CMPT 816 Exploratory Topic Assignment 3

Svetlana Slavova
sds797@mail.usask.ca

Quality metrics of a software product

1. INTRODUCTION

Quality is an important aspect of every software product that has been defined in various ways by researchers and organizations. According to the International Organization for Standardization (ISO), quality is a “degree to which a set of inherent characteristic fulfils requirements” [1]. ISO 9000 is a quality standard, which specifies the quality metrics that must be taken into consideration when evaluating a software product. The main advantage of the international standards is that various products can be compared according to the same quality metrics and to distinguish a high quality product from a low quality product. However, the standard is general, in order to fit in a large number of organizations. This leads to the lack of specific instructions showing what kind of actions to be undertaken in different situations.

Another definition of quality is given by Crosby: quality is “conformance to specifications” [1].

These definitions suggest that the requirements are gathered thoroughly, correctly, and represent completely the needs and the preferences of the customers. They correspond to the deontological paradigm, which suggests that the software product should have zero defects.

In contrast, the teleological paradigm describes the quality as optimization with respect to customer satisfaction, profit, or quality constraints (such as time and cost). Weinberg’s definition of quality follows the teleological model. He represents the quality from customers’ point of view: quality is “value to some person” [1].

These quality paradigms suggest that different quality metrics should be considered in order to evaluate a software product. These metrics depend on the main participants who are involved in the product life-cycle. Three main groups of participants can be distinguished: customers, who use the product; managers of the company which develops the product; and a team of developers that implements the product.

The first group initiates the development process of the software system. I.e., if there are not customers for the product, then there is no sense of making investments and creating such a product. Developing a product and then looking for clients is like having a hammer and looking for a nail. Therefore, the clients are very important part of the software system, which has to be designed in a way that represents their needs and preferences. These needs and preferences correspond to the quality metrics (such as performance, reliability, security, mean-time-between-failure (MTBF), etc.) that the product should provide.

The second group participants, the managers, represent the company, which develops the product. This group must be able to communicate with the clients, using a common, domain-specific language, in order to gather the correct requirements of the system from clients' perspective. Along with the explicit requirements, the managers must identify and analyze the derived requirements of the product that occur implicitly from the customers' requirements. This suggests that there are other quality metrics from management perspective, which should be taken into account by the second group participants, such as cost/opportunity cost, budget, profit, deadlines, customer satisfaction, etc. Usually, the metrics from managers' point of view are related to the entire development process.

The team of developers represents the third group, which takes into account lower level characteristics of the software product. The developers must be able to understand the explicit and the implicit requirements, and to implement the behavior of the system, according to its specification. They observe the quality metrics, which are related to the implementation of the product, such as code coupling, code optimization, naming conventions, lines of code, programming language-related features, functional defects, etc.

On the other hand, the quality metrics vary according to the "iron triangle" [2]: quality, time, and cost. These metrics can be seen as aggregated measures that are used to determine the overall quality of a product with respect to the different groups of participants.

The main questions of this work are as follows:

- What kind of metrics is used with respect to the people who are involved in a software product?
- What kind of procedures and techniques are used in order to evaluate these quality metrics?

2. PROBLEM DEFINITION

This exploratory topic focuses on metrics that are used in Software Engineering to evaluate the quality of the software products. The metrics are discussed regarding the three groups of participants who are involved in the software product life-cycle. In addition, various procedures and techniques, such as causal loop diagrams determining the dynamics of the systems, six sigma, peer reviews, and testing, for quality metrics evaluation are presented depending on the level of abstraction – customers, managers, and developers.

3. RELATED WORK

Different quality metrics characterize a software product. There is not a unified classification that describes all possible metrics, nor a set of standardized techniques, which should be used in order to evaluate the quality of the product.

The following sections of the paper refer to some quality metrics, which are discussed by researchers in the field and industrial developers. The metrics are distinguished according to the three main groups of participants in the product life-cycle – the customers, the managers, and the developers. In addition, methodologies and techniques for quality evaluation are presented.

3.1. Quality metrics

Quality metrics from customers' perspective

The main metrics from clients' point of view vary, due to the diversity of possible customers. The most important quality metrics are as follows:

- **Understandability** – The different functions of the software product are clearly outlines. The user manual is unambiguous and complete. The user is able to work with the system using the given documentation [3]. In addition, the application provides consistent terminology using a domain-specific language, which is understandable to the client [48]. At the same time the consumer is able to navigate in the system in an intuitive manner, knowing the effect of their actions at any moment of time by receiving appropriate feedback. As a result, the user is able to operate with the system efficiently and to achieve their "goals faster, with fewer steps, and fewer errors" [48]. Although the system might be complicated, from customers' perspective it must look neat and simple, without any implicit assumptions about the users' behavior. The whole complexity must be hidden to the clients;

- **Completeness** – A software system is complete when its all functions are fully developed and integrated with the other components of the product [3]. Completeness guarantees that the product provides functionality as specified in the requirements;

- **Efficiency** – The system is able to operate without consuming more resources than it is needed, as specified in the requirements [3];

- **Portability** – The product is able to run on various platforms, without a necessity of modification [3]. This feature allows the user not to be limited by specific configurations and environments. Portability can be achieved by using system independent programming languages, such as Java, XML, HTML, Perl, PHP, and SQL;

- **Maintainability –** The software system can be updated easily [3] at a low level of cost. For example, the user must be able to remove old versions of the system and to install patches and new versions without difficulties;

- **Usability –** The users must be able to operate with the system in an easy and intuitive way, using appropriate User Interface [3]. At the same time, sensible error messages, clearly marked operation steps and exits must be provided. Usability is tightly related to understandability;

- **Dependability –** It assures that the system is able to work properly according to the specifications and at the same time the users' protection is guaranteed. Dependability is an aggregated concept, which includes availability, reliability, safety, and security [49];

- **Availability –** The system is accessible and the user is able to work on it [5]. Availability guarantees that the application is up and running, but does not guarantee the correctness of the result. This feature can be achieved by providing backup devices, which are used as alternative in case of a problem in the main device;

- **Reliability –** The system is available and it operates properly [3]. The results of the performed operations are accurate. One approach to guarantee the correctness of the obtained outcome is to assure parallel computation of the same request by different providers, for example replicated servers. The result provided by the majority of the servers is considered as the correct one;

- **Safety –** "Absence of catastrophic consequences on the user and the environment" [49]. Safety instructions and guidelines regarding the proper usage of the product and the required hardware must be provided to the customers in order to reduce the risk of injuries;

- **Security –** The system is protected against unauthorized access [3]. At the same system integrity is guaranteed, i.e. there are not improper states. Security plays a significant role in the area of mobile and ubiquitous computing, where the communication between various devices is realized by network, usually the Internet. User authentication can be achieved by various ways separately or combined together, such as the following:
    - Username & password (widely used approach);
    - Verification questions & answers (used in online banking – Royal Bank of Canada);
    - Unique identification number, automatically generated by a device provided by the organization whose system is accessed (used in auditing firms – KPMG);
    - IP address (used within organizations to form an Intranet network of devices which have the right to access the target system).

Quality metrics from managers' perspective

The quality metrics from management point of view refer to the entire process of development of the software product [6], including quality management and risk management.

Product development is a continuous process, which includes direct factors such as requirements discovery, planning, design, development and documentation, evaluation, and maintenance. Requirements analysis is a key part of the process, because it refers to identifying what must be done in order to meet the needs and the preferences of the customers [24]. Usually, gathering the right requirements is a discovery process, which goes through multiple iterations.

