4 Information Systems

4.1 Information Systems Basic

An *organization*, such as a company, firm, corporation, institution, agency, is a system focused on the achievement of specific objectives and business goals. Each organization has official structure, employees, material and technical equipment, and management. Deliberately inflicting the functioning of the organization is not possible without extensive internal and external information flows. The *information system* of the organization is a deliberate juxtaposition of elements, e.g. people, documents, data, processes, methods of communications, network infrastructure, and computer equipment, that have to work together to enable the daily functioning of the organization. Information systems (IS) include all the *information flows* within the organization, as well as all elements related to the processing and transfer of information, namely: data sources, methods of transmission, collection points and transformation processes. An information system is a subsystem in relation to the organization. Information systems of modern organizations (especially business and industrial) make intensive use of information technologies. The technical bases of such information systems are computer hardware and software, network infrastructure and telecommunication channels.

Information flows are not evenly spread throughout the structure of the organization. The most intensive exchange of information takes place in the management subsystem, which covers a large set of informative and communicative activities, e.g. analyzing, planning, organizing, controlling, communicating, motivating, and so on. The management subsystem is a set of actions and measures for finding, collecting, storage, transmission and processing of information in order to be able to make decisions about managing the organization. Because of the importance of the processed *managerial information*, IS imposes certain requirements on the quality it has; managerial information should be:

- Honesty and reliable, information must accurately describe the organization's processes and economic conditions.
- Selective, information have to be selected in the context of the economic problem.
- Addressable, the method of delivery and presentation of information have to be consistent with the requirements of each individual manager.
- Aptness and relevance, distributed information has to be consistent with the specific managerial demand.
- Topicality and timeliness, information has to be available on request.

The engineering of computerized information systems consists of *information engineering*, *knowledge engineering* and software engineering. It focuses primarily on

the data that are stored and maintained by computers, and the information that is derived from these data. The main purposes of engineering information are:

- Important organizational data have to be located in the processing center by providing repositories or databases.
- For these datasets, there is a logical way to present and interpret them.
- Types and structures of the data used in the organization should not vary; only little changes (evolution) over time may be acceptable.
- In a stable organization, data values are relatively constant, but the processes that use these data may vary freely.

The main task of the designer of information systems is to provide a subset of *organizational knowledge* by means of known information. For knowledge representation in information systems, technologies of knowledge engineering and *knowledge bases* are used. Knowledge engineering is a discipline dedicated to creating effective systems that represent knowledge (methods of mapping reality) in expert systems and decision support systems.

The *primary data* collected in a computer's information system represent the *raw facts* about the organization and its activities, e.g. about financial and non-financial transactions. Deliberately structured and filtered data, having a specific meaning, can deliver new information, e.g. about trends. When this information is practical and useful, we can interpret it (e.g. for making decision). Such interpretation is seen as a phenomenon of obtaining knowledge. In conclusion, we can say that a computerized information system has to processes data into useful information and can generate knowledge (in other words extract it from the data). Such systems are sometimes called intelligent information systems or more precisely *intelligent decision support system*.

4.1.1 Organized Collection of Data (Databases)

To create a computerized information system, it is essential to have the hardware and software to support the accumulation and handling of large data collections belonging to the organization. A core technology needed to create such data repositories is database technology. It is useful not only for data storing, but also for data analysis and processing, for information management, and for data visualization and presentation in practically all business and industrial contexts. A typical database presents its own data to the user as an informative model of a fragment of reality, for example, the user can see an airport database as an official online flight timetable of arrivals and departures. The database is not a standalone self-sufficient application, it must be created and administrated (managed) by using the *database management system* (DBMS); alternatively, it can be provided through other software that has the ability to communicate with its data set through specific interfaces. For example, a

standard database access method called *Open DataBase Connectivity* (ODBC) makes it possible to access data from any application, regardless of which DBMS it is handling. It is important that databases are networking products; this means that information systems computers and database computers are usually located in different places. User terminals can also connect to the information system from any desired location. Network operation is the basic style of exploitation of the IS.

Before creating a database for the information system, the designer builds a logical (conceptual) model of data, which will be stored in the database. There are many ways to compose such a model; the best recognized is *entity-relationship diagramming*. In such a text and graphic model the entity is something existing in the real world, regarding which data will be stored in the designed database. On the model of an entity, DBMS will generate tables, which will be written copies (records) of data items. An entity may be a physical object, e.g. a particular employee, building, or car and it may be a conceptual object, e.g. a job, a bank, or an automobile manufacturer. Each entity can be described by some properties, which have to be presented in an entity relationship diagram (ER diagram) as attributes of the entity. In the above example, the car is the entity and its model, doors number, color, transmission, number of seats, weight, and release date are the attributes. Each entity must have an attribute, which allows it to be uniquely identified. Such an attribute is called the primary key. In the above example, the social security number may be the primary key of an employee, and the vehicle registration number may by the primary key of a car. A relationship is the thing which allows the description of a substantive connection between different entities. In the above example, there is a relationship between the car and automobile manufacture. There may be a relationship between a bank and a building. The connection of entities is characterized by a multiplicity of relationship, e.g. one-to-one, one-to-many. The relationship may have attributes as well. An example of an ER diagram is shown in Figure 19.

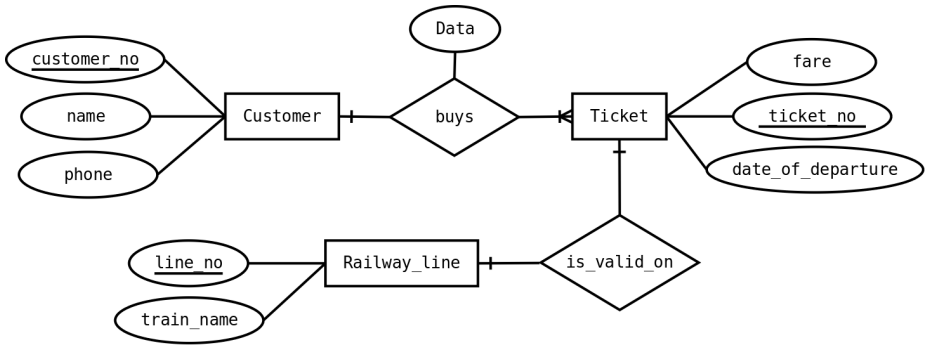

Figure 19: ER diagram illustrating the logical structure of database.

