Interreg
EUROPEAN UNION
Austria-Czech Republic
European Regional Development Fund

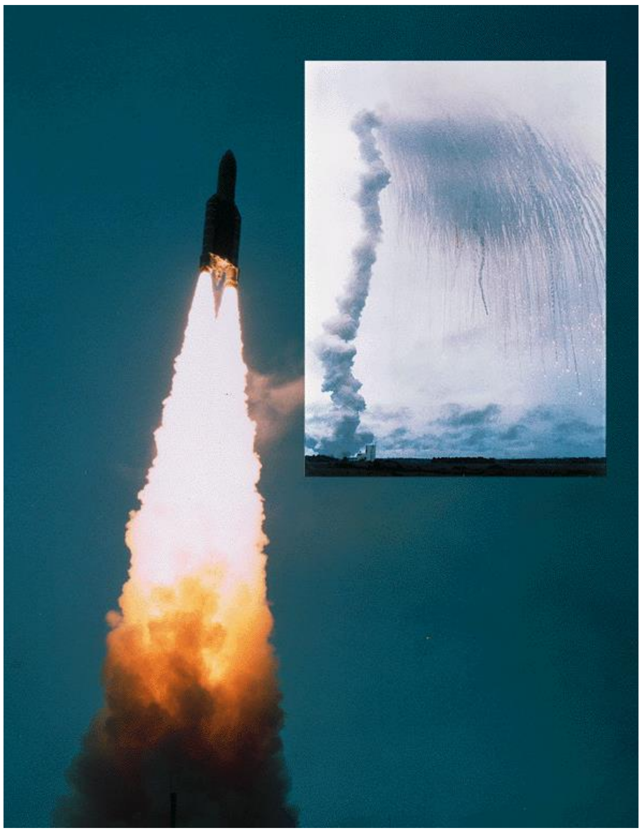
Software engineering and modeling

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA
VŠTE
EUROPEAN UNION

Contents

1. INTRODUCTION TO SOFTWARE ENGINEERING

Software error may cause either an application crash in a phone, or a really big problem. Some of the adverse consequences of faulty software are suggested in Richta's lecture (2011):



Fall of Ariane-5 rocket: As a result of an error, the main computer issued an order to deflect nozzles of solid rocket boosters and engine nozzles. Thus the rocket course was dramatically changed. In addition, given to aerodynamic forces, the upper part of the rocket broke off. Subsequently, the self-destruct system of the rocket was activated and the rocket turned into a cloud of burning shards. Total sum: 100 Mill. $ (including Cluster satellites, which were parts of the rocket, 500 Mill. $).

The North-east blackout of 2003 in the USA: It was caused by an unregistered blackout of the automated fault detector in the power station at the Niagara River. The blackout gradually spread via the network. The blackout decommissioned 21 power stations altogether. About 50 Mill. people were affected; three people died; damage amounted to 6 Bill. $

The collision of the landing module on Mars (1999): There was a problem of communication between components - the interface user expected a value in kilometres; however, the provider reported it in miles. Instead of planned 140-150 kilometres above the surface, it landed 57 kilometres above the surface. In such an altitude, Mars atmosphere is too heavy for a probe. Climate Orbiter probably burned in the altitude of 80 kilometres. (Source: Technet.cz)

1.1. Reasons for software engineering development

At the beginning of computer development in 1940s and 1950s, computer computing power was very low. In time, this computing power was developed by geometric series according to well-established Moore’s law. Software development was still in its infancy. No methods or techniques which would facilitate the development were devised. In 1960s, when computing power significantly developed again, so called software crisis arose.

The distinctive features of the software crisis were unsustainable delay and price-increase of projects, poor program quality, maintenance and innovation difficulties, low efficiency of programmers, development ineffectiveness, poor results etc.

Main causes of the crisis are as follows:

- Poor communication between people involved in software development and between developers and customers.
- Wrong approach. The particular software was developed and approved by a developer and a dissatisfied customer was referred to as a person who does not understand the matter.
- Bad planning of the project. Developers hoped to eventually meet the deadline in a way.
- Unrealistic estimates of the duration of development, costs and scope.
- Low work efficiency. Programmers paid attention to unimportant things and ignored the essentials.
- Ignorance of basic rules. E.g. Brook’s law from 1975: “Adding a research capacity to a delayed project will only increase its delay.”
- Underestimation of threats and risks. The threats were not regarded; instead of being prevented, they caused big troubles.
- Bad technology application. False idea that all problems will be solved after introducing a new technology.

As a response to the above-mentioned issues, a conference which would introduce basic principles of software engineering took place in 1969.

At the time, it was defined as follows:

“Software engineering is a discipline which deals with establishing and applying key engineering principles to software development in order for us to achieve economic software development which is reliable and works efficiently on available computing devices.”

Software engineering can be referred to as an engineering discipline dealing with actual problems of development of complex software systems. This discipline involves several complex thought processes. This engineering discipline follows pragmatic approaches. These approaches include knowledge of special requirements for the software product, its analysis, design, implementation, testing and installation of the system to the end user. At the same time, they involve managerial approach to the project management, which enables effective use of individual techniques whose main goal is to effectively devise a high-quality software product.

1.2. Psychological excursion

Cognitive psychology explained different ways in problem solving between experts and novices. Experts are able to recall partial solutions of similar situations and apply the previous knowledge to the identification and definition of the problem (Eysenck and Keane, 2008). Experts deal with the problem by exploring the heart of the matter and all external circumstances. Based on this information, they try to find a solution. Novices, on the other hand, try to guess a solution and afterwards assess whether it is possible to work out the suggested solution from relevant facts. They thereby use the backward shift strategy (Nolen Hoeksema at al., 2012). According to Plháková, the biggest difference between novices and experts lies in the extent of knowledge organization (Plháková, 2003). Highly developed self-regulating abilities (explored in a separate chapter) comprise characteristics of successful individuals (experts). These abilities allow effective use of knowledge and skills (Helus & Pavelková, 1992), In fact, laymen knowledge to understand problems is usually very superficial. On the other hand, experts first approach the heart of the problem and try to identify and organize it. They do not approach the solution until they have a feasible plan. Furthermore, experts focus on preparation much longer that laymen, but find the optimal solution to the problem much faster (Plháková, 2003). Říčan (2016) says that experts: "Not only do they have greater knowledge (quantitative aspect), but also differ in a qualitative way; i.e. in terms of a more developed strategic behaviour when dealing with problems (they thereby differ in their schemes for learning). Experts have better organized domain-specific knowledge; therefore, they are able to get new information from their field of study faster since the previous knowledge, which is available, functions as an integrating structure for new incoming information via associative memory with only a minimal cognitive effort."

In general, we can say that software engineering integrates problem solving strategies which, according to cognitive psychology, only experts are capable of.

2. LIFE CYCLE OF INFORMATION SYSTEMS

Wikipedia aptly likens the life cycle to the bridge. The bridge is actually a means for making human activities easier. Accordingly, software systems should serve the similar purpose. Like bridges, software has to be designed, used, maintained until the end of its lifespan and eventually decommissioned. In construction engineering, there is a life cycle of particular buildings which is very similar to that of software.

There are a lot of classifications of software life cycles. We have chosen Systems Development Life Cycle (SDLC) by The Department of Justice of the USA from 2003.

- **Initiation** - firstly, it is necessary for the object with enough money (customer) to feel the need of an improvement implemented by a software system. Let's call it 'intention'.
- **System Concept Development** - this intention is re-examined in terms of feasibility and suitability. The system scope, on which base costs that must be approved by the customer are calculated, is loosely defined.
- **Planning Phase** - this concept is developed in a way that describes the function of the enterprise after the installation of the approved system. In addition, specific project plans are drawn up and approved.
- **Requirements Analysis Phase** - all requirements are profoundly set out, which enables a further development of the system. These requirements usually concern data, computing power, security system, requirements for maintenance etc.
- **Design Phase** - physical features of the system are designed within this phase. Main parts of the system, its inputs and outputs are defined. All that requires an input or user's approval has to be recorded and re-examined by the user.
- **Development Phase -** all specifications laid down in previous phases are put in the developed software. They are subsequently tested.
- **Integration and Test Phase** - individual system components are integrated and systematically tested.
- **Implementation Phase** - the system is installed and launched at the customer. The customer must test and approve of the system. The phase ends by putting the product into operation.
- **Operations and Maintenance Phase** - The operation of the system should be adequately monitored. If necessary, required system modifications are introduced. This process goes on until the system can be effectively modified to the customer's requirements. If the process is not possible to continue or it is too expensive, a phase of a new software planning begins.

- **Disposition Phase** - decommission of the system. A special attention is paid to precisely storing data processed by the system in order for the information to be effectively transferred to another system or archived in accordance with official regulations and policies of the record administration in case of a future access.