The entire development process, as well as the quality of the final product, depends on the applied model. Kan [6] presents the following software development models:

- **The waterfall development model –** The development process is broken down into sub-tasks – gathering the right requirements, design of the system, implementation, test, and release to the market/the clients. After the end of every step, the completed work is tested to validate whether the requirements are followed correctly. This is an example of divide-and-conquer approach, which splits the complex problem into logically independent steps that can be verified before the completion of the whole process. However, this model assumes that the gathered requirements will not change by the end of the process development. Since collecting the right requirements is a continuous and discovery process, especially for unknown domain or unknown clients from company's point of view, they might not be completely investigated after the first stage of the process. This suggests that the next steps of the development process will not address the last modifications/clarifications in the requirements;

- **The prototyping approach –** This approach can be applied when the complete requirements are not known. It is related to the development of a prototype system that can be used by the customers, in order to feel the system and to provide feedback. The collected feedback is analyzed and updates in the requirements and the design of the system are made. Then, the system is presented to the customers again who are able to provide more feedback. Once the clients are satisfied with the prototype version of the product, full development begins. The benefit of this approach is that it helps gathering the requirements of the system. However, it is more applicable technique for small problems, since the partial implementation of a complex system might be time-consuming;

- **The spiral model –** This technique is a combination between the waterfall development model and the prototyping approach, taking into account risk management. The development process is broken down into several phases. Each phase includes risk analysis, requirements update, design, prototype/implementation, and testing, and each phase is based on the results and the experience, gained from the previous phases. This flexible approach is appropriate for complex projects, since it allows extending the software system incrementally, improving the quality of the product after every phase;

- **The iterative development process model** – This approach is a combination between the waterfall development model and the prototyping approach. Risk management can be taken into consideration as well. The requirements and the design are reiteration steps; the iteration includes risk analysis, prototype, testing, and integration with previous iterations;

- **The object-oriented development process** – It contains three phases – analysis, design, and implementation. During the first phase, the requirements are gathered and represented regardless the software and the hardware characteristics. During the design phase, potential software and hardware is considered, as well as the main classes are designed. The class hierarchy is implemented in the last phase of the model.

The development model must fit in the company and the complexity of the software system. Quality and risk management are long-run related aspects, which are taken into consideration by some of the presented development models. Quality management ensures that the development process is "effective and efficient" [7] and is up-to-date all the time, in order to address the changes in the dynamic environment. Risk management evaluates the risk points of the development process and undertakes different strategies, in order to deal with uncertain situations [8]:

- **Risk avoidance** – It avoids all risk situations. Although, this approach is safe, it suggests that potential successful projects, which require some risk, might not be undertaken at all;
- **Risk reduction –** It deals with small risks at a time;
- **Risk retention –** It accepts the results of the occurred uncertain situation;
- **Risk transfer –** It relocates the risk to another party, such as insurance company.

Quality management and risk management are related to the project schedule, the cost of the development process, as well as to the customer satisfaction. The cost refers to investments and support costs in terms of resources devoted to the software product, such as people, working on the system, software and hardware technology that is applied in the product, as well as opportunity cost, related to the value of the next best alternative project that is not realized.

Aggressive deadlines increase the possibility of uncertain situations during the project development. In order to release a high quality product, the company must take into consideration risk management. This augments the investment costs of the product. On the other hand, extended deadlines increase the opportunity costs for the company and at the same time influences on the customer satisfaction. Therefore, balance between the metrics of the iron triangle – quality, time, and cost – must be found.

Quality management and risk management are tightly related to the organization type of the company. In some software subcultures these factors are not taken into consideration at all. The different patterns, presented by Weinberg in [11], are as follows:

- **Pattern 0: Oblivious –** The employees do not even realize that they are involved in a development process;

- **Pattern 1: Variable –** The company/team succeeds/fails because of a "super-programmer";

- **Pattern 2: Routine –** There is a "super leader" in the company who follows the current development process, without understanding it. A company that follows this pattern can hardly react to any changes in the process, since no one realizes the occurrence of the current steps in the process;

- **Pattern 3: Steering –** The managers deeply understand the entire process and are able to react accordingly to the dynamics of the environment. Quality and risk management are part of the company's business culture;

- **Pattern 4: Anticipating –** The managers in this pattern not only react appropriately to the changes of the environment, but also use the experience, the knowledge, the evaluation and analysis, gained from other projects. This helps them to act in advance and to achieve a stable level of quality and risk management;

- **Pattern 5: Congruent –** The managers have many years of experience, which helps them deal with various uncertain situations, taking into account a high level of quality and risk management.

Patterns 3, 4, and 5 refer to large organizations, which are able to develop complex products, due to the understanding of the potential risk factors. They are able to react appropriately, when it is needed, due to the quality and risk management. However, the number of these organizations is limited (patterns 4 and 5 do not even exist at this time), compared to companies that follow patterns 0, 1, and 2.

Quality metrics from developers' perspective

From developers' point of view the design of the software, its source code, and the different components must be measured, in order to evaluate the product's quality. Some important metrics are listed below:

- **Conciseness** – The code is optimized in terms of lines of code [3]. Conciseness refers to removing redundant code, providing reachable statements, decreasing computation time of expressions, presenting clear and unambiguous branches as well as state transitions. It can be achieved by following specific programming rules, standards, and conventions. In addition, peer reviews can help improve the code optimization;

- **Readability** – Well-written source code is easy to be understood by other developers [3]. It can be achieved by structuring the program in a modular way and keeping the units of the application relatively small. The chosen architecture of the system plays a significant role as well. In case of a complex system, every function can be seen as a different component which follows the same terminology, conventions, and commenting style;

- **Consistency –** The product consists of components, which correspond to the same domain-specific terminology, language-specific conventions, comment style, and notations [3]. This facilitates the readability of the programming code, as well as decreases the possibility of errors, due to misunderstood terminology;

- **Testability/ease of debugging** – The source code can be tested in an easy manner, in order to evaluate its characteristics, such as error rate, reliability, security, performance, etc. [3]. It is tightly related to the used programming language or a set of programming languages. For example, low-level languages (such as Assembler) are faster and more powerful, but the probability of error occurrence is bigger, compared to high-level languages (such as C++, Java);

- **Structure** – The implementation of the components of the system follows a particular design pattern [3]. This metric might take into account the layering architecture of the software system. Each layer separates concepts on different levels. The code within a layer is tightly-coupled, whereas the code between layers is loosely-coupled. This allows clear design, better maintenance, flexibility, and parallel development of the system;

- **Maintainability** – The source code can be updated easily [3]. This metric is related to the structure of the software system. A layered system facilitates maintainability, since one layer can be replaced by another one, which provides the same interface, without changing the rest of the layers. In addition, maintainability depends on the used programming languages and conventions, and code complexity;

- **Flexibility** – A flexible system is able to adapt to the changes of the dynamic environment at run-time [50]. For example, a fault-tolerant application can be seen as a flexible application, since it is able to perform its service in the presence of failure in one or more of its components by using replicas. The flexibility can be achieved by appropriate programming languages and techniques. For instance, Service-Oriented Architecture (SOA) allows developing autonomous, system independent, interoperable Web Services, which can be deployed at run-time, depending on the needs;