Based on the entity relationship diagram, the database administrator generates a set of related tables – it will be a database skeleton. After this, somebody will have to fill it in with real data. The presented data collection modeling style is closely associated with relational databases, the idea of which was proposed by E. F. Codd in 1970. Nowadays, informaticians have non-relational databases technologies, e.g. based on the concept of a document. It is noteworthy that older, hierarchical databases from mainframe era are still in use. For example, the MS Registry is a hierarchical database that provides data which are critical for Windows OS. Also, IBM Information Management System is the widely used commercial hierarchical database management system.

The database specialist has at his/her disposal many database management systems, which generally do not interact directly with one another, mostly for historical reasons. The *structured query language* (SQL) is a thing that unites all popular relational databases. It is a quaint original data definition and manipulation language, which is widely used for communication between the information system and the data collection. The standard SQL commands such as *Create*, *Select*, *Insert*, *Delete*, and *Update*, can be used by programmers to achieve everything that they needs to do with data. For example, to create a new table describing the car, a programmer can write the following SQL code:

```
CREATE TABLE CAR
(REGISTRATION_NUMBER CHAR(10) PRIMARY KEY,
DOORS_NUMBER SHORT,
SEATS_NUMBER SHORT,
COLOR CHAR(10),
RELEASE_DATE DATA);
```

The next fragment of SQL code may be used to insert some records into the new created table:

```
INSERT INTO CAR VALUES ('EBX 5525', 3, 5, 'BLACK', 2011);
INSERT INTO CAR VALUES ('NBH 7789, 5, 7, 'WHITE', 2014);
INSERT INTO CAR VALUES ('NDM 7879', 4, 5, 'YELLOW', 2015);
After the table has been filled in, it is possible to select some data from it:
SELECT * FROM CAR
WHERE RELEASE_DATE > 2012;
```

In the above example, the database will send in answer on such a SELECT request the list of data concerning cars that were manufactured after 2012. It is worth noting that the SQL code is sufficiently clear and can be read with understanding by a non-computer specialist. This is one of the most important advantages of the SQL language, because the specialists in the subject (non-informaticians) should shape the substantive question about the organization's data collections.

4.1.2 Discovering IS Functionality

To design an information system, informaticians need to understand the organizational structure of the data and the processes by which these data are generated, transformed and presented. A business or industrial organization is a *complex system*. It is not a simple task to fully comprehend the structure of the data circulating in an organization, which should be represented in the computerized information system. Every specialist working in the organization understands the functioning of the organization from his or her own perspective. An objective, impartial, and multilateral representation of the organization should be developed through targeted analytical activity. One classic way to do this is called *data flow diagramming*.

In information systems theory, the data flow [43] is the term represented by the path taken by data items within an organization moving between data stores, processes (operations), and outside entities, i.e. people, or external systems. Data flows are shown on diagram by a line with an arrowhead to indicate the direction of the flow; each data flow has a unique name indicating what data are being passed. Data flows and processes can be modeled at different levels of detail. The highest level (number 0) is called a context, because it shows the information system as a single process. By this convention, level 1 (as is shown in Figure 20) is designed to review the main functions of an information system, and levels 2 and above - for detailed analysis.

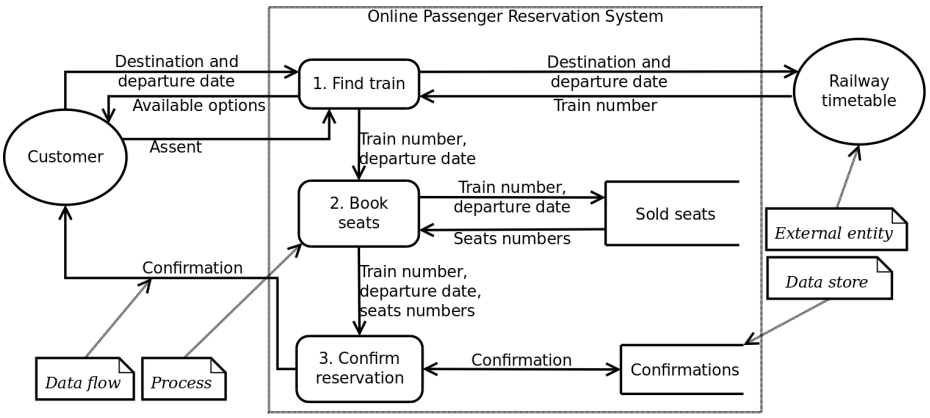

Figure 20: Data flow diagram for a reservation system of a railroad company (level 1).

With a detailed model of the flows of data, developers can proceed directly to the technical design and coding of system modules. In very simplistic terms, each process must have its own application window and the connected data flows will show which

values should be entered in a dialog box, and which results of the calculations are to be expected. Both data flow diagramming and the entity relational model are the main techniques of structured systems analysis and design method [44] (SSADM), which is associated with the waterfall model of the software development process. Initially, SSADM was released inside a *systems approach* [45] to the analysis and design of information systems.

4.2 Analysis and Modeling of Information Systems

The complexity of information system expresses the degree to which its components engage in organized structured interactions. Business systems, such as system of air traffic control at an airport, the banking system, or a production management system achieve the very high level of complexity. The increase in the number of components and in the number of relationships between parts of a system gives an enormous rise in the complexity of the collective behavior of a system. Prior to the design and development of a complex system, it is obligatory to carry out modeling and deep analysis of the future system, together with its possible context (environment). It should be emphasized that even sophisticated modeling does not easily capture the complexity of real business and industrial information systems. A human has a limited capacity to anticipate the behavior of relatively simple systems and cannot predict the behavior of complex systems at all. However, *complexity theory* [46] believes that complex phenomena can allow pattern predictions through modeling.

Models are used in software system engineering because of the impossibility or impracticality of creating experimental conditions in which a developer can directly examine outcomes. The generated model as a *conceptual representation* of a system will typically refer only to some aspects of particular system. This means that the systems analyst should generate several models to embrace the system from different perspectives. Methodical and planned information systems analysis is conducted currently using *information models* [47]. Nowadays, the unified information modeling language (UML [48]) released by the Object Management Group [49] in 1997 is preferred.

4.2.1 Brief Introduction to the Unified Modeling Language

Object-oriented modeling languages began to appear in mid-1970 as various methodologists experimented with different approaches to object-oriented analysis and design. This was quite long and "painful" process until it converges to UML. The main authors of UML were Grady Booch, Ivar Jacobson, and Jim Rumbaugh. They were the first to introduce in 1996 a unified, standard, text-and-graphical modeling

notation for IT professionals to interchange blueprints of software system architecture and design plans.