The entire software life cycle is described in the following text. Individual methods of software development will be shown in order to closely focus on RUP method (Rational Unified Process). This method involves phases from Initiation to Implementation. RUP method works a lot with UML; therefore, six different kinds of UML diagrams will be shown. Business Process Model and Notation, which can be useful at the beginning of new software development, will be discussed. Software testing and its maintenance by means of ITIL method will be described in a separate chapter.

3. SOFTWARE DEVELOPMENT METHODS

In general, they refer to procedures, rules and devices leading to the software development. Sometimes they are also called software process. Individual devices designed for software development are called Framework. The aim of these procedures is an effective development and maintenance of the software.

Following these procedures, we reduce the risk of unexpected problems and make the work schedule of particular software clearer. If individual developers follow the same procedures, they thereby encourage the cooperation thanks to the previously formulated program development rules.

As time went on, a lot of software development methods were devised. All the same, until now, there is no detailed and clearly defined frame of a software development method which could be considered as referential. These methods include:

3.1. Waterfall model

Waterfall model is a sequential and linear process which is approached as a sequential progress which flows through individual phases of conception: design, testing, integration, maintenance and alternatively transition. This model originated in 1970s. It can be regarded as an essential model on which other models are based.

Moving from one phase to another should be carried out only when its preceding phase is partially or completely verified. Unfortunately, this model is often not possible to be made use of in real life. For example, the customer often does not change program specifications until the stage of program development or until its application. The main problems are as follows:

- It is not possible to assess the final quality of the product in the course of software development on the grounds of previously established criteria for software specifications.
- The final product depends on the initial requirements for the final software.
- The time needed for the software development is too long.

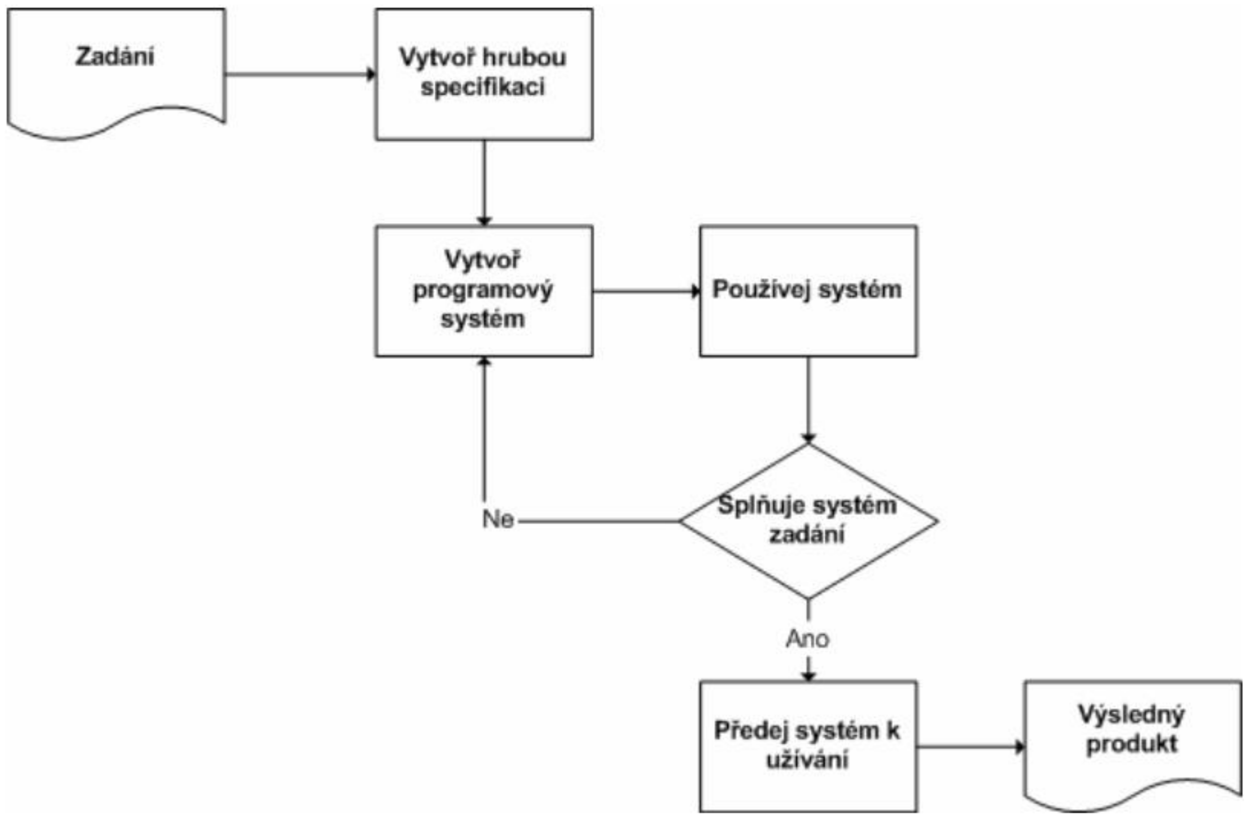
In regard to this model, an effort to remedy defects would result in development of other models. These processes are explores by prof. Vondrák (2002):

3.2. Iterative and incremental development

Individual steps are taken in repeated cycles "iterations". Each iteration takes on a smaller portion (increment) of functions into the developing software which leads to the target state. Thus the required program is being developed and its functional capabilities are being enhanced. The customer thereby has the opportunity to try out the developing software as soon as its early phase of development. In this way, he can specify his requirements for the developed software more effectively. Thus errors, defects or wrongly imposed requirements for the program will be detected significantly earlier. Owing to the fact that the customer sees the software from its early phases, he can be more involved in the final form of the product. In contrast to Waterfall model, differences between individual phases are very slight. It is rather a parallel in which iteration represents one small waterfall. RUP process, which will be dealt with in a separate chapter, also falls into this category.

Exploratory programming

Prof. Vondrák (2002) regards this model as a deterrent example which is still sometimes used. He calls this model Exploratory Programming.



3.3. Agile software development

Most of the software processes originated in large corporations. These processes strictly required following the scheduled procedure and obligatory creating a lot of documents. That is fine in large companies where a lot of people work on the project, or where a high level of reliability is required (e.g. power grid control). In case of small projects, the time spent by obediently following all the procedures and other formalities was much longer than the time meant for programming and testing.

In response, agile methods or techniques arose at the beginning of new millennium. Basic principles of these techniques were formulated in the Manifesto for Agile Software Development.

- Individuals and interactions are more important than processes and tools.
- Working software is more important than comprehensive documentation.
- Customer collaboration is more important than contract negotiation.
- Responding to changes is more important than following a plan.

The basic principle of these methods is to provide developers with the environment and support they need for the software development. Developers focus on the software development, its design; they do not excessively deal with documentation. These methods fall into iterative methods. Its high priority is active customer collaboration. Work schedule is usually drawn up within the next iteration. Because of their “anarchistic” nature, these techniques are suitable for smaller quality developer companies. The key principle of conveying information within a development team is face-to-face communication.

Currently, there are several agile methods Such as Extreme programming and Scrum (Sklenář, 2007).

4. RUP

RUP process (Rational Unified Process) originated in the second half of 1990s in collaboration with several companies under the leadership of Rational Company. This company became an IBM division in 2003. RUP is not the only established process; it is rather an iterative process framework which should be adapted to the needs of developer organizations and a software development team. This adaptation is implemented in the way that development teams chose process elements which correspond to their needs. RUP originates from UP (Unified Process). RUP currently belong among widespread software process models. A lot of tools support it.

A common Waterfall model defines separate roles. It thereby happens that an analyst accepts customer's idea of the system functioning. This idea is subsequently translated to a document, which will be handed over to the programmer. If there is no effective collaboration between the analyst and developer, the developer has to focus on requirements set out in the document. In such a case, developer's interpretation of the requirements can significantly differ from customer's original idea.

This process is built on several principles. This process in based on the idea of **software iterative development**. Thanks to that each program should be developed in recursions (iterations). Iteration should be created by an executable code.

Žáček (2017) lays down basic RUP principles:

- The effort to minimize project risks as soon as possible on a regular basis.
- To make sure that customers are provided with added value.
- Focusing on executable software.
- To undergo changes in early phases of the project.
- Early drafting of executable architecture.
- Re-using existing components.
- Close collaboration - all are one team.
- Project quality is determined from all its proportions, not only a part (testing).

Original requirements for the software are essential for the quality development of the product. RUP creates **Requirement Management System**. It manages their acquisition, documentation and monitors changes in requirements. Subsequently, this system is used by developers in communication with customers. Changes in the software development are managed in the same way.

A lot of parts of the software product are alike. Therefore, their reinvention is waste of time. RUP introduces **Component-Based Development**. As soon as we have necessary components, the development of the product starts to integrate relative components. This integration can be likened to Lego or building up a desktop computer.

