Single quality model in software life cycle

RUDITE CEVERE, Dr.sc.comp., Professor
Faculty of Information Technologies
SANDRA SPROGE, Dr.sc.ing., Head of Department
Studies Centre
Latvia University of Agriculture
Liela Street, 2, Jelgava
LATVIA
rudite.cevere@llu.lv sandra.sproge@llu.lv http://www.llu.lv

Abstract: - The article deals with the problems associated with quality definition and assessment for software development processes, intermediate and end products throughout the entire software life cycle. The main objective of the research is to develop measures to improve software quality.
It is known that it is not possible to develop a software which is free from problems. Therefore methods and techniques of software quality improvement are still being developed intensively. Our work is based on years of experience in IT companies and higher education institution. In the information technology sector a lot of attention has been paid to Software Quality Assurance. Our experience has led to the hypothesis that software quality model can be generalized and applied to description and evaluation of quality in a wider area, including quality of the processes. The extended software product quality life cycle is offered including the study process, because during it the future IT professionals acquire their basic knowledge.
When starting any quality evaluation activity, at first it is necessary for all stakeholders to agree on a definition of quality. In our work a single quality model and its application procedure has been developed based on quality model defined in standard ISO / IEC 9126.

Key Words: - Quality Model**,** Software Product Quality, Internal and External Quality Model, Quality Assurance, Study Programme, Study Courses

1 Introduction

Software products nowadays have become widespread and affect quality of functioning in many sectors. A lot of attention has been devoted to definition and improvement of software product quality since the very early days of programming. This has resulted in development of a common approach to quality in software lifecycle. The model of internal and external quality has been defined to describe the quality of the software product [1].

This approach is described in ISO / IEC Standard 9126. The quality model has been obtained by generalizing the number of software quality models developed before [2, 3]. This model has gained quite a wide range of applications, however, it describes only the internal and external software quality. Standard describes also a quality model framework which explains the relationship between different approaches to quality. In accordance with this view software quality in the lifecycle includes process quality, internal quality, external quality and quality in use.

Internal quality is the totality of characteristics of the software product from an internal view. It may be evaluated to a non-executable software product during its development stages (such as a request for proposal, requirements definition, design specification or source code).

External Quality is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing.

Moreover, a wide range of concepts and terms is found in software development related to quality: quality assessment, quality assurance, quality control, quality characteristic, quality certification, quality evaluation, quality improvement, quality management, quality measurement, quality models, quality planning, quality requirements, quality system. Typically, in each case, an understanding of quality can be different [4, 5].

In the field of software development one of the observations is that “graduates do not receive the knowledge and skills needed for industrial software development. It results in low quality and unusable software systems” [6]. In order to improve quality

of the software product we should improve the professional knowledge of IT specialists. Our proposal is to achieve it by creation a single view to the quality throughout the whole software life cycle. This article provides a solution to establish a common understanding of the quality throughout the life cycle on the bases of transformation of the software internal and external quality model.

2 Problem Formulation

Looking at the software quality model development history, it is evident that a hierarchical structure is selected for the models, in which terms that characterize the quality of the software have been placed in various combinations and interconnectedness. Overall view point of which quality characteristics should be selected is various for different models. There is part of the terms that appear almost in all quality models, but there are those that appear only in one particular case [7].

The model described in standard ISO / IEC 9126 can be considered as an international agreement on software quality model and the characteristics used in it. The product quality in ISO 9126 is defined as a set of characteristics and the relationships between them which forms the framework for quality requirements specification and evaluation. Characteristics that affect product operation within its intended environment are known as external characteristics; those relating to the product being developed are called internal characteristics. Model contains 6 characteristics and 27 sub-characteristics (see Fig.1)

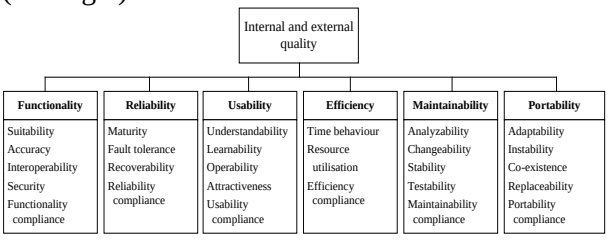

Fig.1 Software internal and external quality model [1]

The model defines the following quality characteristics:

- *Functionality* – the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions;
- *Reliability* – the capability of the software product to maintain a specified level of performance when used under specified conditions;
- *Usability* – the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions;
- *Efficiency* – the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- *Maintainability* – the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications;
- *Portability* – the capability of the software product to be transferred from one environment to another.