From the outset, UML was independent of programming languages and development processes. This facilitates communication between IT communities that work with essentially different development platforms and hardware and operating systems. UML has integrated the best practices of all object-oriented design of software. It is important that UML supports high-level design concepts such as patterns, frameworks, and collaborations. UML modeling is used for specifying, visualizing, constructing, and documenting software artifacts. UML provides [50] model elements such as concepts and semantics, and the notation, by which a visual rendering of model elements takes place. The major UML diagrams elements (as is shown in figure 21), through which designer expresses the structure and behavior of the information system, are described below:

- Two-dimensional graphical objects (some of them are partitioned) with changeable size, which are containers for text, icons, and other graphical objects;
- Path and area symbols (e.g. time line segments with attached endpoints or rectangular system boundaries); they may also have icons and text qualifiers;
- Text objects, especially names (uniquely identifies some model element), keywords (text enclosed within Latin quotation marks «...» to express some concept), expressions (a linguistic formulas), and labels.

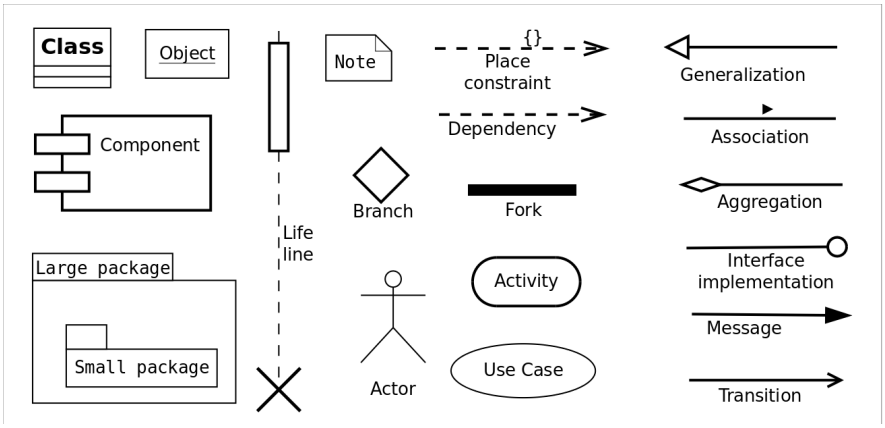

Figure 21: Some UML model elements.

A typical UML model is a set of diagrams; each UML diagram is an insight (view) into the model of a system from the viewpoint of a particular stakeholder. The diagram provides a partial, incomplete representation of the system and has to be consistent semantically with other views. In the UML, there are nine standard diagrams, which

can represent static and dynamic views of an information system; use case, class, object, component, and deployment diagrams constitute the static set, and sequence, collaboration, statechart, and activity diagrams – the dynamic set. Below are shown some cumulative information about more popular UML diagrams.

Use Case Diagrams

These diagrams mould an image of interactions of actors with system components. Here, an actor is an external entity that cooperates with a software system to achieve its own goals. Formally, a use case diagram is the graph of actors and use cases enclosed by a system boundary and focused on what a particular actor is doing in a particular action, as it is shown in Figure 22. The main idea of the use-case diagram is to visualize the functional requirements of a system. Typically, only high-level functions of the system are shown and non-functional requirements are not presented in this view. In practice, use case diagrams are used to represent the functionality of system from a top-down perspective. Further particularities are added later to make clear the details of the system's behavior. Use cases are graphically shown as ellipses.

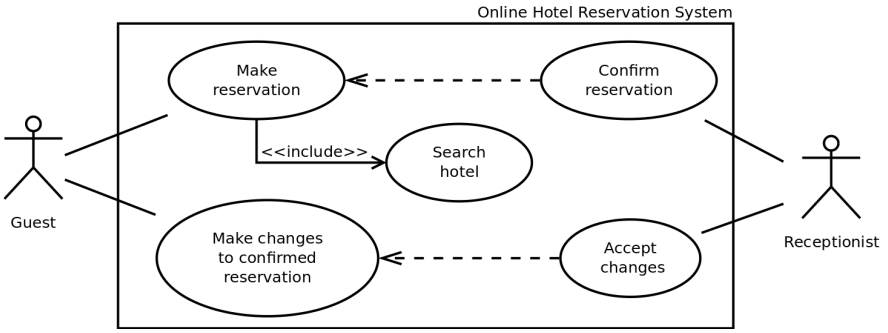

Figure 22: Example of a use case diagram for N system.

Class Diagrams

The class diagram is the graph of entities together with the relationships between them. In fact, the diagram shows the static structures of the information system. Often, class modeling is made at a level of subsystems or system modules, because modeling of a complete system may be impractical due to the huge number of classes. The main concepts of this view are class, association, generalization, dependency, realization, and interface. Here, a class is an internal entity of a software system, which describes the set of objects that share the same attributes, operations, relationships, and semantics. Classes are graphically shown as partitioned boxes. At first, a conceptual class diagram is created to display only the logical structure of system (as is shown in Figure 23). During this first stage, the analyst and designer can choose an optimal set

of domain classes to fulfill the user requirements. Then, the expanded form of class diagram is created to display the physical structure of system. During this second stage, the designer adds to diagram special classes to fulfill the implementation platform requirements. Programmers typically deal with a physical class diagram. A conceptual class diagram has to be approved by a client representative.

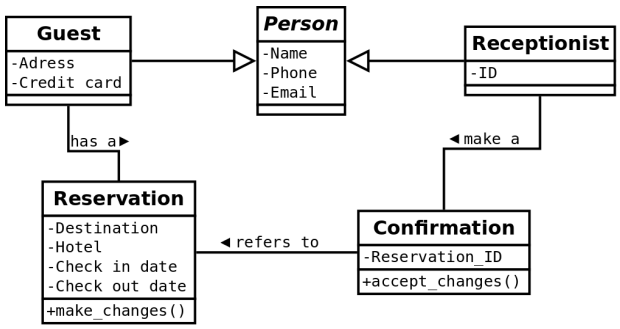

Figure 23: An example of a conceptual class diagram for N system.

Sequence diagram

A sequence diagram links use case and class diagrams. It shows a thorough information flow between objects for a specific use case. In other words, it shows the sequence of calls between the different objects modeling the interactions of these objects during the realization of the use case. The sequence of calls and messages is shown on the sequence diagram in the time order that they occur. The interaction starts always near the top of the sequence diagram and ends at the bottom. Sequence diagrams are usually used to model and examine usage scenarios, the logic of methods, and the logic of services (high-level methods invoked by a wide variety of clients). An example of a sequence diagram is shown in Figure 24.