- **Scalability** – A scalable system allows handling "growing amounts of work in a graceful manner" [51]. It can be achieved by adding replicated components in the bottle-neck areas. For example, when dealing with a web-based system, the amount of parallel users influences the performance of the server as well as the database system. In order to deal with this problem more servers can be added to the application to process the coming requests. However, this requires an appropriate routing techniques to be applied, such as load-balancing using round-robin or random selection of the target server, or even a complex reasoning mechanism if additional information about the request is available – for example, if the request is with high priority, or if it is a read request, which does not change the state of the system, or some quality of service (QoS) information about the servers is available;

- **Fault-tolerance** – A fault-tolerant system is able to continue working even in the presence of failures in one or more of its components [4]. The system is stable and has error-free states. There are two types of recovery in fault-tolerant systems – roll-forward (in case of a failure, the system corrects its state, in order to be able to work forward) and roll-back (In case of a failure, the system reverts to an earlier correct state) [10]. Duplication is a popular technique for achieving fault-tolerance. There are three types of duplication:
    - ***Replication*** – The system consists of multiple identical components, which are requested in parallel and the final result is chosen using a quorum;
    - ***Redundancy*** – The system consists of multiple identical components and only one component is used at a time. If it fails, the system starts using another component (the backup);
    - ***Diversity*** – The system consists of components that provide the same functionality but have different implementation and uses them in parallel, in order to get the correct result.

  However, a fault-tolerant system increases the complexity of the product, since all components must have the same internal state when they are used interchangeably. It requires both state synchronization and data consistency to be taken into consideration. On the other hand, the augmented complexity of the system reduces its ease of maintainability and reflects on its structure. This suggests that there is a tension between some of the metrics. Therefore, it is crucial to find a balance between them.

Quality metric classification according Kan

Kan [9] classifies the quality metrics into the following categories:

- *Product metrics*, related to the software product, such as size, complexity, performance, customer satisfaction, etc. This category includes criteria which are related to two groups of participants – the customers and the developers. The product metrics, presented by the author, are summarized in table 1;

- *Process metrics*, related to the continuous development process of the software system, such as defect density, which can be controlled by various testing techniques. This category, includes quality metrics that should be taken into account by the team of developers;

- *Project metrics*, related to the management of the project, such as number of developers, cost, schedule, etc. These metrics are considered by the managers of the company that develops the software product.

Product Metrics	Description
Mean Time To Failure (MTTF, MTBF)	MTTF shows the average time between failures in a software product. It is based on two measures – **time** between failures and **defects** in the system. This metric is difficult to be collected and a high level of correctness.
Defect Density	“Defect is an anomaly in a product”, which causes a failure. - From developers’ point of view, this metric can be represented by the number of **lines of code** of the software. However, this representation is ambiguous, since it does not specify how exactly to count the lines – whether to count definitions, comments, or only the logical/physical statements. This ambiguity results in the software size and respectively reflects on the defect density. Another way to determine the defect density is to take into account the number of functions of the system. - From customers’ point of view, the defect density is represented as **quality per product**. This suggests that new versions of the product must have lower number of defects, regardless the size and the complexity of the new version, compared to the previous one.
Customer problems	Another product quality metric is the **number of problems per month, experienced by customers** when using the product. It shows much more than the defects in the system – it is related to user errors, usability issues, ambiguity in the product documentation, etc. In order to reduce this quality factor, functional and non-functional defects have to be analyzed and removed. This metric represents the customers’ point of view and it is tightly related to the customer satisfaction.
Customer satisfaction	This metric represents the overall level of satisfaction of the customers of the product. It is evaluated via surveys, which ask the clients how they feel about the product, regarding various categories – functionality, performance, reliability, usability, etc.

Table 1. Product metrics

3.2. Quality metrics evaluation techniques

Software testing is a process that identifies faults and is directly related to the quality of the observed system. Testing is an evaluation methodology which confirms whether the application meets requirements in the specification, but does not guarantee that the system is defect-free. There are several levels of testing, depending on the degree of completion of the software product:

- **Unit testing** – The unit testing verifies that a particular component (module) of the software system meets the requirements [36]. This test method examines the system on a module level and validates whether the individual parts work as specified.

  Advantages of unit testing:
  - It simplifies the process of integration of the system components. Once the modules of the application work properly, then it is much easier to be integrated in a fully-functional system;
  - When there are changes in the module, it can be tested again, in order to verify that there are no regression defects.

  Disadvantages of unit testing:
  - It does not allow catching integration defects;
  - It does not allow observing the behavior of the whole system, such as performance, reliability, availability.

- **Integration testing** – The integration testing verifies whether the components (modules) of the system work together properly [37]. This testing method follows after unit testing and examines the inter-components communication.

- **System testing** – The system testing follows after the integration testing. Its goal is to verify whether the system works as specified in the requirements [38]. The whole system is analyzed as a black box and its behavior is observed.

- **Acceptance testing** – The acceptance testing is realized from clients' point of view and focuses on the quality of the system [40]. It assures that the observed application meets the needs and the preferences of the users, as stated in the product specification.

Most of the evaluation techniques have three main components: the observed system, a set of system inputs, and a set of system outputs. They can be applied at different levels of the system development. The results of the evaluation depend on the obtained outputs and give a direct overview of the quality metrics of the product.

The following sections discuss evaluation techniques from different perspectives – the customers, the management, and the developers.

<u>Evaluation techniques from customers' perspective</u>

- **Black box testing –** Black box testing is a testing technique, which verifies the behavior of the observed system, without knowing its implementation [25]. The tester has a set of inputs and knows their correct outputs. Depending on the collected results, he/she is able to observe whether the application corresponds to the system requirements. This type of testing is considered as "testing with respect to the specifications".

  Black box testing can be realized in a structured way using decision tables, state transition diagrams, orthogonal arrays, and Pareto analysis, or in a less structured way using random or exploratory testing [44]:
    - A decision table shows a combination of conditions and corresponding actions (rules). Test cases can be derived from the decision tables;
    - State transition diagrams represent the system's transition from one state to another. The system is tested in order to observe whether the state transitions are correct. As minimum all transitions must be evaluated;
    - Orthogonal arrays represent pairs of interactions and covers pair combinations, rather than all possible combinations. The main reason for using orthogonal arrays is that the compatibility problems are pairwise;
    - Pareto analysis identifies the important test cases, instead of applying all possible case scenarios;
    - Random testing tests the system in a random manner, without following any specific test cases;
    - Exploratory testing allows the tester to learn about the behavior of the system during the testing period. The next test case is based on the experience of the previous one.

  Advantages of black box testing:
    - There is no need of knowing the exact architecture and the corresponding implementation, in order to perform black box testing [25];
    - The system developer and the system tester could be independent from one another [25];
    - The tests verify the behavior of the system from users' perspective, which provides a high-level overview of the system [25];
    - It helps detect any ambiguities and/or discrepancies between the specifications and the built system [25];
    - Once the system specification is completed, the test cases can be created [25];
    - It can be applied to different development levels, such as unit, integration, or the whole system [26].

  Disadvantages of black box testing:
    - According to [25], only a small fraction of all possible inputs can be tested, which leads to many not tested scenarios. However, by testing the

boundary cases as well as some situations in between, the tester obtains a good overview of the system's behavior;
    - According to [25], this approach requires neat and clean specification. Actually, without such a good specification and unambiguous requirements, no one is able to build a high quality system;
    - Since the developer and the tester could be completely independent, a repetition of already completed tests might occur [25];
    - According to [25], it does not allow to test a particular segment of code. However, since this approach focuses on the users' perspective, there is no real need to test a specific block of code, since this can be done using another type of testing technique;
    - Since the tester does not have visibility to the business logic and the developed code, it is not possible to detect the cause of the failures.