Thanks to the complexity of developed programs, **software visualization** is very important for a better idea. RUP works on UML language.

Verification of software quality is a very important aspect for the feedback to developers in order to satisfy customer's needs thanks to the working software. This verification should involve all product developers. RUP specifies specific procedures assessing the software quality.

RUP software development is divided into four phases. RUP phases could be referred to as individual project stages and its changes in time. Each phase often consists of several iterations.

4.1. Schematic model

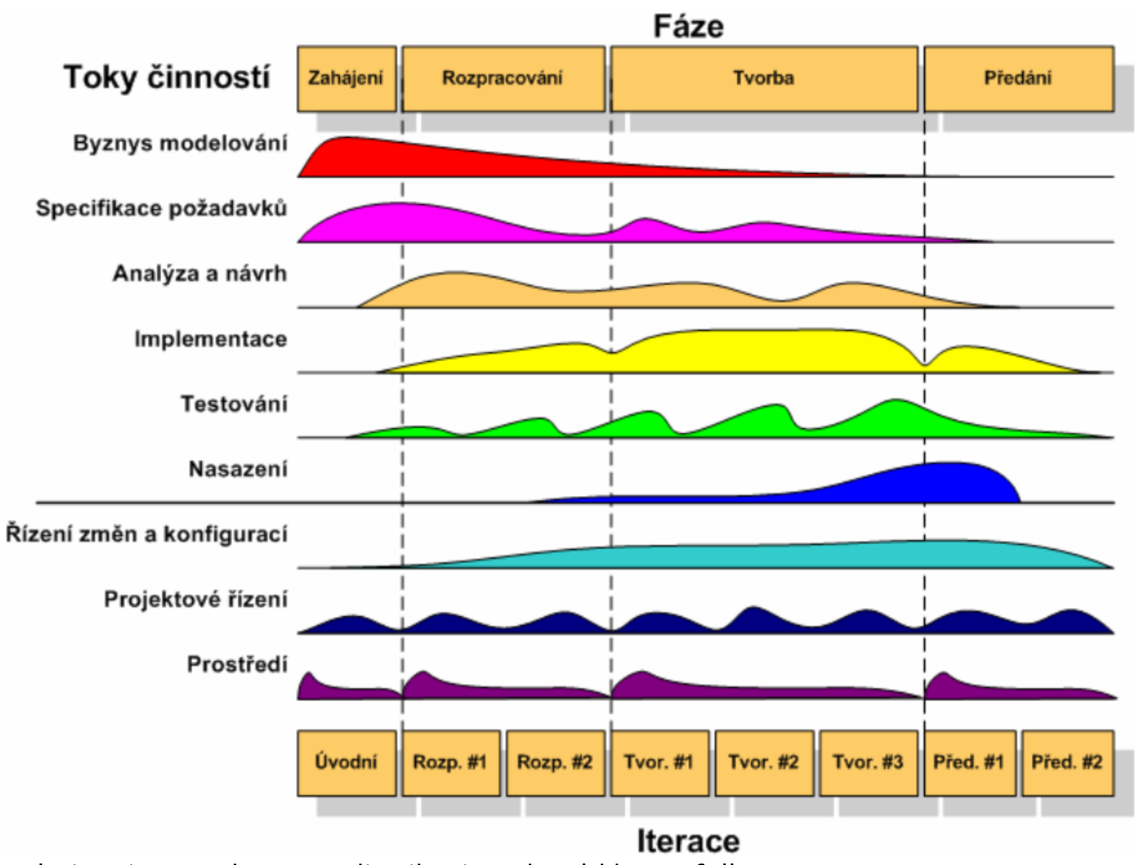
As such, schematic model consists of six engineering disciplines and three auxiliary disciplines. Engineering disciplines include:

- Business process modelling
- Requirement specification
- Design and analysis
- Implementation
- Testing
- Deployment

Auxiliary disciplines involve:

- Configuration and change management
- Project management
- Preparation of the project environment

A relative frequency of activities of individual disciplines in correlation with individual phases and project iterations was worked out by prof. Vondrák (2002).



A relative time and source distribution should be as follows:

	Inception	Elaboration	Construction	Transition
Sources	Ca 5%	20%	65%	10%
Time	10%	30%	50%	10%

Žáček (2017) suggests outputs of individual phases:

- The output of Inception is the understanding of crucial issues, the aim of the project, assessed risks.
- The output of Elaboration is executable and verified architecture (=
- Functioning part of the application).
- The output of Construction is beta-release application - a relatively stable, re-executable and almost completes application.
- The output of Transition is a product ready for final deployment including the entire documentation and hardware. The major difference also lies in the fact that each phase is characterized as follows:

Individual phases involve various numbers of employees and posts. It is important for the employees to always work together in one or more teams. If needed, they help and advise each other.

4.2. Inception

At the beginning, it is necessary to assess the project size, the requirements, time and financial costs of the whole project implementation. Žáček (2017) suggests 5 goals of this phase:

- Understanding the process - conveying the vision, defining the system size, its limits; understanding the person who wants the system and how he could benefit from it.
- Identification of key system functionalities - identification of the most critical Use Cases.
- Suggestion of at least one possible solution (architecture), (i.e. Whether it is possible to continue in the development Note: author)
- To be informed about costs, project conception and risks.
- Definition/modification of the process; tool selection and setting.

Definition and imposing of requirements

Within imposed requirements, we try to obtain maximum information about what users expect from the new system and which essential functions the system is supposed to perform. Techniques of data collection include:

- Documents and records
- Interviews with future users
- Questionnaires
- observation
- Oral history of users

These requirements can be divided into:

- **Functional requirements** these are requirements for system functionality from the system; alternatively, system functional capability and incapability should be defined. A use case diagram is an effective tool for illustration of functional requirements.
- **Non-functional requirements** these are requirements for system properties as a whole. They can concern efficiency, reliability, security, system capacity; alternatively, they can refer to organizational requirements such as maintaining standards

and following established procedures. Alternatively, they can involve requirements regarding valid legislation.

Goals

The key goal is to convey a vision of the project. This vision should be defined in regard to the user. However, it should at the same time contain a basic technical view of the applied technology or architecture components. This vision should mainly set out requirements for the future software. This phase takes place only once. All the same, in more complex cases, it can be repeated several times.

At the same time, the degree of risks should be assessed in certain areas; these risks are as follows:

- Risks of technical branches (technologies, compatibility, communication with other systems)
- Financial risks - development costs and subsequent maintenance
- Time risks - the more sources we have, the shorter the independent development is likely to be

At the end of this phase, it is necessary to assess whether the development is possible to continue. The sooner we put an end to an obscure project development, the lower the costs of the project development are. Lifecycle Objective Milestone (LOM) is used for that purpose with evaluation criteria as follows:

- Interested parties concurrence on the project scope, costs and schedule estimates
- Agreement that the right set of requirements for the system have been imposed and that there is a shared understanding of these requirements
- Agreement that the cost and schedule estimates, priorities and risks correspond to the development process
- All risks have been identified and mitigation strategies have been decided on

4.3. Elaboration

This phase initiates forming of the project. The aim of this phase is to establish the system architecture in order to produce a majority of healthy functionalities in the next phase. This architecture is based on findings from the previous phase. Crucial issues of the architecture design of the developed software are to be dealt with here. The system functionality does need to be perfect. Nevertheless, the majority of essential functionalities should have already been integrated into the system. The research team should be aware of the fact that the architecture design must stabilize during this phase.

At this point, the crucial issue is analysed and the project architecture acquires its basic form. The accurate architecture design efforts to mitigate risks associated with the software development. These risks mainly concern:

- Time and finance
- Architecture accuracy
- Processes and technologies
- Software development objective

A series of small iterations can mitigate the degree of above-mentioned risks at this point; they also help monitor the principal objective of the development. If we create the system using similar technologies like before, there is obviously a lower degree of potential risks. Therefore, we are able to reach goals of this phase within iteration. On the other hand, if we are not familiar with the used technology or the project is more complex (e.g. Stricter safety requirements), we need two or more iterations. In case this phase sees a more substantial change in the architecture, another iteration, which will verify necessary functionality and stability of the modified architecture, should be added. A basic principle "the later the foundation of the construction is changed, the more expensive the whole reconstruction will be" applies here.

In order to verify the software development objective, a beta version of user interface, on which the most important system functionalities would be tested, should be created.

At the end of this phase, the system architecture must be stabilized. The executable architecture must verify the system architecture and simultaneously serves as a means for verifying critical system functionalities. These critical functionalities must also be modelled, for example, by means of UML sequence diagram. We also try to find identify potential problems and risks. We should afterwards group selected objects to packets regarding visibility rules and future product configuration. We should also have an idea how the data in the database will be handled, as a result of which we should regularly integrate the components. That means to determine the order and way in which the selected components will be integrated.