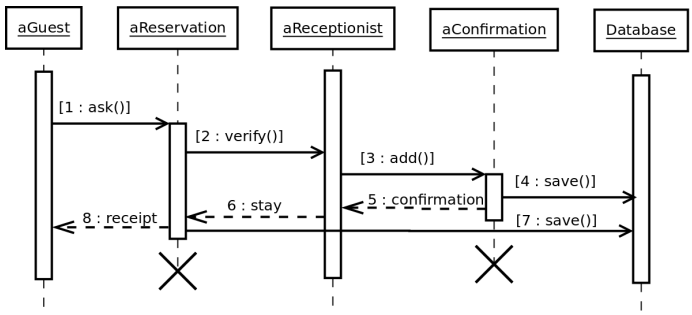

Figure 24: An example of a sequence diagram for N system.

Activity diagram

An activity diagram represents one of most popular UML views. It is a kind of flow chart and shows in a procedural style the flow of control between class objects while processing an activity. The activity diagram can express not only sequential, but also branched or concurrent flows. This diagram can be used at a high level of generalization to model business processes realized by means of a software system, or at a low level to model an internal component's or class's actions. Activity diagrams are useful for examining the logic captured by a single use case or a usage scenario by using both forward and reverse engineering techniques. The relatively simple notation makes the activity diagram a useful tool in customer relations, especially in the context of the negotiation of the business aspects of software systems. An example of an activity diagram is shown in Figure 25.

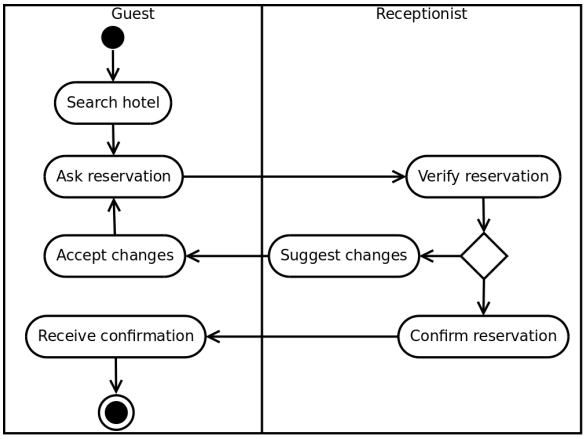

Figure 25: An example of an activity diagram for N system.

Component diagram

A component diagram presents a physical view of the software system. It is a map of software components together with their mutual relations (dependencies) within the system. Here the component is a software element (a structured class) or a module, which may be designed independently (separately of system project) and can be deployed independently. An activity diagram can be used both at a high level of information system modules (logical components) and at a low level of platform dependent component packages (physical components), e.g. containers in .NET or packages in Java technology. Recently, component-based software engineering has gained a great deal of popularity. Components are not only the basis for .NET and Java platforms. For example, Linux specialists intensively use components in KDE and GNOME projects. Nowadays, larger pieces of a system's functionality may be

assembled by reusing (ready-to-use) components. It is a very popular manipulation in the production of computer game software.

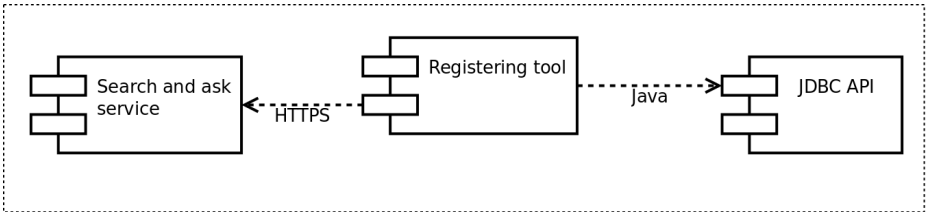

Figure 26: An example of a deployment diagram for N system.

4.2.2 Modeling the Architecture of Information System

The integration of all system components and control of the system interfaces is made by system engineers in accordance with the approved architecture of system. At the analysis and modeling stage this activity should be prepared by the introduction of some high-level logical constructs. These architectural constructs require a lot of mental effort at a high level of abstraction. This is why software architects tend to reuse successful architectural decisions, e.g. via the architectural frameworks [51]. Some examples of such general-purpose architecture frameworks are TOGAF, MODAF, RM-ODP, and Kruchten's 4+1 View Model. The organization domain has its own enterprise architecture frameworks, e.g. government frameworks, consortia-developed frameworks, and defense industry frameworks. The area of software architecture for information system is rich in detailed information. In getting acquainted with this area of expertise, one can familiarize oneself with the idea of architectural views.

It makes sense to classify UML diagrams against the 4 + 1 Architectural View Model, which was introduced by Philippe Kruchten [52] for describing software-intensive systems. This model is commonly used to describe information systems from the viewpoint of different stakeholders, e.g. end-users, development team members, and project managers. The model provides four fundamental views – logical, implementation or development, process and deployment views. Furthermore, it provides usage scenarios or use cases which serve to describe architectural elements (system objects and processes) and to illustrate and validate the architecture design. Logical and process views deal with conceptual architecture models, when implementation and deployment views are known as physical architecture models. Here are some very short, simplified characteristics of these five views with notes about related UML diagrams:

– The first, logical view presents end-user functionality. The logical architecture of information system has to support the functional requirements specified under

the project. The most important UML model for this view is the class diagram, a supporting role relative to its play sequence and communication diagrams.

- The second, implementation view presents the developer's perspective on the software system; this view is intended for software (code) management. In this architecture, system components are described by means of UML component and package diagrams.
- The third, process view is dedicated to system integrators. Such information system parameters as performance, scalability, concurrency, throughput, and others are modeled and analyzed in this view. The UML activity diagram is used to model this perspective, i.e. dynamic aspects of the information system – computational processes and threads complete with communication between them.
- The fourth, deployment view is intended for system engineering. System engineers deal with the topology of software components, with the delivery and installation of modules on the physical layer. They manage physical communication between these components as well. Most often the UML deployment diagram is used to analyze this perspective.
- The separate, fifth view utilizes scenarios or more general use cases view and integrates the four previous views. The usage scenarios have to represent in the architecture the most important requirements, which are presented by means of object scenario diagrams and object interaction diagrams.