- **Alpha testing** – The alpha testing is performed by a limited number of users within the company which develops the software in order to observe the behavior of the product and to detect failures in an environment, which is close to the real one [42]. Although the product does not provide all features, it is tested to collect feedback from the users and to find undesired behavior [43]. Usually, the alpha testing is followed by beta testing.

- **Beta testing –** The beta testing is a user testing which involves a large amount of clients during the testing period [42]. The product is placed in a real world exposure in order to observe its characteristics and quality metrics from users' perspective.

Evaluation techniques from managers' perspective

- **System dynamics**
  This is an approach which helps evaluate the dynamic nature of complex systems [14], [15]. The system dynamics model is represented by causal loop diagrams and is observed via simulations. The causal loop diagrams display every factor of the process as a node and describe the dynamics of the processes using feedbacks. Two types of feedback can be applied – positive and negative. The former shows that the nodes improve in the same direction and represents a reinforcing loop, whereas the latter changes the nodes in opposite directions and corresponds to a balancing loop. The reinforcing loops can be extremely dangerous, since they are "associated with an exponential increase/decrease" [16]. Fast positive loops are able to cause harm in the process in a very short period of time. They are called vicious cycles. Long-run vicious cycles are very hard to be controlled and require outside efforts/investments to be added to the system, in order to deal with the situation successfully. In contrast, the negative loops help reaching a balance level in the system. Figure 1 shows a causal loop diagram, which contains both positive and negative loop. The factors of this CLD are the following: Software development, Number of faults, Product quality, Testing, and Necessity of modifications. The software development adds faults to the software product. When the number of faults increases the product quality decreases and more testing is required. The testing eliminates faults and as a result, the number of faults is reduced. This loop is an example of a balancing loop. In contrast, the positive loop takes into account the necessity of modifications. When the software development increases, it augments the number of faults in the system. As a result, the product quality decreases. This requires modifications in the software to be done. The increased need of modification suggests more software development.

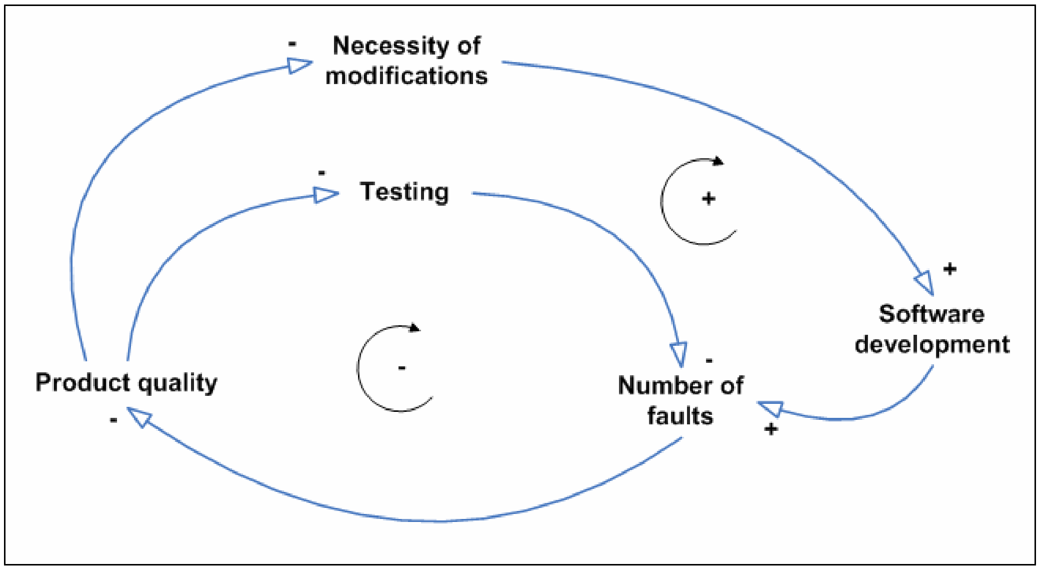

Figure 1. Causal Loop Diagram

In order to observe the dynamic behavior of the system model, the method of simulation is used. The simulation allows observing key characteristics of a system or a process in a simplified manner [17]. At the same time it gives a point of control, so that the whole process can be evaluated using different situations and parameters. For example, AnyLogic [18] simulation environment can be applied for assessing the dynamic nature of complex processes.

- **Six sigma**
  Six Sigma is a data-driven methodology which eliminates unacceptable faults in software products [13], [14]. Six Sigma consists of two sub-methodologies – **DMAIC** and **DMADV**. The former stands for ***Define***, ***Measure***, ***Analyze***, ***Improve***, ***Control*** and “is used to improve an existing business process” [14], whereas the latter corresponds to ***Define***, ***Measure***, ***Analyze***, ***Design***, and ***Verify*** and is used to develop new business process designs that lead to products which are close to the specifications. The first three phases of DMAIC and DMADV are similar:
    - ***Define*** – Specifies the goals of the development process so that they fit in the company’s style and correspond to the customers’ requirements. When it refers to DMAIC, the focus is more on improving goals, whereas for DMADV this phase defines the goals without relying on existing goals;

    - ***Measure*** – Specifies basic measures which are critical for the development process. These metrics can include both product measures (such as metrics specific to the product) and process measures (such as risk management). When it refers to DMAIC, the focus is on defining the measures that are crucial for the existing process and that give precise information regarding the performance of the existing process. These metrics are used for future comparison. When it refers to DMADV, the metrics represent the new development process;

    - ***Analyze*** – This phase identifies the reasons for the defects in the existing process (for DMAIC) or defines alternative process options which can be used and evaluates the design that fits best in the customers’ specifications and the company’s strategy (for DMADV).

  The last two phases of DMAIC try to ***Improve*** the existing process, taking into account the information gathered during the previous phases and to ***Control*** the future performance of the process.
  In contrast, the goals of the final phases of DMADV are to ***Design*** in detail the development process to ***Verify*** the design whether it meets the set goals.

- **Software inspections**
  Software inspections help improve the quality of the software systems [22], by performing peer reviews of the target systems to detect defects [23]. This technique can be applied at a global level of the development cycle, as well as at a low level, focusing on smaller details. Software inspections are one type of peer reviews, but are discussed here, since they can be seen as evaluation technique from managers' perspective. The other less formal peer review techniques are presented in the next section.

  The inspection team consists of individuals who analyze the development activities in order to find defects in the observed system [45]. Every member has a specific role in the inspection. The roles of the participants are one or more of the following [44], [45]:
    - ***Author*** – An individual who develops the program and is responsible to remove the found defects. In addition this person shares issues and concerns about potential defects in the system;

    - ***Moderator*** – The moderator leads the software inspection process and verifies whether the other participants perform their assigned roles;

    - ***Reader*** – This individual leads the inspection meeting and presents the inspected content;

    - ***Inspector*** – The inspector is responsible to study and find defects in the system;

    - ***Recorder*** – The recorder documents the found defects as well as some additional information about them regarding their type, location, importance, etc.

  The inspection is a formal process, presented in figure 2, which includes the following stages [45]:
    - ***Planning*** – Planning is an important part of the software inspection process. The observed system is "frozen" during the inspection. The preparation should be at least 2 days, whereas the meeting must be up to 2 hours long. Important checkpoints must be specified and a checklist to be created, which covers no more than 20 pages of the system.

