successful testing, the software can be delivered to the customers to be used. This fifths stage is not a simple as it may seem. A customer's hardware and infrastructure should be ready to implement the new software. The customer's personnel should be trained in the use of the new software. The customer's data and archives should be available for the new software. After resolving all the problems, software developers often have to adapt product to local requirements.

In the sixth phase, the customer operates the installed software. During the period when the customer is using the software, problems may arise and they need to be solved (occasionally very quickly, e.g. when they relate to information security). Developers solve such problems and update the product. This activity is called maintenance. With time there are so many fixes (patches) in the software that the developers decide to release a new optimized version of software. This is a software upgrade. The old version is withdrawn, and the new version is deployed. Sometimes, this means the complete withdrawal of software from circulation for business reasons, e.g. the production of a given software ending.

The modern software development process tends to run iteratively through some of the seven phases rather than linearly. This is the most common flow of work: detailed design, code construction, unit testing, integration, complete software testing and release. Iterative production of software allows for managing software projects of high complexity effectively. Successive iterations can be planned based on the priorities identified by the stakeholders in relation to the functions of the software. The most important functions should be scheduled for the first releases, and the less important – for later. The necessity of such planning usually results from the limited resources of software teams.

In implementing an operational business process methodology, software developers use different approaches to the organization of the development process. To visualize and then study the development processes, several models have been elaborated. These models and corresponding approaches are referred to as *software development process models*. The best-known classical approach is represented by the waterfall model (Herbert D. Benington, 1956). This was often used for the creation of large software systems; metaphorically, it looks like a cascade waterfalls. This is why some people call it a cascade model. This model describes a linear process in which the activities resulting from the software life cycle are executed one by one. The model contains the ability to go back to one of the preceding activities when conditions make this necessary or desirable. The main negative aspect of the waterfall model is the trouble of making convenient changes when the process is ongoing. One can say that this process is inflexible.

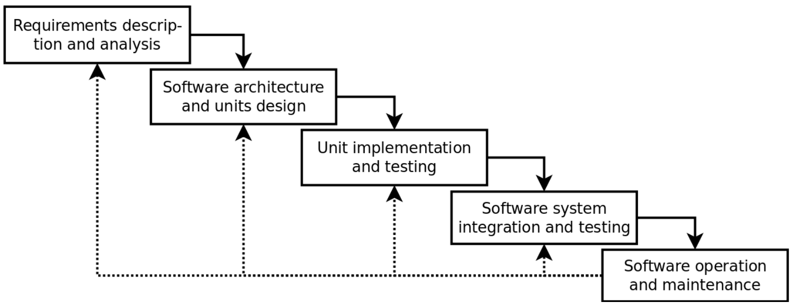
Requirements description and analysis
Software architecture and units design
Unit implementation and testing
Software system integration and testing
Software operation and maintenance

Figure 11: The waterfall model of software development process.

The idea of a more flexible process shows another classical model of software development process – the spiral model (Barry Boehm, 1986). Here, the process is represented as a set of loops – each loop leads the team to the final product, and the current loop execution phase constitutes an indicator of the progress of the stage. This model does not establish fixed phases of the life cycle, for instance requirements, design and implementation. Loops are selected depending on what is required. The spiral model provides an explicit assessment of the risk of failure, so managers can respond quickly to changes in the project. One can say that it represents the process driven by risk management.

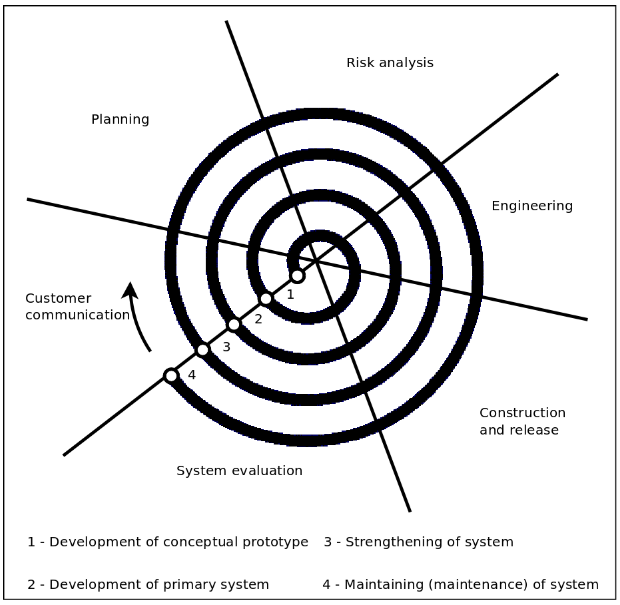
Risk analysis
Planning
Engineering
Customer communication
1
2
3
4
Construction and release
System evaluation
1 - Development of conceptual prototype
3 - Strengthening of system
2 - Development of primary system
4 - Maintaining (maintenance) of system

Figure 12: The spiral model of software development process.

Both of these models now have a generally historical significance. New experiences and new knowledge have helped expand the idea of software production. The most influential idea has turned out to be agile software development. This was implemented in the methods called *Scrum* (Jeff Sutherland and Ken Schwaber, 1995) or *extreme programming* (Ken Beck, 1999) and others (which fall between them conceptually). An agile paradigm has changed priorities in software development. More attention should be paid to people and interactions than to procedures and tools, to functioning software than to complete documentation, to customer collaboration that to contract negotiation, to responding to changes than to following the plan. All agile methods take seriously active user involvement in software development. Customer representatives are regular participants in all activities of the project team. Development works through small increments of functionality (frequent releases with completed features). This allows the evolution of requirements and timescale fixing simultaneously.