Eventually, testing of critical scenarios constitutes a very important part of this phase. Testing of critical scenarios may demonstrate our progress in the elaboration phase. Apart from that, critical scenarios should be tested in regard to aspects such as heavy system load, sufficient system architecture performance or collaboration with external systems.

At this point, we test system objectives by means of Lifecycle Architecture Milestone (LCA), while LCA evaluation criteria are:

- The product vision and requirements are stable.
- The architecture is stable.

- The key approaches and procedures are tested and verified, so that they have been proven applicable.
- Testing and evaluation of executable prototypes have demonstrated major project risk elements, which have been successfully resolved.
- The iteration plans for the following phase are drawn up to allow the work to proceed.
- The iteration plans are supported by credible estimates.
- All participants agree that the current vision can be met if the current plan is executed in context of the current architecture.
- Actual costs versus planned costs are acceptable.

The project may be aborted or otherwise re-thought if it fails to reach the milestone.

4.4. Construction

The key goal of this phase is to build a credibly working software system while minimizing its costs.

This phase focuses on developing components and other system elements. This phase is further characterized by creating encryption which is heavily time and human (it requires a lot of programmers and testers) consuming. The rest of the functionalities are being designed here; i.e. remaining main ones (required by the customer) and most of the others. If an intervention in the system architecture is to be dealt with, the previous phase is likely to be wrongly completed.

Essential prerequisites for a successful completion of this phase are: architecture compactness, parallel development of individual teams, configuration and change management and automated testing.

Larger projects may perform several iterations (commonly 2 - 4). This phase usually performs the most of the iterations. Iteration planning corresponds to the number and type of not yet implemented functionalities. Each iteration deals with a separate project segment. Ideally, the architecture, which is further developed, used, or re-used, is created from previous phases. The system is subjected to integration and testing. With respect to the effective organization, one team is responsible for the architecture while other teams parallel work on the subsystem development.   These "parallel teams" mainly communicate with the team responsible for the software architecture.

Thanks to the iterative development, a lot of files and their versions, which are subsequently tested and integrated, are developed. For that reason, configuration and change management must be introduced. This management registers individual configuration and changes. If such a system works, developers can entirely focus on the further development and thereby enhance their effectiveness.

At the end of this phase, a beta version of this program must be developed including helps in the application, installation instructions, user manuals and tutorials. In this way, future users will be able to try out the software and give us a constructive feedback.

At the end of this phase, we should pose ourselves several questions according to IOC Milestone (Initial Operational Capability)

- The beta version is ready to be released to the first users (testers)
- Users are prepared and able to use our beta version
- Actual costs versus planned costs are acceptable

4.5. Transition

This phase aims at the system transition from its development to its regular use and remedying its major defects (they usually concern its performance, user-friendliness and functionality). These phase activities include:

- Training of administrators and end users
- System testing to verify expectation fulfilment of end users
- Preparation of marketing materials
- Environment and data preparation, for example data migration from the old to the new system
- The product is closely monitored in terms of qualitative frameworks established at the beginning of the project.
- Internal project evaluation, learning from mistakes and problems arising from working on the project.

English wikipedia from 24. 6. 18 further states: *"The system also undergoes Evaluation phase; each developer who does not carry out a purposeful work is replaced, or removed.*

This phase should also be finished by a milestone. PRM (Product Release Milestone) criteria are:

- Users are satisfied.
- Final costs versus planned costs are acceptable. Things to be changed in order to approach the problem

5. BUSINESS PROCESS MODEL AND NOTATION

The key goal was to devise a graphic tool which is comprehensible to all business users (from analysts through developers to business process owners), which means a more effective communication between the developer and customer. Business Process Model and Notation (hereinafter BPMN) consists of a set of simple graphic elements from which a business process model can be modelled. BPMN has been developed since 2005. Version 2.0. came into existence in 2011. Process flowchart is based on a flowchart which is very similar to UML activity diagram, which will be discussed in the further text.

Flow Objects

It is closely connected to information flow in the process. Flow objects are the most important graphic elements. They can be divided into three groups - Events, Activities and Gateways.

Events

Events are marked with a circle in which an icon can be displayed. It marks an event which directly controls the process flow. Events are divided into **Start** (Start event) **events**, which initiate the process and are marked with a circle with a narrow border - sometimes with green colour.

Furthermore, there are **End events**, which indicate the end of the process, or the result of the activity. These are marked with a bold border, or a bold icon inside the circle - sometimes with red colour. Some sources also mention Intermediate event which is marked with a double border.

Activity

Activities are marked with a rounded-corner rectangle. This element describes work or activity. The activity can be either **atomic** (i.e. Task), which is considered a single unit and which must always be performed as a whole.

Alternatively, it can be broken down to a single process which is called Sub-process. This kind of activity is used in processes which we do not want to reveal on a particular level. Sub-process activities are marked with a plus sign at the bottom line of the rounded-corner rectangle.

Gateway

Gateways are marked with a square or diamond shape. It determines forking and merging of process flows, e.g. decision-making or parallel processing.

Connecting Objects

They connect flow objects or artefacts. Thanks to the connection, a new structure (framework) of the business process arises. Connecting objects are divided into three basic types:

- **Sequence Flow** - is marked with a solid line and arrowhead. It shows the order in which the individual flowchart units should be performed.
- **Message Flow** - is marked with a dashed line and open arrowhead. It displays the message flow across organizational boundaries of the process.
- **Association** - is marked with a dotted line. It associates objects to additional information. It is also used for displaying inputs and outputs.

Swim-lanes

They display participants of the process and refer to the organization and categorization of activities in the process. They consist of two types.

- **Pool** - represents participants of the process, or alternatively, it separates different organizations. It can contain more lanes. One pool contains one separate process.
- **Lane** - is situated under the pool. It is used for organizing and categorizing activities.

Artifacts

They bring some new information into the process. They are independent from flows.

- **Data Object** - is marked with a bent-corner rectangle (a sheet of paper). It shows data produced in an activity; alternatively it tells us which data are required for an activity.
- **Group** - is marked with a rounded-corner rectangle and dashed lines. It is used to group different objects.
- **Annotation** - gives the reader a clear and understandable impression.

We provide a model of ordering and delivering pizza as an example.

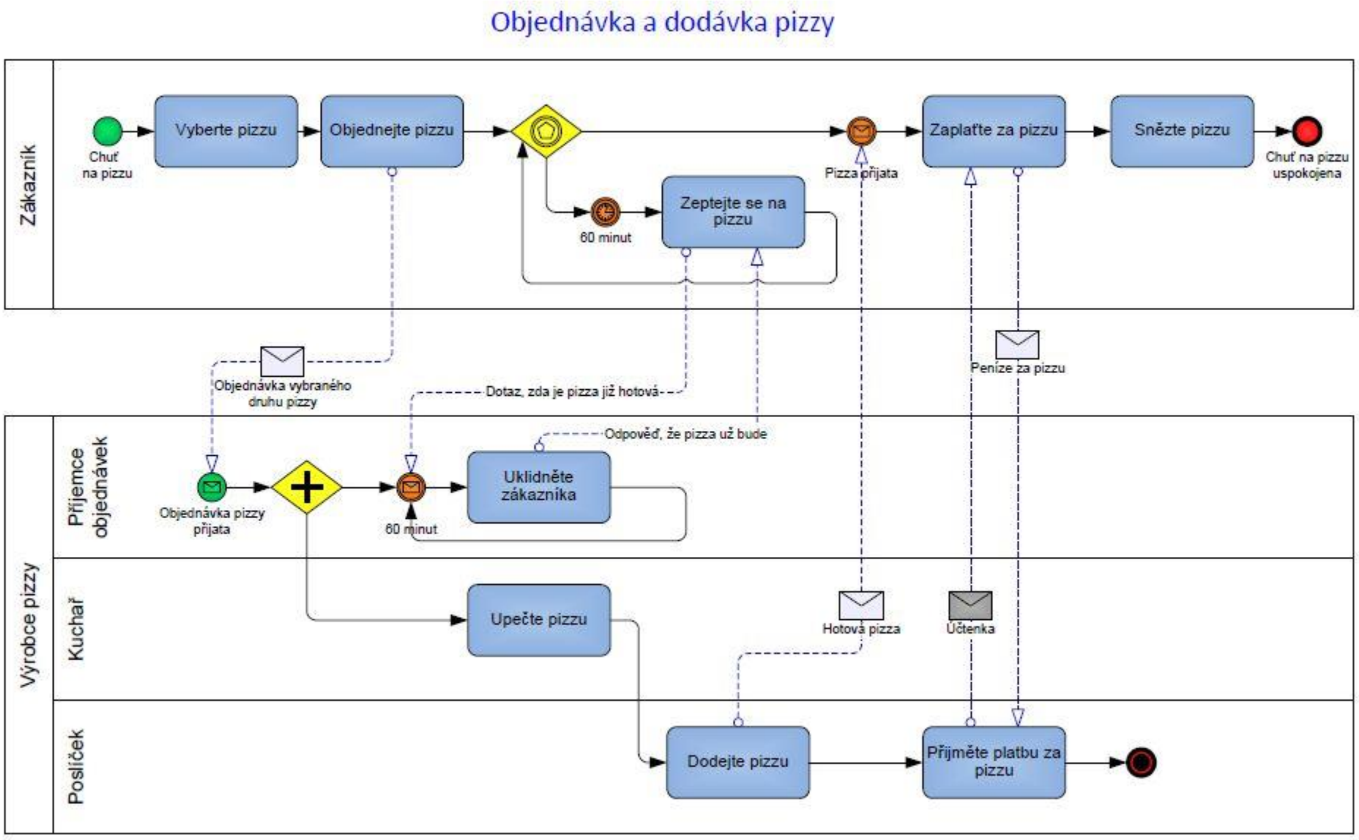


Figure 1 source: http://bpmn.horcica.cz/wp-content/uploads/2011/11/BPMN-2.0-Examples_CZ.-Dod%C3%A1vka-pizzy.jpg