    - ***Overview*** – The goal of the stage is to give a general overview of the product regarding the main functions and description of the most important techniques which are applied. As a result the inspection team has a better understanding and higher knowledge about the observed system.

    - ***Preparation*** – During the preparation stage every inspection team member inspects the system to find defects in the requirements, the development process, the business logic, data structures, name conventions, and the source code. As a result a list with potential defects is completed.

- ***Inspection meeting*** – The potential defects, found in the preparation stage, are discussed at the meeting. The product can be accepted, confidentially accepted, or re-inspected. The system is accepted when there are not deviations from the requirements; confidentially accepted when there are major defects which must be removed; and re-inspected when there is a significant amount of defects which require change in the primary system.

- ***Rework*** – The goal of this stage is to remove the detected defects in the system.

- ***Follow-up*** – If the product is conditionally accepted during the inspection meeting, revisions regarding regression defects are done. However, if the result of the meeting is re-inspected product, the software inspection process starts from the beginning.

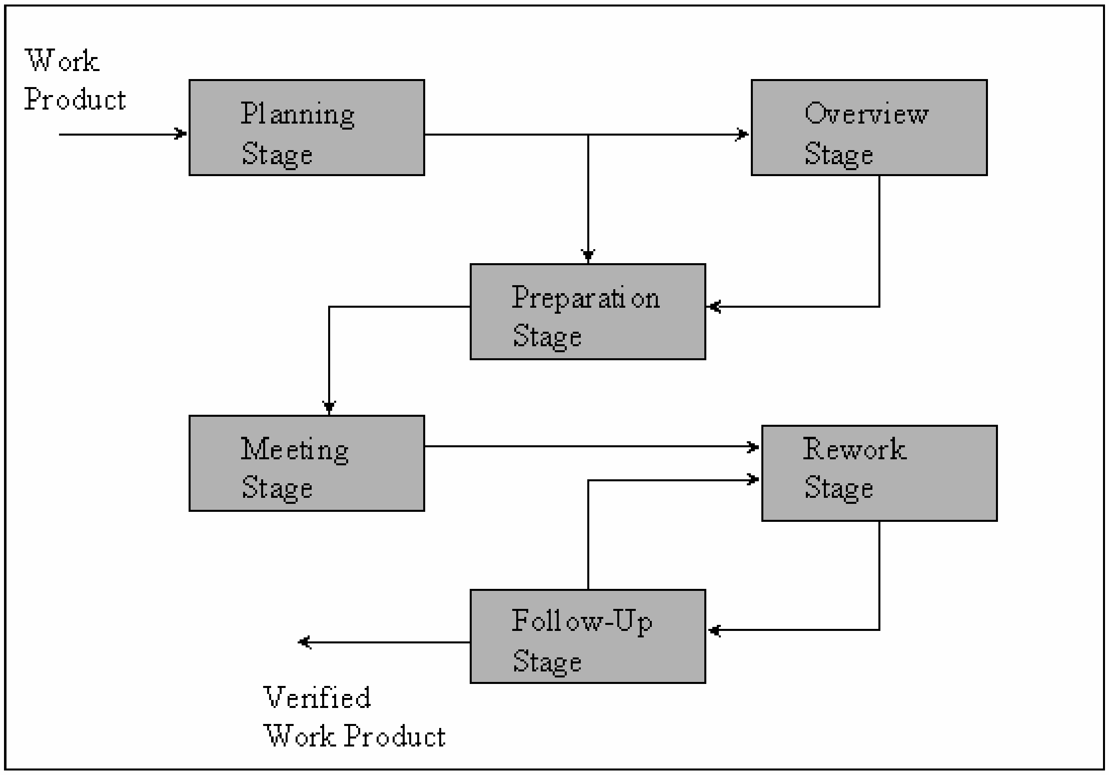

Figure 2. Software Inspection Process presented by [45]

There are four types of reports used during the software inspection process [45]:

- ***Inspection meeting notice*** **–** It informs the inspection team for the inspection process.

- ***Inspection defect list*** **–** The defect list is completed during the preparation, the meeting, and the rework stage. During the preparation stage, information about defect location and description is filled in. The defect type, class, importance as specified at the meeting. Finally, the time needed to remove the corresponding defects is filled in at the rework stage.

- ***Inspection defect summary*** – It is completed after the inspection meeting and represents a summary of the detected faults in the system divided into three groups according to the importance of the defect – major, moderate, and minor.

- ***Inspection management report*** **–** It is completed after the inspection meeting and contains information related to the overall impression of the product, the found defects, as well as the time needed for the entire inspection process broken down into hours per stage.

Evaluation techniques from developers' perspective

The evaluation techniques from developers' perspective are related to software testing, which assesses the quality metrics. These techniques are widely used by system testers as well. Sometimes the testers are part of the programming team. In other cases external testers might be hired to evaluate the system. Since the testers must have background knowledge which is close to the developers', both the programmers group and the testers group are considered as developers in this paper.

This section describes some evaluation methods which can be used to verify the correctness of the target application. Although in the real-world a particular evaluation technique is a combination of several testing methodologies, the methods here are discussed separately, in order to distinguish their main characteristics.

- **Manual testing** – The manual testing requires all steps of testing to be completed manually [33]. It is used mainly in the beginning of the software development, when automated testing cannot be applied or when the test is run only few times and it is too expensive to create an automated test [34].

  Advantages of manual testing [34]:
  - It is much cheaper to run a test manually than automatically;
  - It allows using creativity and intuition during testing.

  Disadvantages of manual testing [34]:
  - After every change in the software, the same test cases must be run manually again, which is a very time-consuming process.

- **Automated testing** – Automated tests allow verifying the quality metrics of a system using software which controls the tests [20].

  Advantages of automated testing [34]:
  - It saves time when the same tests must be run frequently, especially when the code changes. For example, one run of the automated test might find N defects in the software product. After fixing these faults, the same test must be run again, in order to observe whether the previous bug fixing process has introduced new error states in the system;
  - It allows running more test cases than the manual testing.

  Disadvantages of automated testing [34]:
  - It requires investments;
  - Not every test case can be automated, such as test cases that are related to GUI.

- **Static code analysis** – The static code analysis refers to system testing which evaluates the system without executing it [19]. Instead, this method uses automated tools that analyze the behavior of the source code in terms of "semantic errors". This approach is appropriate for separate components of the system, due to the fact that the static code analysis does not allow detecting errors in multiple levels of the system architecture. Some examples of tools which can be used for static code analysis are presented in the table below:

N	Language of applicability	Name	URL
1	HTML	W3C Markup Validation Service	http://validator.w3.org/
2	JavaScript	JS Sorcerer	http://www.dhitechnologies.com/products/jssorcerer/
3	PHP	Armorize CodeSecure	http://www.armorize.com/
4	Java	Jtest	http://www.parasoft.com/jsp/products/home.jsp?product=Jtest
		FindBugs	http://findbugs.sourceforge.net/
		JDepend	http://www.clarkware.com/software/JDepend.html
		SofCheck Inspector for Java	http://www.sofcheck.com/
5	C/C++	Axivion Bauhaus Suite	http://www.axivion.com/
		Cantata	http://www.ipl.com/products/p0000.uk.php
		CodeWizard	http://www.parasoft.com/jsp/products/home.jsp?product=Wizard&
		OpenC++	http://opencxx.sourceforge.net/
		C++test	http://www.parasoft.com/jsp/products/home.jsp?product=Wizard&
6	C#	.TEST	http://www.parasoft.com/jsp/products/home.jsp?product=TestNet&redname=testnet&referred=testnet
		DevMetrics and DevAdvantage	http://www.anticipatingminds.com/Content/Home/Home.aspx

Table 2. Tools for static code analysis

- **Dynamic code analysis** – The dynamic code analysis verifies the performance the system during its execution [19]. The tester is able to observe the behavior of the desired application with the operating system, as well as to identify its compatibility with other programs (if applicable). This approach is used when the whole product must be tested with real-world situations.

