Software engineering is an area of computer science that deals with the construction of software systems, i.e. a number of interacting software components that are so large or complex that construction of such systems requires participation of a dedicated development team or several interacting development teams [33]. Under the developers we imply not only programmers or coders, but also system analysts, project managers, testers, system architects, documenters, quality assurance people and maintenance personnel. That is quite a big team, which focuses on the production of a software product in an existing environment of software systems of the customer. Software engineering is therefore vital in terms of management of all levels and all aspects of software development, such as requirements analysis and specification, preliminary and detailed design, implementation and testing, integration, transfer to client and customer support.

In Software Engineering, V. Lipaev, the patriarch of the Russian software engineering, gives the following definition: “The software engineering refers to a set of tasks, methods, tools and technologies, which is intended for design and implementation of complex, scalable, replicable, high-quality software systems, possibly including a database” [35]. Each word in this definition is specifically meaningful for crisis conditions and mission-critical systems.

According to the above definition, software engineering as a branch of science is precisely aimed at creating mission-critical software systems. Because these software systems are complex, it is not economically feasible to immediately replace all their components that are used by the large-scale software subsystems. More likely, particular subsystems of a large-scale system are reused, since they are high quality by design and often replicable. Examples of such solutions include Microsoft Dynamics and Oracle e-Business Suite. Under scalability, we mean a gradual decrease in performance in case of intensively growing system load. In addition, these systems should be reliable, predictable, ergonomic and maintainable, i.e. they should be developed to provide a sufficiently flexible and relatively evolutionary interaction with the user at the stage of pilot and commercial operation. The above definition of software engineering includes “design and implementation”, i.e., it follows the core of the software development lifecycle. An important addition is that a large-scale software system often contains a database. These databases can be heterogeneous, i.e., they can include object components, so they are not purely relational. For example, recent versions of Oracle DBMS are referred to as object-relational. Other “new generation” databases, such as O2, Orion, etc., combine relational and object-oriented paradigms.

Software systems development generally includes such concepts as software project and software product. Currently, we are going to focus on the software product, i.e. to look at the lifecycle from the perspective of software system architect. Software project is the perspective of the project manager, who is responsible for managing the project team, communication of the people involved in the project, time and cost. On the other hand, concerning software engineering we can generally discuss product development for a specific customer. However, we should plan all processes and technologies related to the software development in such a way that it is possible to supply the product to a wider audience of consumers, and ideally to make it reproducible and commercial off-the-shelf (COTS). It is advisable to provide a high percentage of reusable elements of the product, such as code, documentation, database structure and software architecture, so that after final release is ready any possible customer could spent minimum time, budget and labor to customize it. This approach is essential for crisis management of software production.

At the initial stage of product development, as a rule, there is as little as only a concept or an idea. Of course, at least a minimum amount of initial investment is required. However, in case of project development, a draft, high-level plan should exist, which delimits such key indices, as budget, functionality and time, and there should exist a certain customer, i.e. a specific shareholder who will provide funding for the project. In some cases, we can build software by mixed development, so that a proprietary system can later be transformed into a relatively open solution, which suits a wider range of customers.

What are the main features of a software product? First, it should have a certain commercial value. This means that the product is intended to solve a specific problem for a particular class of end users, clients or consumers. Thus, the product should be supplied to the market in order to meet specific needs and custom business objectives. What are the examples of such software products? Often these are physical objects, such as an information media, e.g. DVD, CD, etc. However, these might also be non-material objects. In any case, a software product should include proper customer documentation, and a number of legal agreements, such as a license, a partnership agreement, and so forth. A software product can also be offered as a service for deployment, customization, maintenance, or consulting.

Software products can be classified on different bases. One type of classification is the scale or scope: it is for personal use, non-commercial, or comes as a commercial COTS product for a wide range of organizations or individuals. Another method of classification is the purpose of end user. In this case, we can divide products into specialized software aimed at solving relatively specific tasks, such as software developed to solve astronomical
problems, laser ranging, and more general-purpose products, such as the operating system, office software etc. One more type of classification is the degree to which the product is open for interfacing with others. In this respect, the software can be classified as ready-made proprietary products and customizable component-based products, such as an API library.

Any software development takes place according to a certain lifecycle pattern, which includes a sequence of stages; it generally begins with basic concepts and ideas, and it generally ends with the product retirement.

The concept of lifecycle is applicable to any kind of systems, for example, such systems as skyscraper buildings; however, the lifecycle of software products has its own distinct characteristics. Software development is usually a gradual evolution, which starts from the initial concept or a rather abstract idea. It is further elaborated into a piece of software, which includes not only code but also a large number of documentary artifacts, such as inline code documentation, specific documentation for software administrators who setup, install and maintain the product, and others. The lifecycle of a software product ends at the stage of retirement, which follows the maintenance.