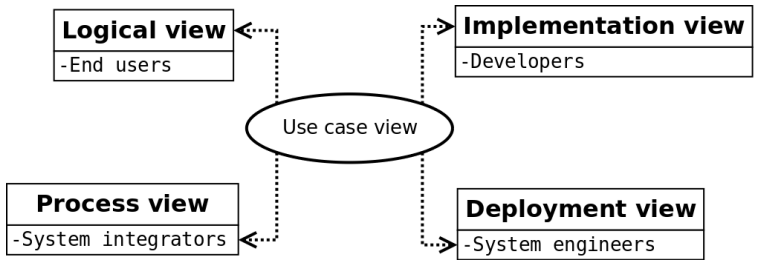

Figure 27: The 4+1 Architectural View Model.

An architecture-centered approach has been in the last several years the standard approach to enterprise system development and integration, especially for complex and critical enterprise systems. Please note that the field of enterprise architecture is still evolving. Nowadays, it seems that enterprise architecture is a path, not a destination.

4.3 Design of Business Information Systems

Business information systems [53] are the combination of people, information technology, and business processes to successfully complete business objectives. Nowadays, most professions make use of business information systems. They provide excellent conditions for information activities not only in manufacturing, marketing, trading, accounting, and finance, but also in science, healthcare, education, and even in entertainment. One of the most important factors of successful information systems is good design. It is necessary that such a system not only has excellent technical design, but also analytical design (i.e. optimal adaptation to the specificity of the profession), graphic design (i.e. visual perfection), and ergonomic design (i.e. high usability of software). Present-day businesses are driven by the information they offer using information technologies and particularly the web. Professional information systems support the processes of enterprise information exchange, so their perfection largely determines the success of a business.

Each information system is planned to improve the business in someway. Before making such an improvement, it is very important to recognize the current business situation, especially from the perspective of business processes. Business analysts, and with them managers, system engineers and developers use business process modeling for this purpose, i.e. some technique to diagram business processes. The visualizing (diagramming) of the current business process as-is helps in understanding of a cause of a failure and possible paths for development by stakeholders. After studying the business process model and before developing the renovated information system, business analysts have to diagram a future process. In the given context, a business process should be understood as a flow of activities. Each of such activities have to represent the work of a person or an internal system, so one can think about the progress of work or about the workflow. In conclusion, the attributes of the business process are:

- Task – the least of process elements, a logically separated portion of the work that has been assigned or done as part of the participants duties;
- Activity – a task or set of tasks carried out in order to produce a deliverable;
- Participants – people or systems that give input to, or perform tasks within a process;
- Roles – participant's positions, an abstraction that links participants and performed tasks or activities;
- Flow control or sequence to cause a process – the logical order in which tasks are performed;
- Events (or other words triggers) – occurrences and appearances that direct a process sequence, inter alia, cause a process to start or to end.

A logically complete hierarchy associated with business process modeling may be presented as follows: cycle, process, subprocess, activity, and task. The introduction

of an intermediate level of a subprocess is needed to compartmentalize and focus on specific process segments.

4.3.1 Business Process Modeling Notation

The first standard *business process-modeling notation* (BPMN 1.0) was released to the public in 2004 [54]. From the very beginning, this notation was created for a broad audience, i.e. for business users, business analysts, technical developers, and business people managing real business processes. In fact, BPMN defines a business process diagram, which is based on a well-known flowcharting technique modified for creating better graphical representations of business process operations (activities). In this notation, the business process is a flow of tasks and activities, which are performed by the process participants in roles. These tasks and activities are organized in specified chains directed by events and measured in time. Therefore, a business process model is a representation of a set of tasks and activities where some pairs of them are connected by links. This is consistent with the definition of graph in the mathematical meaning, i.e. tasks and activities may be treated as objects or the graph's vertices.

A basic set of BPMN symbols (as is shown in Figure 28) includes only three flow objects – *event*, *activity*, and *gateway*. An event symbol (circle) indicates an effect on the flow of the process. This effect usually has a cause (trigger) and an impact (result). There are three types of event symbols, based on when they affect the flow: start, intermediate, and end event symbols. An activity symbol (rounded rectangle) is a representation of work that is performed by a business (e.g. by company, a company department, or a particular worker). There are two types of BPMN activity symbols to distinguish the divisible or compound (indivisible) character of real business activities; the types of activities are task and subprocess. A gateway symbol (diamond) marks traditional business decisions, as well as the forking and merging of sequence flows. The type of gateway depends on the diamonds shape's internal marker, which indicates the logical operation on the connected flows.

Besides flow objects, BPMN defines connectors, which create the skeletal structure of the business process. There are three *connectors*: sequence flow, message flow, and association. A sequence flow show the order that activities and tasks will be performed. A message flow shows the messages between participants, and finally an association shows data associated with flow objects. In addition to this, BPMN supports the *swimlane* concept as an organizational entity, which is realized by pool and lane symbols. The pool presents a self-contained sequence of activities, i.e. individual process. It is important that only message flows may cross its boundary. The lane is a part of a pool and, in practice, is used to show who is responsible for specific work or to which group activities belong a specific business function.

BPMN version 2.0 has been available since 2011 [55]. This defines the notation, metamodel, interchange format, and includes some essentially new features. For example, it enables the provision of XML schemes for model transformation and in such a way, that orient BPMN toward business modeling and executive decision support. Some very new concepts were introduced, e.g. a choreography, which represents an expected behavior between two or more participants of a business process.

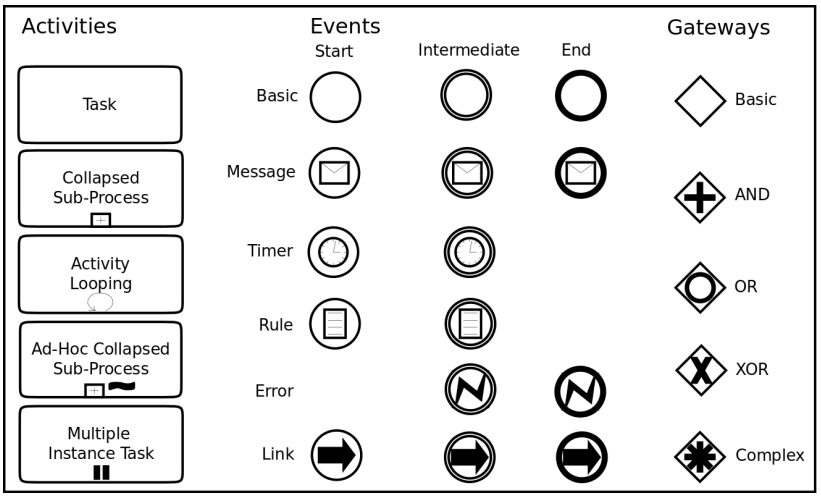

Figure 28: A basic set of BPMN symbols.