  A daily build and **smoke test** is an example of dynamic code analysis. A daily build is created every day from scratch and represents the current level of the system's components as a single entity, which is compiled and linked together

[21]. This approach allows the integration of the program components to be an incremental process. A daily build is tested with a smoke test. The goal of the test is to determine the problems that have occurred in the last daily build. This dynamic code analysis is appropriate for large projects, which contain hundreds to thousands components that must work properly together.

Advantages of a daily build:

- It detects whether there are critical incompatibilities between components as well as problems in the global design of the software system;
- It verifies the behavior of the product in incremental steps and makes it much easier for defect detection. As a result, the test period of the product can be reduced;
- It helps build team confidence during the project development.

Disadvantages of a daily build:

- It requires considerable amount of time and resources, in order to be done every day;
- It requires a large amount of stub code;
- The smoke test must be kept up-to-date.

- **Black box testing –** Black box testing can be applied not only by users (as discussed in a previous section), but also by developers. Since the main characteristics of the method are already presented, they are not covered in this section.

- **White box testing –** The white box testing, known also as structural testing, clear box testing, and glass box testing, verifies the correctness of the code of the system by considering its architecture and implementation [27], [28], [29]. The tester has a "full visibility of the internal workings of the software product, specifically, the logic and the structure of the code" [27] and must know the structural design and the business logic of the system. The tests cover different paths of the application, including small and large code fragments, such as statements and branches.

  White box test cases [29]:

  - *Basis path testing* suggests that all independent paths in the system must be tested. An independent path is defined as a path that contains one or more programming statements, which are not specified in another path. The simplest way to calculate the independent paths in the system is the following: **Number of paths = number of conditions + 1**;
  - *Boundary testing* covers the extreme cases in the system, i.e. looks at the possible boundary conditions;
  - *Control flow testing* takes into account the order, in which the programming statements are executed;
  - *Data flow testing* considers the data that is defined and used in the system;

- *Failure testing* requires deep analysis, since the goal of this method is to find those inputs, which can cause a program failure, such as security issues, buffer overflow, incorrect input, division by zero, and others.

Advantages of white box testing:

- The test inputs can be determined easily, due to the fact that the internal structure of the system is known;
- It allows detecting which code fragments cause a failure in the system;
- It allows code optimizations;
- It can be applied to different development levels, such as unit, integration, or the whole system.

Disadvantages of white box testing:

- The tests can be created after the system is developed;
- A knowledgeable tester that understands the internal structure of the system is required;
- In case of large applications, it is not feasible to run all possible test cases, which increases the chance of leaving defects in the product. However, all paths in the system must be evaluated, in order to discover potential defects;
- After every modification in the code and/or the business logic of the system, another set of tests is needed, in order to verify whether there is not any unpredictable behavior in the system.

- **Grey box testing –** The gray box testing is a system testing method that is based on both black and white box testing [30]. The tester has some knowledge about the internal structure of the target system and applies test cases to it. The part of the program, which is not known, is evaluated using black box testing.

  Grey box testing is appropriate for Service-Oriented Architecture (SOA) and Web Services in particular, since neither pure black, nor white testing are applicable in the domain [31]. By nature, the Web Services are distributed and independent components that can bind to each other at run time and can run on various platforms. Usually, they are provided by different parties, and in most cases, the exact requirements and specifications might not be accessible. This does not allow black box testing to be applied. On the other hand, the source code of these services is not available, which does not allow evaluating the system using white box testing. However, the Web Services are described by their service description (WSDL) in XML format. It hides the implementation details but at the same time gives enough information for the service interaction, such as method names, input and output parameters. The service descriptions are available and can be used in order to perform grey box testing.

- **Exploratory testing –** The exploratory testing, also known as ad hoc, is a testing method which allows the tester to learn more about the observed system during testing time. As a result, the tester is able to create better tests for the next runs

[32]. “The testing is dependent on the testers’ skill of inventing test cases and finding defects.”

Advantages of exploratory testing:

- The tester does not need to have any particular preparation, before testing the system;
- It is appropriate to be applied when the system specification is not detailed.

Disadvantages of exploratory testing:

- The tests are not known in advance;
- Since the tests depend on the tester, different testers might obtain different results, i.e. results repeatability is not guaranteed.

- **Regression testing** – The regression testing, known also as verification testing, is a testing technique that searches for regression faults. Regression faults occur after one or more changes in the system are made [35]. The testing requires the test cases to be repeated in order to observe whether regression faults have appeared.

  Advantages of regression testing:

  - It allows identifying the cause of a failure observed after the last modifications of the code;
  - It assures that intended changes do not cause side effects in the system;
  - It finds accidental or unintentional errors in the system.

  Disadvantages of regression testing:

  - It is time-consuming, because the test cases must be repeated after every modification in the system;
  - It requires a high-level of discipline to be followed.

- **Functional testing** – The functional testing verifies whether the functions of the observed system work as expected [40]. Usually, it is done at a later stage of the product development cycle, when the distinct functions are completed. However, since it can be applied on a component level, it would be beneficial this type of testing to start as soon as a module of the system is developed.

- **Peer reviews** – The peer reviews provide quality control of a software system [47]. They can be in different forms as follows:
  - ***Inspection*** – The software inspections is the most formal type of peer review. It is detailed, requires preparation, and is more effective than the informal types. It is mainly related to the entire development process in order to find defects in the specifications. The stages of the inspection process (planning, overview, preparation, inspection meeting, rework, and follow-up) are discussed in the previous section of this paper since they

can be seen as an evaluation technique from management's perspective as well;

- ***Team review*** – The team review is a more informal type of software inspection, but contains the same stages;

- ***Walkthrough*** – This informal type of peer review involves several participants – the developer and co-workers. The developer is the leader of the meeting who describes the application, whereas the other participants make suggestions. The walkthrough does not involve any specific criteria or procedures, nor are any reports created;

- ***Pair programming*** – This type of peer review requires two developers to work together on the same problem to review each other's work all the time. The benefit of the pair programming is that comments and suggestions are made by people who are directly involved in and responsible for the system. However, this methodology requires a high level of communication and synchronization between the peers;

- ***Peer desk-check*** – This informal type of peer review involves two participants – the developer and a colleague who looks at the application;

- ***Pass-around*** – The pass-around peer review represents multiple simultaneous peer desk-checks.

Advantages of peer reviews [44]:

- It helps detect design errors, problems in the requirements and even in the development process;
- More flexible than testing since it can be applied at any stage of the development cycle and to find defects early in the process;
- More cost-effective than testing;
- It helps clarify domain-specific terms;
- It increases the communication within the development team;
- It helps improve the programming habits in terms of used standards, conventions, coding style, etc.