6.UML

As a matter of fact, a lot of software systems are exceptionally complex. Thus, their development lays down stringent requirements for developers' imagination. When a project involves more people, the future software system needs to be precisely defined - modelled. There are a lot of approaches towards such modelling. All of them have their pros and cons. In order to encourage an effective cooperation, it is important for everyone to know, use and understand the modelling system its terminology. For that reason, UML (Unified Modelling Language) was developed. Standard UML is defined by Object Management Group (OMG).

UML refers to a unified language for drawing diagrams. UML enables specifications (structure and model), visualization (flowcharts), construction (Software development methods) and documentation of software system artifacts. There is a large scale of diagrams. Each of these diagrams is used for a different goal and in different project phases. UML provides 13 types of flowcharts in total. In this text, diagrams will be divided into Structure diagrams and Class diagrams.

Structure diagrams

Structure diagrams describe the project structure. They are often used to describe the architecture of the developed software. We will deal with several selected diagrams belonging to this category.

Class diagram

This diagram shows project class divisions, their relationships, operations and methods or methods of the methods. These classes are written in a rectangular vertically divided into 4 parts. This diagram is the cornerstone of object-oriented modelling. It is also used for data modelling. Furthermore, the diagram depicts a static structure of the system, and to some extent depends on a specific programming language. The aim of this activity is to suggest the way in which the product is implemented in the implementation phase.

Examples of class diagrams are provided below.

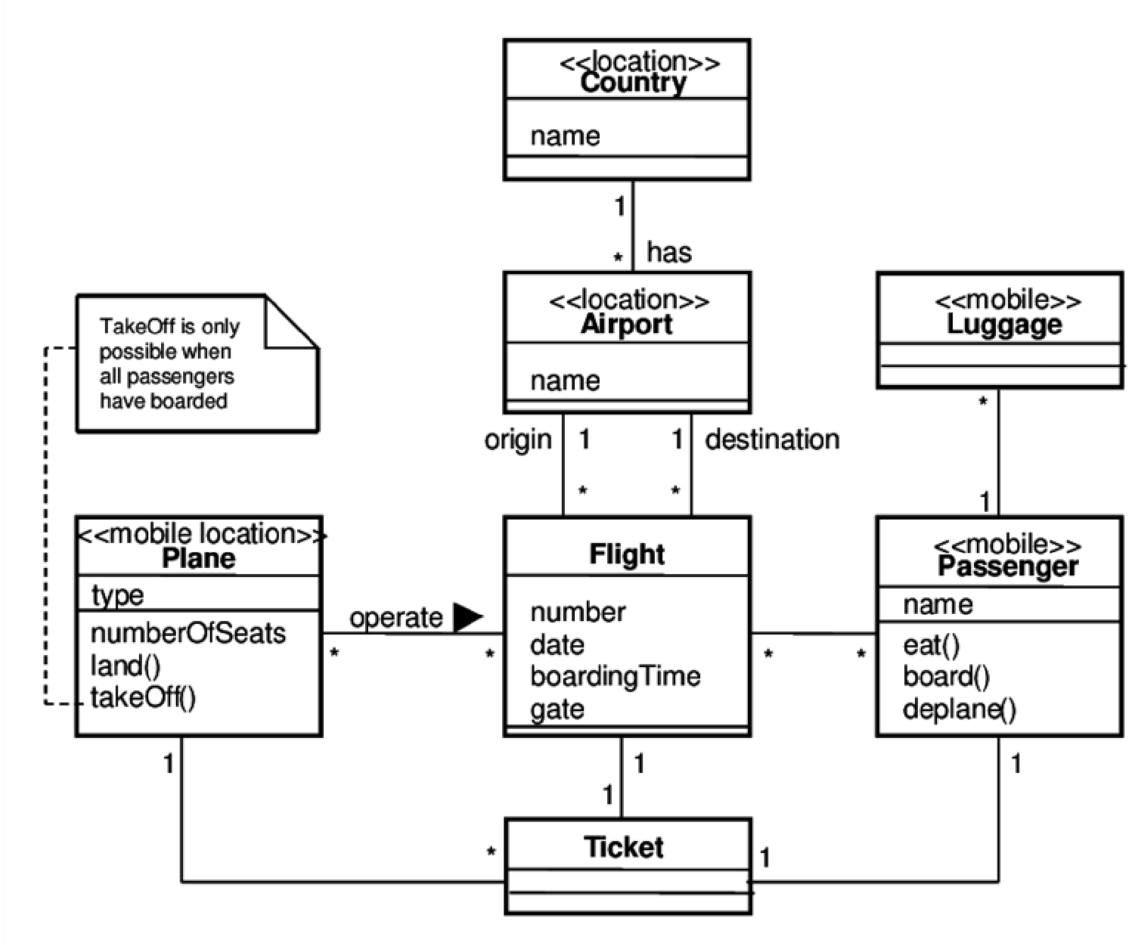


Figure 2 Source: Andrade et al, Recent Trends in Algebraic Development Techniques—16th International Workshop, WADT 2002, Frauenchiemsee, Germany. Vol. 2755 of LNCS.

Component diagram

This diagram works with components used in the system and describes the way the components are interconnected to form larger components of software systems. It is used to describe the structure of different complex systems. The diagram is mainly used for designing software by components. It also has a higher level of abstraction than class diagrams. One or more classes are implemented within a single component.

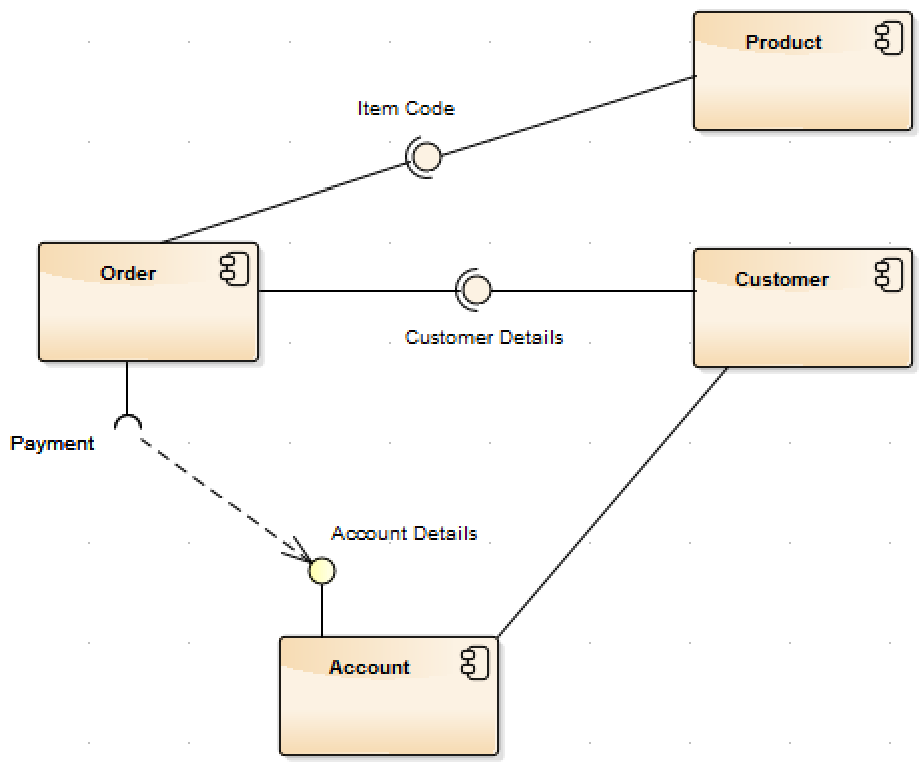
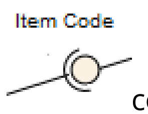


Figure 3 Source: http://sparxsystems.com/enterprise_architect_user_guide/13.0/model_domains/componentdiagram.html

The picture illustrates 4 components in total.