Each stage of the lifecycle is completed after developing a certain artifact for the system. Depending on the model lifecycle, after each lifecycle loop, the system can be either fully functional or not full-featured. Each stage ends with the production of documentation, which may include global artifacts, such as a project plan, a test plan, an implementation plan, a maintenance plan and more specific documents, such as use cases, Administrator's Guide, Quick Start Guide, high-level requirements of the product, or more detailed requirements in the form of technical specifications. Size, nature and elaboration of the documentation artifacts depend on the scale and scope of the software. Of course, each stage of the production of software should be clearly defined by its start and end time points, as well as by deliverables for the next stage in terms of code and documentation. In practice, however, mission-critical software product development is often more complicated; however, the software engineering approach, even in a crisis, requires a thoroughly defined lifecycle, each stage of which should yield to new product artifacts including new documentation.

To study the lifecycle of mission-critical software systems, we should first understand how software development is organized, i.e., how the critical software development processes are associated with the lifecycle stages. Failing to understand the lifecycle in general, we can hardly speak of any systematic organization and management of these processes. Of course, the most successful projects have to draw conclusions and replicate the principles that led us to success. We should study and improve the practices and techniques that allow for efficient and systematic development of mission-critical software in order to improve the product operational quality, user interface, documentation and all the related processes that underlie the lifecycle. We should do these improvements based on the analysis of the historical data for the previous projects. These should guide us in the future project planning and creating other “global” documents, such as testing plan, integration plan, implementation plan, maintenance plan etc. We should also use the error reports, and other documents created during software product testing in order to improve the processes and make them manageable even in a crisis. In this respect, thorough study of product development lifecycle provides an important basis for forming anti-crisis patterns of software development, which allows more accurate planning, monitoring and reproducing the processes of the lifecycle. Therewith, we need a methodology, which is designed for scalable teams of developers, and which makes it possible to adjust and adapt the lifecycle in a crisis, and to achieve an adequate product quality. Each software development project should follow a certain procedure, i.e., a methodology, which includes all the previous experience and historical data available, and which must adapt to the nature, size and scope of the particular customer and the specific conditions of production, including specific crisis conditions. Oftentimes, the customer already has a certain and a unique combination of the hardware and software environment in which the new software product must be implemented. This is particularly important in relation to large-scale and mission-critical systems, due to a large number of relationships and a significant complexity of the software environment. Thus, thorough analysis and planning of the lifecycle is a requirement for any mission-critical software product, and it is essential in case of crisis.

When talking about the lifecycle, we need to make some important remarks. First, we have to say that the lifecycle processes, which embrace each stage of the software product development, include a number of parties. At a minimum, these are the customer's representatives and representatives of the developer and management. The representatives of the customer, who are going to accept the product, are often technically competent people. As a result, they are doing the quality assurance. The developer's side includes a wide range of experts, such as analysts, risk managers, designers, system architects, documenters, programmers, testers, maintenance specialists and more. The management side often includes project manager, product manager and others. It is clear that the client’s and the developer’s employees have very different business goals. Similarly, the attitudes of the management of the client and developer differ in many respects.
These different perspectives are due to differences in product expectations in terms of functionality, design constraints, deadlines, cost, functionality, and they often result from different interpretations of certain terms and conditions. The customers expect software product to be good enough for assisting in their business needs; however, they often can be unaware of sophisticated technologies that support the features of the software product, which are clearly understood by the developers. In this respect, even reasonable views, approaches and attitudes towards lifecycle requirements and restrictions may appear to be very different for the customer and the developer sides (including their various representatives at a number of levels), which can lead to negotiation problems and to a significant increase of the project time and cost. Coordination of these problems between the developer and the customer becomes a matter of great importance: it allows to arrive to a common understanding of the key design constraints, and in many cases helps to avoid the crisis, which is often human factor-related. Therewith, the customer usually wants to impose the bottom limits on the key product features, for example, requesting that the number of concurrent users must exceed a certain value. Thus, the software developer should be able to arrive to a written agreement with the client on these mission-critical parameters. This agreement may be a legal document contract, such as technical requirements, requirements checklist or some other document. The software product developer, in contrast to the client, often seeks to impose the constraints on the top limits of the product operating parameters, such as number of concurrent queries and throughput, or to verify that the product is still going to behave adequately under the technological and financial constraints, so that it will operate within the required performance range.