Disadvantages of peer reviews [44]:

- Time consuming, especially for the formal peer reviews since they require preparation;
- Ineffectiveness if the focus is on minor issues and details, rather than on the global view of the system;
- There could be a domination of certain people during the discussion.

The figure below shows that the peer reviews can be applied at any stage of the development cycle.

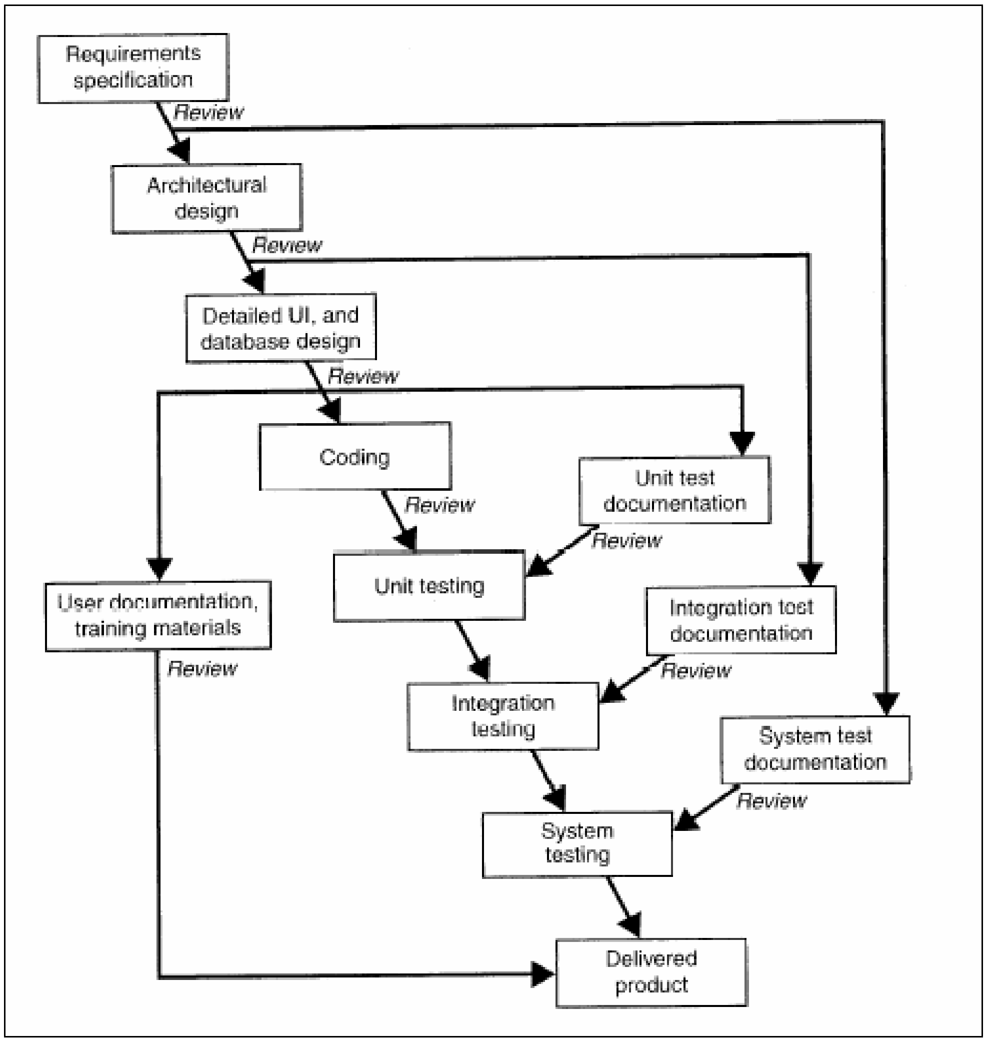

Figure 3. Peer review checkpoints by [46]

4. CONCLUSIONS

The quality of the software product is important. There are various factors that influence on the quality and various metrics that represent the quality of the product.

It is important to analyze the quality metrics from all perspectives – the customers, the management, and the development side. Some of them can be included in more than one point of view, whereas others are derivable from high level metrics and/or requirements and have to be taken into consideration from the corresponding group of participants.

Assessing the quality metrics is a key issue in the product development process. The evaluation techniques from managers' perspective represent a high level assessment of the quality of the business process. This is extremely important and influences the product in long-run. For example, Six Sigma, used by many successful companies, such as General Electrics, is a very powerful methodology for improving business processes.

There are no specific rules or patterns that can be applied to all cases of software systems. The quality metrics, which are taken into account, as well the methods for their assessment vary and depend on the product requirements and specification, the business model of the company, and the development team. However, in order to provide a good quality product, a high level of coordination and communication between the involved parties must be established as presented in the figure below. This helps detect critical errors/ambiguities in the product at an early stage of its development.

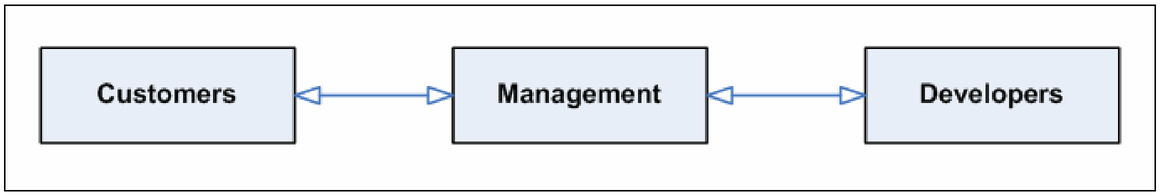

Figure 4. Coordination & communication between the three groups of participants

Testing is a crucial part of the software product development in order to eliminate defects. Defects can occur due to problems in the source code of the system, the development model, as well as ambiguity in the specification of the application. Therefore, it is essential to analyze the software system from all perspectives. As discussed in this paper, various testing methods can be applied. However, the most appropriate set of testing approaches depends on the target application, the specification, as well as the selected business process and the development team. Peer reviews play a significant role in the development process, since they might help detect a significant number of defects at an early stage. In addition, they help improve the process and to prevent defects in future projects. For example, software inspections detect 80% of the defects in a Motorola's project and less formal reviews detect 60%; HP's return is 10 to 1 per software inspection [47].

Although there exist many ways to test a software system in order to detect faults, there is no perfect application. The main reason for this is the fact that it is impossible to test the system with all possible real-world scenarios and to assure 100% product quality. As discussed in [41] even released applications, which have already been tested by experienced developers and testers, can fail due to various reasons, such as “improperly constrained input”, “improperly constrained stored data”, “improperly constrained computation”, and “improperly constrained output”.

This implies a tension between the main metrics of the iron triangle – quality, time, and cost. In addition, there is opportunity cost associated with the product metrics, which must be taken into account as well. For example, the company might have to choose between continuing testing a software system in order to provide a higher quality product, and investing the same resources into another project, which has a high potential for success.

On the other hand, people strive for excellence and further growing all the time. Even when a specific goal is completed, it is not enough and a full satisfaction is not achieved, because other goals appear. This behavior is part of the human nature. That is why when there is a product which is good enough to be used, the clients require more features, higher level of usability, better interface, etc. These improvements are due to the fact that the users’ expectations grow further, which requires changes in the software. This is an indication that gathering the requirements is a continuous discovery process and the requirements evolve with the time. Always there will be a necessity of improvements and therefore there will not be perfect software applications.