Assembly connectors 

connect the order interface with the product and the customer. In this case, components require ID product information and details about the customer.

A simple line between Customer component and Customer Account component illustrates the relationship between the components.

Object diagram

Object diagram depicts objects (class instances) and their relationships (links - association instances) in one moment. This diagram looks like a class diagram and can be actually considered as its specific example. It is mainly used when we want to show examples of data structures in one particular moment.

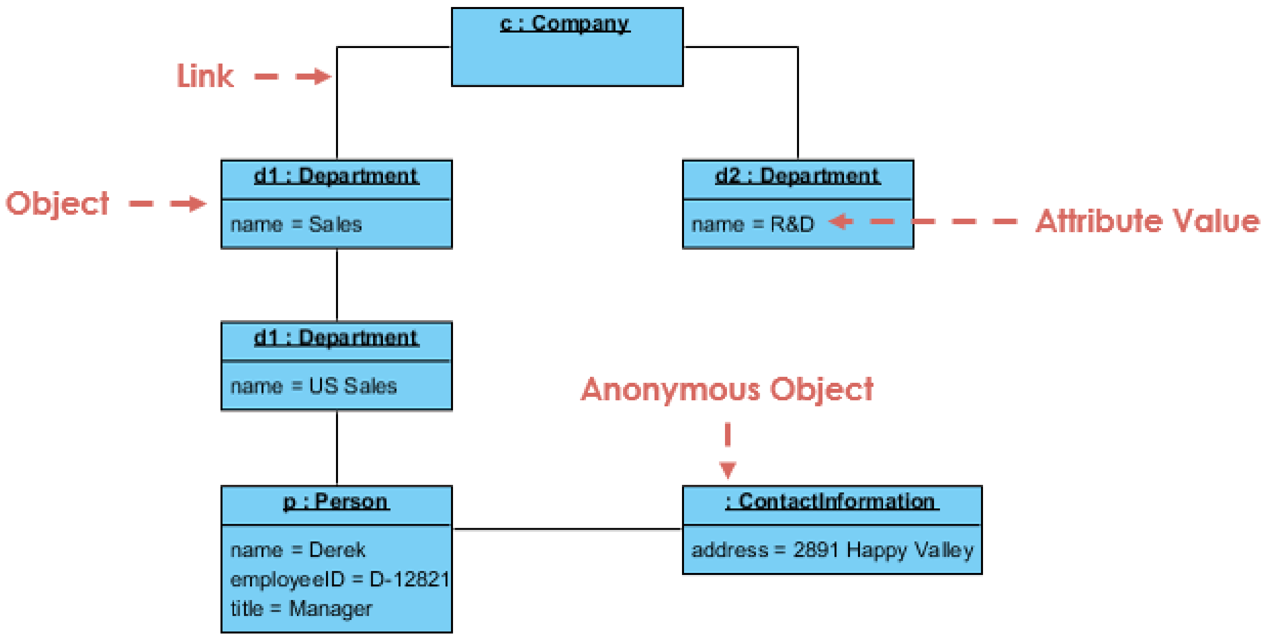


Figure 4 Source: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/

7.BEHAVIOUR DIAGRAMS

These diagrams describe separate processes by means of activities representing their states and their transitions. Regarding the fact that activity diagrams describe behaviour of the system, behaviour diagrams are used to describe software system functionalities.

Activity diagram

This diagram is designed for modelling computing and organizational processes (i.e. Work process) and can be used for modelling data flows. The diagram models processes by means of activities representing its states and transitions between individual states. Thanks to that it is one of the diagrams capable of identifying the behaviour of the system.

- Rounded rectangles represent actions.
- Diamonds represent decisions.
- Bars represent separate activities.
- The black circle represents the start (initial node).
- The encircled black circle represents the end (final node).

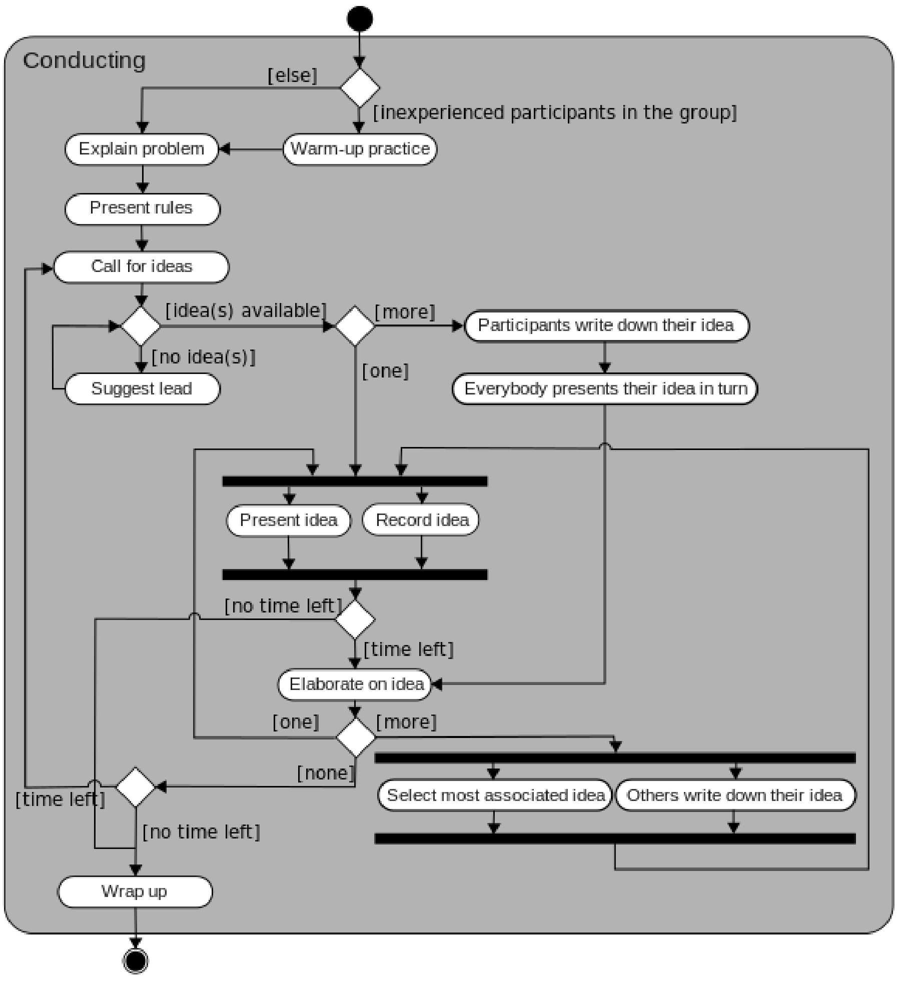


Figure 5 source: https://en.wikipedia.org/wiki/Activity_diagram#/media/File:Activity_conducting.svg

Sequence diagrams

Sequence diagrams displays object interactions which participate in particular system functionality. Apart from object and classes, it displays sequence of messages necessary for achieving the particular functionality between individual objects in regard to their time-order. These diagrams are often used in the implementation of use cases regarding the logic of the development system. The aim of this activity is to suggest the way in which the product will be implemented in the implementation phase.

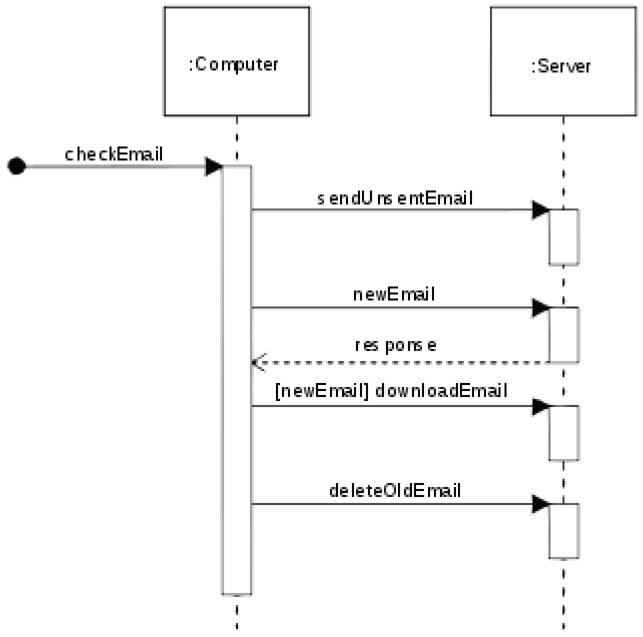


Figure 6 Source: https://en.wikipedia.org/wiki/Sequence_diagram#/media/File:CheckEmail.svg

The time-line in the diagram is vertical (time flows from the top down). The placement of objects is horizontal. Our picture depicts 2 objects (Server and computer); dashed vertical line is also called the lifeline. Our case refers to objects which had existed before the diagram was drawn and will exist even after. Rectangles on the lifeline illustrate an activity. Horizontal arrows represent messages. The arrow direction displays the sender and addressee. The full line demonstrates the mandatory message, while dashed line represents a non-mandatory one.