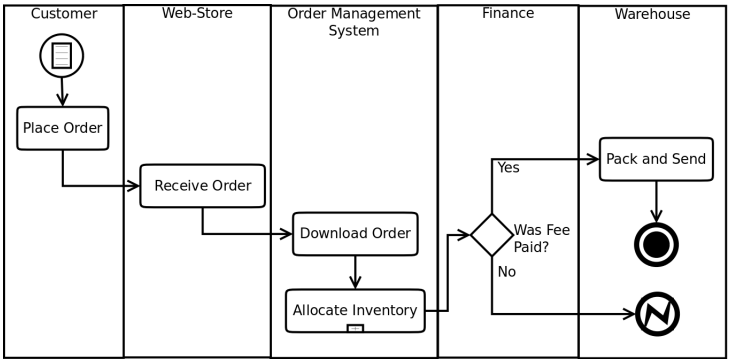

Figure 29: An example of simple business process model.

An example of a business process model is shown in Figure 29. To create such a model one needs a software tool for business process modeling. The above model has been developed in a DIA environment, which is good for educational purposes, but not for real projects of information systems. On the software market, there are many proposals that implement the full range of BPMN 2.0 (Business Process Model and Notation). Bizagi BPM Suite [56] seems to be the most progressive.

4.3.2 Business Process Reengineering

Business process reengineering [57] (BPR) is the idea of business modernization through redesigning the company processes and the synchronous deployment of information technology. In practice, this means discovering and eliminating wasted or redundant effort and improving processes efficiency [58]. This is done in parallel with the implementation of new software, which will be responsible for maintaining an optimum discipline for the completion of tasks and replace workers in such jobs that a computer can perform better. The business processes which could be redesigned in such a way, cover everything from production to sales, to customer service. It is significant that during the reengineering revolutionary change is expected, not evolutionary improvement. Business process reengineering is particularly important for companies to achieve maximum improvements in their critical areas (e.g. high quality of service or low costs of production) through the use of information technology. This entails the need to rethink not only structure of business processes, but also management style, organizational structure, job definitions, workflow, and so on.

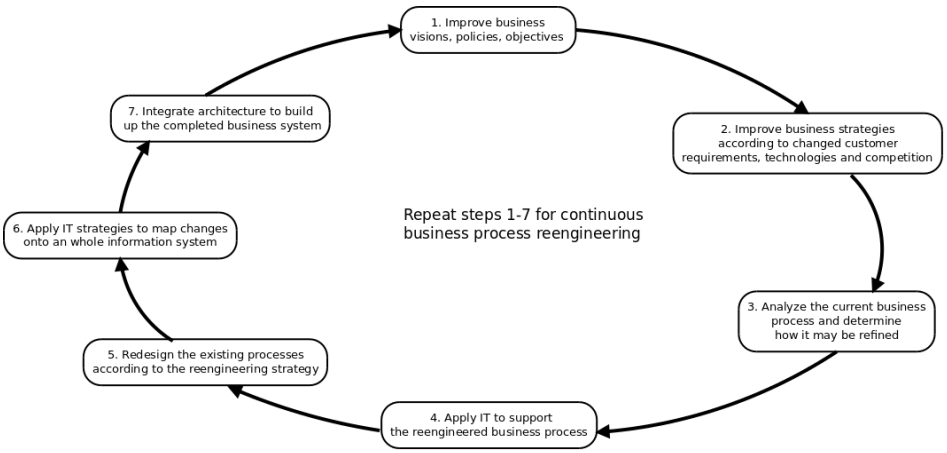

Figure 30: Business process reengineering cycle.

In accordance with a traditional, structural approach to business engineering, the structure of business and the plan of processes were directly based on business strategy. IT implementation was planned as a last stage of the establishment of the company. Business process reengineering has put forward a process-oriented approach, which has to eliminate some problems typical for business hierarchical structures. Namely, BPR changes the hierarchical relationships between management and employers into an interactive process, which is readily computerized because it follows precise, well-defined procedures. At present, information technology is considered to be the most important factor in the development of new forms of collaboration within an organization and between organizations as well. Among these technologies the following should be mentioned: telecommunication networks, wireless data communication, mobile computing, cloud computing, voice over internet protocol (VoIP), tracking systems, and shared databases.

A big improvement in business processes arose in the 1990s based on the idea of a *workflow management system*, which was positioned as a system that defines, creates and manages the execution of workflows using software [59]. The core of such software is a workflow engine. It is able to understand process definition, act together with workflow participants and invokes the use of necessary external software tools and applications. In figure 31 a workflow reference model is shown, which representents the architecture of a workflow management system identifying system interfaces. It is worth recalling that workflow is defined as a sequence of activities through which a piece of work is completed.

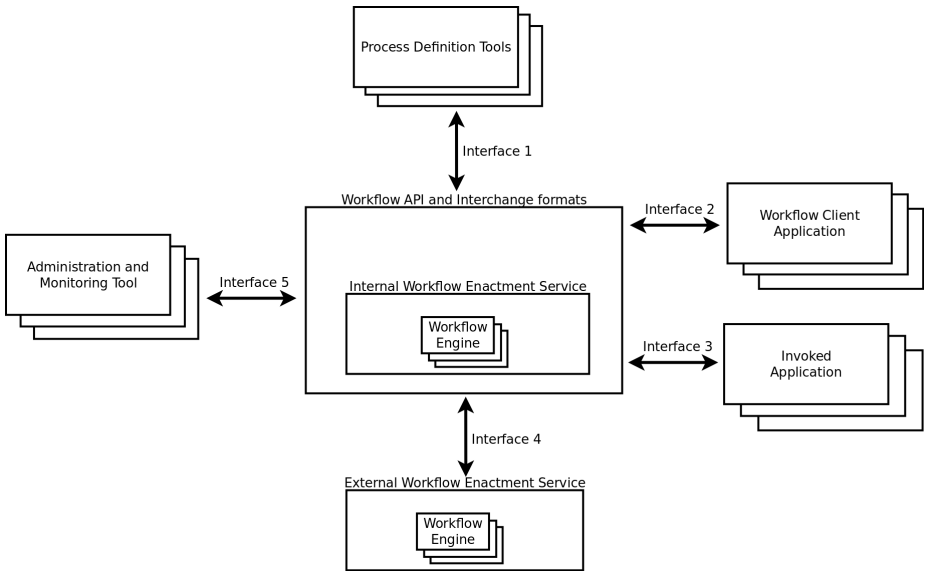

Figure 31: The workflow reference model.