An important attribute of agile methods is that software teams are empowered to make decisions. One can say that agile methods actively use the *human factor*. In this context, the aforementioned Scrum is an agile development method, which uses the concept of a team-based development environment. It emphasizes team self-management, which in conjunction with experiential feedback allows building the correctly tested product gains within short intervals. At the same time, the necessary role of product owner in Scrum team ensures the representation of the client's interests. In contrast to Scrum, extreme programming (XP) is a more radical agile method. It concentrates more on the development and test phases of the software engineering process. XP teams use a simplified structure of planning and monitoring activities to choose what should be done next and to forecast project finalization. An XP team usually delivers very small increments of functionality. It is focused on continuous code improvement based on user involvement in the development process. An unusual feature of this method is so-called pair-wise programming. Two programmers, sitting side by side, at the same machine, create production software (one can say this is continuous reviewing of code).

In conclusion, software development processes combines all activities associated with the production and evaluation of software. To understand the development process abstract representation, a model of this process, which shows the structure (organization) of typical activities (e.g. specification, design, testing, implementation and installation) is used. There is no possibility to closely classify all processes and their models, because they were created in a spontaneous manner, in the context of competition. To understand the essence of software engineering, one can look at some historical examples (the best example is the Rational Unified Process). A good idea might be to analyze several international standards on this issue, such as ISO/IEC 12207 "Systems and software engineering – Software life cycle processes" or ISO/IEC 15504 "Information technology – Process assessment".

The Rational Unified Process
Rational Unified Process was developed in the second half of the 1990s. This new model of the software engineering process was distributed as a native key component of professional software (development kit, set of tools) targeted at software developers. It was a very important event in the history of object-based software systems. Instead of providing a large number of paper documents, the Unified Process concentrates on the development and maintenance of semantically rich models [30], which must represent the software system being developed. Rational software tools delivers full support for such models, using the Universal Modeling Language (UML). The modeling of large information systems with the use of this language is described in the next chapter. RUP is not a constant process. Developers can fit it to the scale of the project and adjust it to customer requirements. The Rational Unified Process has adopted the best-known practices of modern software development and assists developers in implementation of these practices.

RUP is an iterative development process. This means that a whole software project is divided into several mini projects (iterations) planned one by one. It helps to recognize the changing requirements and to organize early risk attenuation. After each iteration, developers can correct errors in software and be more accurate with their plans. Each RUP iteration increments the software functionality. This means that the software is evolving to be the result of cumulative effort. Such an evolutionary approach to developing software gives developers a chance to deal with the reliability, stability, and usability of the product.

RUP's standard sets out four phases of the project. The first phase, called inception, refers to capturing the initial requirements, to analyzing initial risks and cost benefits, and to defining the project scope. At this stage, developers arrange an initial architecture of the software system. Then they build a prototype which is not reusable and begin creating the key models – a use case model and a domain model. The first model is used to analyze the required functionality of the software, while the second one – to analyze the problem itself with its surroundings. The results of this analysis will represent the problem in the design of software. One can say that this phase deals most with business modeling and project management, not with the design of software.

The second phase, called elaboration, continues capturing and analyzing the requirements, modeling use cases and problem domain. During this phase, scenarios of user activities are developed, a glossary model of the project is created and prototypes of user interfaces are fleshed out. It is important that at this early stage, even before the meticulous, detailed design, the customer (users) will be able to do some work by using the real views (graphical user interfaces) of the future application. As a result of such interactions with trial software, serious semantic errors are usually detected that were overlooked during requirement gathering. One can say that this phase is concentrated on requirements and on design of software.

The third phase, called construction, focuses on implementation of the design and on testing the created software units. The application features, remaining after the elaboration phase, are developed and integrated into the complete product. At this time, the software architecture is subjected to continuous validation; the same applies to data repositories. All planned user activities and all sequences of object interactions are modeled and validated as well. It is a typical manufacturing process with resource management and operation monitoring, which must optimize costs, schedules, and quality of software development. This phase begins a conversion from intellectual property (created in the previous phases) into the deployable product.

The fourth phase, called transition, relies on the relocation of the software to the users' groups. It includes making software available for users (e.g. via websites), and in some cases the installation, training of end users, and technical support. Once users have begun to use software, problems begin arise. The software team is required correcting mistakes and preparing new releases. At the same time, developers have to work out features that were postponed.

A static perspective of the Rational Unified Process shows development process activities, artifacts and workers in context of workflows. This perspective has to describe the behavior and responsibilities of team members using the template "who is doing what, how, and when". The same team member can be planned as a few workers (to play several roles), e.g. as a designer he or she may deal with object design and as a use-case author he or she may detail a use-cases. During the development process various artifacts are produced – standalone pieces of information like source code or architecture documents; workers produce or modify or use them performing their activities. Finally, a workflow is a meaningful sequence of those activities, which produces a result (a value like a document or program unit). RUP suggests nine core process workflows, which arrange all workers and activities. These workflows are divided into six core engineering workflows (business modeling, requirements, analysis and design, implementation, testing, and deployment) and three core supporting workflows (project management, configuration and change management, and environment).

3.2.2 Software Design Methods

Software design is a core (major) phase of the life cycle of software and at the same time, it is the most "mysterious" process of converting the requirement specification into an executable software system. The practice of software teams is not as spectacular as it may seem. Here, design means developing a blueprint (a plan) for a procedure and mechanisms that will perform the required tasks. It is different from software programming – design should be made before programming and programming (or in other words coding) is the realization of design. Software design is the most critical phase affecting the quality of the created software. Developers use particular design