As discussed in this paper, there are several organizational patterns – from 0 (not even realizing that are involved in a development process) to 5 (development process lead by experienced managers who are able to react to the changes of the dynamic environment). With time more companies will represent organizations of patterns 3, 4, and 5. This means that the undertaken projects will be lead by experienced management teams that know how to adapt and to react to the uncertain environment applying a high level of risk management. This will help increase the difficulty and the range of the projects in order to provide software, which serves the users’ needs and preferences even better. However, this suggests that the size and the complexity of the applications will grow. New problems will appear and improved organizational patterns will be needed.

Although the nature and the domain of the software systems vary, the key quality metrics are the same. This allows comparing applications according to the quality of service they provide using appropriate evaluation techniques. As presented in this paper, different testing techniques exist, which are combined together to assess the software and to detect defects in order to improve quality. All companies which start a software project go through similar iteration steps during the development. Many of them observe the same tensions between the metrics of the iron triangle – quality, time, and cost. Those who take into account risk management are more likely to react better to the changes of the dynamic environment as well as to unexpected events. In addition, the current standards, such as ISO 9000, specify *what* kind of characteristics a product must provide, but does not describe *how* these quality metrics can be achieved. The following important question arises:

Is it feasible to build a standardized framework or an expert system, which controls and navigates the involved participants during the entire development process?

The framework could be seen as a combination of both quality standards and guidelines based on past experience that helps prevent common errors to occur again and again. It would be an upgrade of the six sigma methodology. The main difference would be that the experience and the knowledge of various companies are taken into consideration. There are common patterns – such as programming languages and environments, development model, conventions, communication between users, management, and developers – that could be observed in failed and successful projects and can be taken into account when starting a new one. There is a necessity of integrating the know-how of the whole IT industry. This would help improve the development process of the software systems, the overall quality of the products, and prevent from wrong decisions that might cause a project failure in long-run.

5. REFERENCES

[1] Wikipedia. *Quality*. http://en.wikipedia.org/wiki/Quality/
[2] Ambler, S. *Something's Gotta Give*. February, 2003, http://www.ddj.com/dept/architect/184414962/
[3] Wikipedia. *Software quality*. http://en.wikipedia.org/wiki/Software_quality/
[4] Wikipedia. *Fault-tolerant design*. http://en.wikipedia.org/wiki/Fault_tolerant/
[5] Wikipedia. *Availability*. http://en.wikipedia.org/wiki/Availability/
[6] Kan, S. *Metrics and Models in Software Quality Engineering*. Addison-Wesley, Second Edition
[7] Wikipedia. *Quality management*. http://en.wikipedia.org/wiki/Quality_management/
[8] Wikipedia. *Risk management*. http://en.wikipedia.org/wiki/Risk_management/
[9] Kan, S. Software Quality Metrics Overview. December, 2002, http://www.informit.com/articles/article.asp?p=30306&rl=1/
[10] Wikipedia. *Fault-tolerant system*. http://en.wikipedia.org/wiki/Fault-tolerance/
[11] Weinberg, G. *Quality Software Management*. Volume 1 Systems Thinking
[12] *Six Sigma – What is Six Sigma?* http://www.isixsigma.com/sixsigma/six_sigma.asp
[13] Wikipedia. *Six Sigma*. http://en.wikipedia.org/wiki/Six_Sigma
[14] Weinberg, G. *Quality Software Management*. Volume 2 Tools for Systems Thinking, Chapter 5 (Causal Loops Diagrams)
[15] Wikipedia. *System Dynamics*. http://en.wikipedia.org/wiki/System_dynamics
[16] Wikipedia. Causal loop diagram. http://en.wikipedia.org/wiki/Causal_loop_diagram
[17] Wikipedia. *Simulation*. http://en.wikipedia.org/wiki/Simulation
[18] XJ Technologies. *Simulation Software and Services*. http://www.xjtek.com/
[19] Cobb, M. *Static and dynamic code analysis: A key factor for application security success*. http://searchsecurity.techtarget.com/tip/0,289483,sid14_gci1185006,00.html
[20] Wikipedia. *Test automation*. http://en.wikipedia.org/wiki/Test_automation
[21] Wikipedia. *Daily build*. http://en.wikipedia.org/wiki/Daily_build
[22] Software Engineering Institute. *Software Inspections*. http://www.sei.cmu.edu/str/descriptions/inspections_body.html
[23] Wikipedia. Software inspection. http://en.wikipedia.org/wiki/Software_inspection
[24] Wikipedia. *Requirements analysis*. http://en.wikipedia.org/wiki/Requirements_analysis
[25] Raishe, T. *Black Box Testing*. http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/black.html
[26] Wikipedia. *Black box testing*. http://en.wikipedia.org/wiki/Black_box_testing
[27] Williams, L. *White-Box Testing*. http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf
[28] Wikipedia. *White box testing*. http://en.wikipedia.org/wiki/White_box_testing
[29] Buzzle.com. *Software Testing – White Box Testing Strategy*. http://www.buzzle.com/editorials/4-10-2005-68350.asp

[30] *What is grey box testing?* http://www.robdavispe.com/free2/software-qa-testing-test-tester-2210.html
[31] Yunus, M. Black, White and Gray Box SOA Testing Techniques & Patterns. http://www.theserverside.com/patterns/thread.tss?thread_id=42231
[32] Wikipedia. *Exploratory test*. http://en.wikipedia.org/wiki/Exploratory_test
[33] Automated QA. *TestComplete – Manual testing*. http://www.automatedqa.com/products/testcomplete/tc_manual_testing.asp
[34] Ford, S. *Automation Testing versus Manual Testing Guidelines*. http://blogs.msdn.com/saraford/archive/2005/02/07/368833.aspx
[35] Wikipedia. Regression testing. http://en.wikipedia.org/wiki/Regression_testing
[36] Wikipedia. *Unit test*. http://en.wikipedia.org/wiki/Unit_test
[37] Wikipedia. *Integration testing*. http://en.wikipedia.org/wiki/Integration_testing
[38] Wikipedia. *System testing*. http://en.wikipedia.org/wiki/System_testing
[39] Wikipedia. *Acceptance test*. http://en.wikipedia.org/wiki/Acceptance_testing
[40] Hildreth, S. *Software QA 101: The Basics of Testing*. http://www.awprofessional.com/articles/article.asp?p=333473&rl=1
[41] Whittaker, J., Jorgensen, A. *Why Software Fails*. Software Engineering Notes, vol. 24, no 4, ACM SIGSOFT, July 1999, 81-83
[42] *Alpha version*. http://www.webopedia.com/TERM/A/alpha_version.html
[43] *Alpha Testing*. http://www.epri.com/eprisoftware/processguide/alphatest.html
[44] Osgood, N. *CMPT 816 Lecture notes*.
[45] Frankovich, J. Software Inspection Process. http://sern.ucalgary.ca/courses/seng/621/W97/johnf/inspections.htm#2.0
[46] Wiegers. Peer Review in Software, Chapter 1
[47] Wiegers, K. *Seven Truths About Peer Reviews*. http://www.processimpact.com/articles/seven_truths.html
[48] *Understandability*. http://www.sapdesignguild.org/resources/simplification/Principles/Understand.htm
[49] Wikipedia. *Dependability*. http://en.wikipedia.org/wiki/Dependability
[50] Wikipedia. *Flexibility (engineering)*. http://en.wikipedia.org/wiki/Flexibility_%28engineering%29
[51] Wikipedia. *Scalability*. http://en.wikipedia.org/wiki/Scalability