In the same period, another idea for business process design and improvement was popularized – enterprise resource planning (ERP). It is defined as a suite of integrated business applications; here, software integration is the key. Generally, ERP modules carry out common business functions such as information lookups, accounting, banking, inventory control, payroll and taxes, order processing, workflow approvals, or customer care. A common set of data and a synchronization mechanism helps in integrating these applications for decision making and planning. Nowadays, enterprise resource planning is an established category of business management software. Some typical modules of ERP are shown in Figure 32.

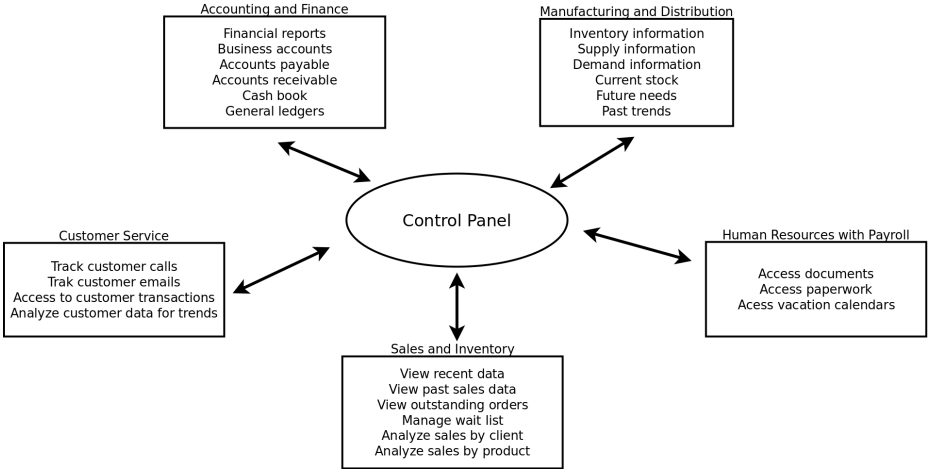

Figure 32: Typical ERP modules.

One of the most important factors in auspicious business process reengineering is considering the right IT infrastructure [60].

The IT infrastructure of organization

The IT infrastructure of organization is planned together with applications portfolios of the organization and may not be changed in a random manner. Here are some IT planning approaches:

- Business-led approach, in which the business strategy defines the IT investment plan;
- Organizational approach, in which the IT investment plan is a compromise between stakeholders;
- Administrative approach, in which a steering committee establishes the IT investment plan;
- Method-driven approach, in which the IS needs are identified using special techniques.

Generally, IT planning consists of four stages: the strategic IT plan, an analysis of information requirements, an allocation of resources, and a project plan. It all is carried out in accordance with the strategy of the company and is intended to ensure the competitive advantage of organization. The main result of the planning is the set up an applications portfolio, through which an organization intends to manage its business. The overall structure of information systems in an organization is called *information technology architecture*. Nowadays, the IT architecture of the organization is designed around business processes whereas departmental hierarchy is not relevant in this context. An example of IT architectural decisions may be the choice between centralized, distributed, or blended computing. Another example would be the choice of the network working group configuration.

4.3.3 Software Measurement

The design of business information systems is not possible without a measure of software, i.e. without expressing software in impartial numbers. The purpose of *software measurement* [61] may be related to the prediction of software complexity or usage, to the control of production hours or costs, and to the assessment of software usability, quality, or security. Software measurement objectives may concern the assessment of the status of IT projects, products, processes, or resources, but also trend identification or self-assurance action. For stakeholders, it is important to know the planned and actual size of a software system, but also the numerical indices of software quality and of the progress of a software project.

Knowing the size of software is important for comparing different systems together, e.g. to evaluate the effort of development teams or to estimate the cost of development work. The oldest approach to measure the size of software is to count the lines of source code [62]. This size metric is called *lines of code* (LOC); in which physical or logical lines of code are distinguished. For larger systems, thousands of lines of code (KLOC) are also used. Modern software systems reach the size of tens of millions of lines of code. As practice shows, the same system developed with different programming languages will have different LOC results, so to compare systems written in different languages, we must take into account the possibility of a language to write concise code. Unfortunately, in order to calculate LOC we have to wait until the system is entirely implemented. This disqualifies the LOC-metric from forecasting IT projects, e.g. prediction of cost and effort of new software development.

A different approach to measure the size of software is based on *function point analysis* [63]. Allan Albrecht at IBM published it in 1979. Function points do not calculate the size of software directly. They measure the functionality offered by a software system. The trick here is that on the one hand, functions represent sets of executable statements (source code logical fragments) that perform certain tasks, but on the other hand, they are visible from the user point of view as services. This allows

for a calculation of the function points before a system is developed. It is enough to have conceptual and architectural design. To be language independent, the function points are calculated as a weighted sum of five components that define the application functionality provided to the user. The components are broken into two categories:

- Data functions (reflect data requirements);
  - Internal Logical Files (files maintained by an application);
  - External Interface Files (files accessed by an application but not maintained by it);
- Transactional functions (reflect needed data processing);
  - External Inputs (e.g. forms or questionnaires);
  - External Outputs (e.g. reports);
  - External Inquiries (e.g. sending request to external search engine).

The row (unadjusted) function points are converted according to functional complexity. It is made based on the number of data types or file types referenced. After that, the value adjustment factor is used to take into account the degree of influence of fourteen general system characteristics, such as heavily used configurations, transaction rate, or complex processing (a five level scale from no-influence to strong-influence throughout). The result of the calculations will be established in form of an adjusted function point count, which will represent the size and complexity of a software system.

IT managers treat *software sizing* as a basis for implementing their own software project management activities, i.e. project planning, implementation of project components in a timely manner, and review of project activities. The most important data for managers concerns *software development effort estimation*, which is the process of predicting the most realistic amount of endeavor expressed in terms of person-hours required to develop the software.

Another important aspect of software measurement applies to defects-based metrics; defect density is one of the indicators of the maturity of the software. Software developers and testers use a rate-metric, which describes how many defects occur for each functionality unit of a system. For some critical systems, the *mean time between failure* (MTBF) indicators is applied.