Use case diagram

It usually represents the relationship between the customer and the system, where it demonstrates different use cases in the customer-system relationship, which is often verbally described by means of scenarios. In order to map scenarios to use case diagrams, use case implementations are applied. Here, the algorithm of transcription of scenario entries to the diagram is to be dealt with.

For example, BPMN can be the first impulse for this diagram to be drawn. The result of this is a list of activities which can be performed by the developing software instead of, for example, people.   Diagram figures are called actors. Actors are associated with relevant use cases. The simple line is called association. The horizontal direction further demonstrates the structure of the actors. The bottom actor inherits characteristics from the actors above him. This type of relationship is called generalization. Include relationship is implemented when the use case from which this relationship develops is implemented.

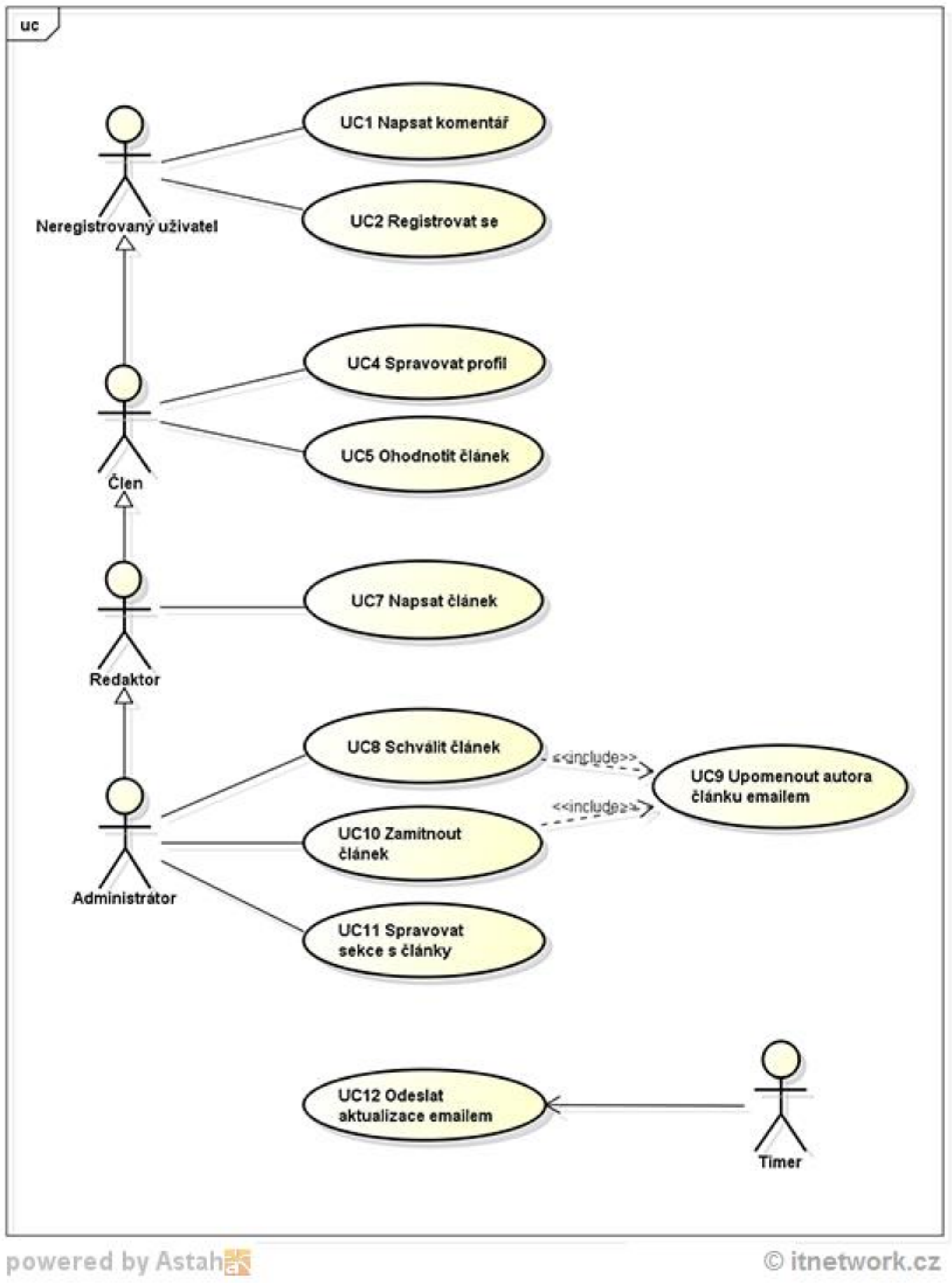


Figure 7 Source: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram

7.1. Software testing and installation

There are a lot of methods of software testing, while each testing method usually focuses on a different testing area. Individual methods can be divided into functional and non-functional testing. Functional testing focuses on meeting all the user requirements. Targets of the tests are defined in ISO/IEC 12207 Standard or can be obtained from results achieved by means of FURPS+. Each requirement is supplied with a test.

On the other hand, informal test cases do not directly test the system functionality; they are based on normal expected operations, which the system should be able to ordinarily perform. These are as follows:

- Security testing - tests the system security
- Load testing - tests putting demand on a system and measures its response.
- Usability testing - measures conditions under which the system can be used. For instance, internet speed connection.
- Documentation testing - user documentation is tested in regard to its comprehensibility and consistency. Within development documentation, the amount of coverage is tested.

Software systems are also tested regarding their verification and validation.

- **Verification** answers the question: whether the developed product meets all the technical requirements.
- **Validation**, on the other hand, answers the question: whether the software corresponds with the intended use.

For example, if a customer wants E-shop system, our developed system has a lot of entertaining functions; however, there is a little opportunity to see them so that customers do not usually find them and thereby do not buy them.

Sometimes we speak about Black box and White box testing.

- **Black box** testing approaches the developed software system as a black box. Therefore, it does not deal with the inside. It examines the functionality only from the end-user's perspective. It usually checks whether given inputs correspond with the required outputs. It also tries to find unusual or unexpected inputs which might not adequately respond.
- **White box** testing, on the other hand, is familiar with the program structure. It examines whether separate program functions work properly. It tries to test each function separately.

Vondrák (2002) advises following activities to deploy software system to the user:

- Development of the final product or its versions
- Completion of the software system
- Distribution of the software system
- Installation of the software system to the user
- Assistant services to users
- Planning management of beta testing
- Migration of existing data and software products

8. MEASURING QUALITIES OF SOFT-WARE SYSTEMS

If we succeed in meeting user's/customer's requirements at the end of the whole process, quality software is likely to have been developed. All the same, this indicator is highly subjective. As a matter of fact, each user/customer has different expectations. It is therefore necessary to adopt more objective criteria for assessing the quality and benefits of software systems.

These criteria are called metrics. (Metric may also refer to Metric - generalization of physical distance in mathematics)

Žáček (2017) defines the concept of metric as follows:

A metric may be defined as such: "A metric is a clearly defined financial indicator or non-financial indicator or evaluation criterion that is used for evaluating levels of efficiency within the particular area of business performance and its effective support by means of IS/ICT." The group of metrics associated for a certain purpose (i.e. Related to a specific area, process or project) are called "portfolio metrics".

Metrics are usually divided into hard and soft metrics.

- Hard metrics - are objectively measurable , for example average response time, the number of error per time unit or warranty period
- Soft metrics - are more subjective and very hard to be objectively measured. They can be evaluated on a scale typically from 0 - 100 in the order of evaluated products or alternatively a verbal evaluation can be carried out

In addition, there are plenty of systems for managing the quality of software system development. By and large, customers like when their products have the quality certificate. In fact, they like to pay extra money for that. These systems help minimize errors and boost the efficiency of the development process. Some of them will be briefly suggested in the following text.

8.1. CMMI

Capability Maturity Model (Integration). This model is remarkably widespread in the USA and Japan. It is designed for development teams. This model is relatively elaborate thanks to which it can serve as a guide for enhancing the quality of the development team. It consists of set of rules and recommendations which should be observed for a rapid development and designing of new products. Furthermore, it focuses on an organization, planning and monitoring of development processes. It is specified by dividing development teams into five maturity levels and capacities. In this way, individual teams can move within these levels. Maturity levels are assessed by a well-trained evaluator in an exact procedure.