4.4 Human Factors in Information Systems

Complex information systems cause difficulties not only to their developers, but also to end users [64]. From the users' perspective, IS may have overly complex dependencies, a lack of transparency, intricate control logic, confusing terminology, complicated navigation, and so on. Human factor engineers (professional ergonomists) believe that the workplace has to be essentially matched to the worker, because the better the match the higher the level of worker efficiency. In the case of the design of information

systems, it is not about physical ergonomics, but most about organizational and cognitive ergonomics, e.g. when designing human-system interfaces developers must account for the cognitive limits of future users. Developers have to optimize the organizational structures, policies, and processes of such socio-technical systems [65]. It is vitally important to create formal methods for vague business processes in complex information systems. The aim is the reducing of the probability of human errors.

The key objectives of applying human factors to the design of information systems are to improve effectiveness, efficiency, safety, and the satisfaction of the end users. This can be achieved by reducing fatigue and of the learning curve and by ensuring operability (in fact, by meeting the user's needs and wants). Software developers and system engineers have to understand that in fact, with information systems they design and create the information environment for the end users. Because of this, they control what information will be accessible and how it will be presented to the end user. Every design decision has implications for users on their knowledge, attention, memory, reactions, and so on. It is clear that in traditional software development processes only after the implementation of the system remains does it remain to be seen whether these requirements are met. In order not to make serious errors in design, designers of information systems need to understand users – who are they, what are their working qualities, what are their professional needs.

It is worth noting that the interaction of users with information technologies is cognitive. This means that IT designers need to take into account the cognitive processes involved in interaction and the cognitive limitations of real users. For IT systems, the core cognitive aspects of users are perception and recognition, attention, memory, and learning. Designers have to focus on these aspects when creating an information system. The discipline known as *Human Computer Interaction* (HCI) embraces the subject of user aspects of software systems most comprehensively, because the goals of HCI is to develop or improve the safety, utility, effectiveness, efficiency, and appeal of systems that include computers [66]. HCI understand human-computer interface to be when people come together with computer-based systems not only as a physical interface (i.e. screens, menus, buttons, sliders, switches), but also as a logical interface. It is a model of the system from the point of view of the user, in other words, the set of available functions and behaviors.

4.4.1 Human Computer Interaction in Software Development

The special interest group on human-computer interaction of the Association for Computing Machinery (ACM SIGCHI) [67] defines HCI as an inter-disciplinary area of knowledge, which is concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them. According to HCI [68] the evolution of human-computer interfaces

has passed through some important stages of development, resulting in today's design methodology, which takes into account not only the user's needs, but also the cognitive characteristics of users. In the beginning, in the 1950s interface was realized at the hardware level. First the systems were not interactive and were available for hardware engineers only. In the 1960s-1970s, the first software interfaces which were accessible for a programmer and which were presented for end users at the terminal level with a command language appeared. During this period, the issues concerning the end-users and software quality of use were not at all important. The creation of user interfaces was not clearly separated as a significant aspect of the design of software systems. In the 1980s, the first interfaces at the interaction dialogue level were designed (graphical user interfaces, later with use of multimedia), but still the quality of communication between human and computer restrained the spread of IT. Relatively recently, at the beginning of the 21st century human-computer interfaces have become pervasive. Nowadays, most end-users of software systems are neither computer specialists nor informaticians. The quality of use has become vital to the success of commercial software. This means that software designers cannot ignore the interaction between the user and the product. Interactive dialogues should equally be in the center of their attention as, say, a database or software architecture.

New specializations among designers became a kind of response to the needs of the IT industry in the context of the achievements of HCI. The first group, *interaction designers* are professionals who are involved in the design of all the interactive aspects of a software product. Interaction designers have to convert business needs and software requirements into user-centered software system experiences. Together with other stakeholders, they have to create user interface mockups or wireframes (prototypes), user flows, and functional specifications; as a result, their work maximizes the quality of use of the software. *Usability engineers* (the second new specialization) perform the measurement of the quality of use (usability or human factors). These people are experts in the relationship between people and software systems, and concentrate on product evaluation. Usability engineers may also be called *user experience specialists*. They ensure that software is user-friendly and the average user can understand and use the software system.

It is worth noting that the terminology concerning human factors and HCI in information systems is fuzzy. The terms are mirrored or rather loosely defined. A full taxonomy was never published. For example, the very popular concept *software usability* is merging semantically in itself the quality of use of a software system. According to the definition from the ISO standard 9241 [69], usability is the "effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments". Effectiveness means the "accuracy and completeness with which specified users can achieve specified goals in particular environments", the efficiency means the "resources expended in relation to the accuracy and completeness of goals achieved", and the satisfaction means the "comfort and acceptability of the work system to its users and other people affected

by its use". Such a non-precise and non-technical definition allows for different interpretations of this concept and the results of measurement of usability according to different methodologies are often not comparable. Nevertheless, there are some successful techniques for designing human-computer interaction and ergonomic user interfaces. Achievements in this field rely more on heuristic techniques, rather than mathematical models.

4.4.2 User Centered Development Methodology

Traditional system-centered design places emphasis on the functionality of software systems [70]; with such an approach, developers add user interfaces to data structures and to business logic at the end of the design process. One can say that system-centered design accents the correct software rather than usable software. This means that users have to familiarize and adapt themselves to the software product. At the turn of the 20th and 21st centuries, the situation has changed fundamentally – user-centered design is needed and user interfaces (UI) have became more important. A new product design philosophy was introduced. This is known as "user-centered". User centered approach highlights the users' tasks. It is possible only if the user participates in the early stages of software development. Under this approach, the early prototyping of user interfaces is vitally necessary. This forces the iterative development of user interfaces by a UI-designer and a UI-programmer. Such a user centered development methodology involves users in the design, testing and revision processes, in which the monitoring of usability of software becomes possible.

ISO 9241-210:2010 describes the widely understood human-centered design process for interactive systems. Essential activities in *user-centered design* [71-72] of information systems are:

- Understanding and specifying the context of use of information systems, especially understanding the parent organization and its business;
- Specifying the users and organizational requirements;
- Producing design solutions, preparation of prototypes (mockups or wireframes);
- Evaluating design with users against specified requirements (correction of requirements is permitted).

The basis of user-centered design is a multidisciplinary design team, which provides among other things the participation of non-technical professionals. Together with the active participation of users in all stages of the life cycle of a system, such a team guarantees a non-technocratic approach to information systems. User-centered design is widely practiced as an approach to design of the business information systems, especially to the design of user interfaces and dialogues for such systems. The designer focuses on the user tasks and goals, and on understanding the user environment – not only physical, but also organizational and social.