Maturity levels are (according to Wikipedia)

- Initial: On this level, teams perform these processes either not at all, or only partially.
- Managed: Project management is defined and activities are planned.
- Defined: Procedures are defined, documented and managed.
- Quantitatively Managed: Products and processes are quantitatively managed.
- Optimizing: The team continually optimizes their activities.

Capacity levels

- Incomplete: Some activities are not performed.
- Performed: Activities are performed.
- Managed: Activities are performed according to their management.
- Defined: Activities are performed according to their definition.

8.2. ISO 9000:2001

In contrast to CMMI, it is only loosely defined. It defines quality management system. This standard allows specific organizations to demonstrate their capacities for production and distribution of products. It is thereby only loosely defined. Thus, the development process is very specific. For this reason, English-Swedish TickIT or ISO/IEC 90003 interpretation were made.

8.3. Six Sigma

It is a set of techniques for process management whose aim is to understand and continually and regularly improve its efficiency, or to meet the customer's requirements by means of understanding customer's needs, process analysis, standardization measurement methods and their analysis using statistical methods.

Basic philosophical principles of this methodology are:

- A sustained effort to achieve stable and predictable results of the process is vital for a commercial success.

- Production and business processes have qualities that can be defined, assessed, analysed, improved and controlled.
- In order to achieve a constant quality improvement, the whole organization must be engaged, including the top management.

Main techniques for example include DMAIC cycle - it is meant for understanding and management of processes.

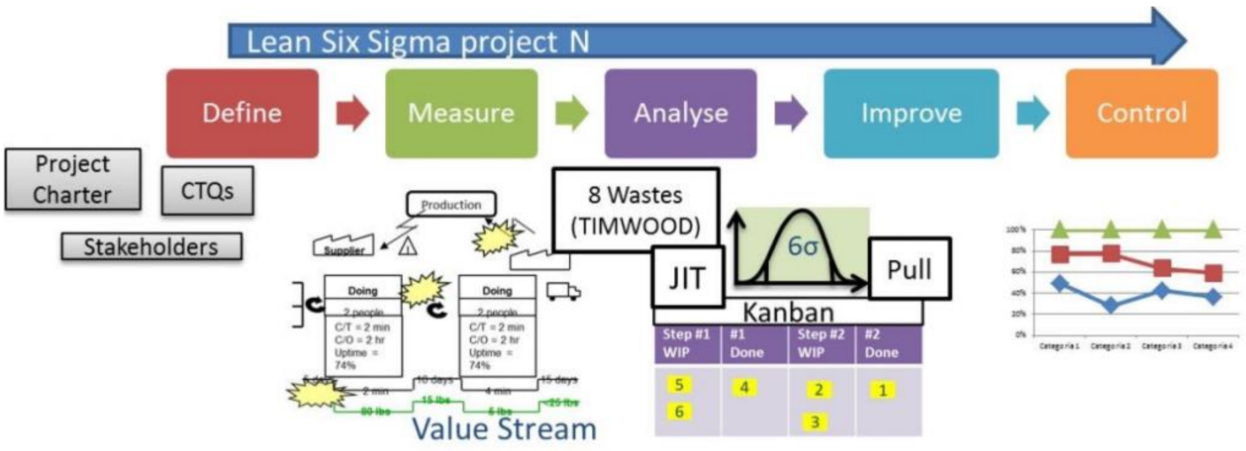


Figure 8 Source: http://www1.osu.cz/~zacek/infs2/02.pdf

DMAIC involves 5 phases.

- **Define** - defining an actual and ideal process. It suggests a way from the actual to the ideal process.
- **Measure** - measuring the current process. It means, for example, the average waiting time which provides us with “Hard Data” describing “objective reality”.
- **Analyse** - the measurement results are statistically analysed in order to produce a working model of a particular project; alternatively, the analysis tries to find out which parts of the process need to be improved.
- **Improve** - the implementation of thoughts coming from the previous phase. Major goals include satisfying the customer, increasing efficiency, reducing costs.
- **Control** - if a change has been successfully introduced, it needs to be standardized and controlled regarding whether or not it brought an improvement.

This phase was originally brought in by Motorola Company. Nowadays, this phase is made use of by for example Honeywell, HP, Texas Instruments, NASA and other companies. It is based on applied statistical procedures supplied with qualitative tools.

9. MAINTENANCE AND OPERATIONS OF SOFTWARE SYSTEMS

Until now, we only dealt with software development. All the same, the software development only marks the beginning of the software life cycle. It is mainly its operation that constitutes its crucially important part. To put it more precisely, if we developed a powerful software system which would after one month of its operation regularly collapse as a result of its poor maintenance, the customer would surely feel that the whole system was poorly developed. It is therefore necessary to carry out a regular maintenance for the software to operate effectively.

ITIL focuses on the techniques of maintenance and operations of information technologies. It is a set of concepts and techniques for a more effective use of IT services. IT services involve IT systems which key goal is to stimulate business processes (i.e. for the enterprises to do their work properly).

ITIL is actually a volume of book publications which contain invaluable experience in the field of Service Management of information technologies. It thereby establishes a framework for IT service management which tells us when and what to do. ITIL concept is actually very simple; or more precisely, why we should design and develop the whole process from the beginning, when a lot of other companies have it already introduced and regularly improved.

ITIL use proactive approach. It deals with problems in advance, not retroactively. If a problem has already occurred, it tries to find it out by active detection for which it establishes correct procedures. The whole process is regularly managed, monitored, assessed, evaluated and constantly improved. All these processes should provide the customer with an added value. However, the terminology can be a problem in some cases where companies use different technical terms. ITIL thereby strives for standardizing the terminology. These procedures are established as platform-neutral.

The biggest advantage of these approaches is probably their efficiency, thanks to which, for example, the duration of IT system blackouts can be reduced.

For that reason, two basic processes **Service Support** and **Service Delivery** have been introduced. Within these processes a customer contact point - Service Desk has been introduced. This point collects customers or users’ requirements. It also registers IT processes, which means that it can respond and fulfil the requirements. It also provides basic IT support.

In case of a problem, **Incident management** process, which tries to minimize unpleasant consequences for customers concerning problems with IT systems, takes place. One of the most important criteria is the speedy solution of the problem.

Moreover, Incident management is supported by **Problem management**, which administers the register for dealing with problems. It at the same time analyses existing problems and their character and, if appropriate, it introduces structural changes of IT systems.

Individual changes are registered in **Change management**. Release management further plans and manages individual changes of IT services (release)

ITIL Version 3 consists of 5 parts and one introductory book; (Žáček, 2017) characterizes them as follows:

- Service Strategy - deals with harmonizing business and IT, management strategy of IT services, planning.
- Service Design - deals with IT services, process planning (creating and maintaining IT architecture and procedures).
- Service Transition - transits IT services to business environment.
- Service Operation - delivers and manages process activities, application management, changes, operation and metrics.
- Continual Service Improvement - deals with IT services, improvement competences, methods, practices and metrics.

10. LITERAURE

Eysenck, M. W., & Keane, M. T. (2008). Kognitivní psychologie. Praha: Academia.

Nolen-Hoeksema, S. (2012). Psychologie Atkinsonové a Hilgarda (Vyd. 3., přeprac.). Praha: Portál.

Sklenář, V. (2007). SOFTWAROVÉ INŽENÝRSTVÍ [Online]. Retrieved June 29, 2018, from https://phoenix.inf.upol.cz/esf/ucebni/syspro.pdf

Softwarové inženýrství [Online]. (2001-). In Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation. Retrieved from https://cs.wikipedia.org/wiki/Softwarov%C3%A9_in%C5%BEen%C3%BDrstv%C3%AD#Softwarov%C3%A1_krize

Plháková, A. (2004). Učebnice obecné psychologie. Praha: Academia.

Rational Unified Process [Online]. (2001-). In Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation. Retrieved from https://en.wikipedia.org/wiki/Rational_Unified_Process

Rychta, A. (2011). Softwarové inženýrství [Online]. Retrieved June 29, 2018, from http://www.ksi.mff.cuni.cz/~richta/NSWI026/NSWI026-1-Uvod.pdf

Říčan, J. (2016). Používané metakognitivní strategie žáků pátých tříd ve specifické doméně čtení (Disertační práce). Praha.

US Department of Justice (2003). INFORMATION RESOURCES MANAGEMENT Chapter 1. Introduction.

Vondrák, I. (2002). Úvod do softwarového inženýrství [Online]. Retrieved June 29, 2018, from http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

WHITE, Stephen A. Business Process Modeling Notation [online]. [cit. 2018-06-26]. Dostupné z: https://is.muni.cz/el/1433/jaro2014/PV165/um/46771256/pr_06_bpmn.pdf

Žáček, J. (2017). SOFTWAROVÉ INŽENÝRSTVÍ [Online]. Retrieved June 29, 2018, from http://www1.osu.cz/~zacek/sweng/skripta_sweng.pdf