In the model each quality sub-characteristic is subordinate to one particular characteristic.

Standards ISO / IEC 9126 and ISO / IEC 14598 recommend in any case of usage to choose only some characteristics and sub-characteristics and make their ranking of importance to a particular application [1, 8]. Experience shows that it is not enough for achieving sufficient mutual understanding on the establishment of quality definition. This is evidenced by numerous variants of using modifications or additions of the standard's ISO 9126 model [9, 10, 11].

Evidence of the need for software quality model improvement is development of a new series of standards 250xx that was launched immediately after Std 9126 TR status was transferred to the distribution. The internal, external and quality of the use approaches to the software product quality are kept in the new quality model too, but the quality characteristics and sub-characteristics have been changed. Now there are 8 characteristics and 37 sub-characteristics (Fig 2).

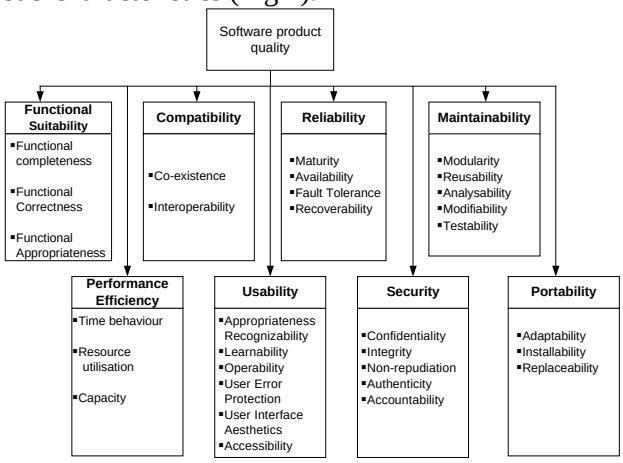

Fig.2 Quality characteristics and sub-characteristics in model ISO 25010 [12]

Analysis of the new quality model shows that a completely new quality characteristics are not offered, but characteristics and sub-characteristics of the previous model have been restructured and new names have been given. A comparison of both ISO models, starting with the name of the characteristic in ISO 9126, and then the characteristic of ISO / IEC 25010 is given below.

Functionality→Functional Suitability.

The name is changed, the current sub-characteristic *Suitability* is added to the name of characteristic; the characteristic has 3 sub-characteristics, two of them are similar to sub-characteristics in previous model (*Functional correctness* with *Accuracy* and *Functional appropriateness* with *Suitability*). Current sub-characteristic *Interoperability* has been moved to position of a sub-characteristic to a new characteristic *Compatibility*. In turn, sub-characteristic *Security* has become a characteristic and 5 new sub-characteristics have been defined for it: *Confidentiality, Integrity, Non-repudiation, Accountability, Authenticity*

Reliability→Reliability.

Names of the characteristic and 3 sub-characteristics are the same, one new sub-characteristic is offered: *Availability* - degree to which a system, product or component is operational and accessible when required for use.

Usability→Usability.

In the new model this characteristic has 6 sub-characteristics. Two of them are identical to previous sub-characteristics (*Learnability* and *Operability*) and new names are offered for two previous ones (meaning of *Understandability* is similar to *Appropriateness recognisability* and *Attractiveness* to *User interface aesthetics*). Sub-characteristics *User error protection* and *Accessibility* are offered as new.

Efficiency→Performace efficiency.

The name of characteristic is changed, the word *Performace* has been added to the previous name. Current sub-characteristics are kept and one new is added: *Capacity* - degree to which the maximum limits of a product or system parameter meet requirements

Maintainability→Maintainability.

Five sub-characteristics are offered. Names of the characteristic and 2 sub-characteristics are the same. Explanation of the new sub-characteristic *Modifiability* - degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality – is quite similar to current *Changeability* - the capability of the software product to enable a specified modification to be implemented. In addition 2 sub-characteristics are offered in the new model (*Modularity* and *Reusability*), and current *Stability* is not used.

Portability→Portability.

Three current sub-characteristics are kept for this characteristic also in the new model (*Adaptability, Installability* and *Replaceability*). Sub-characteristic *Co-existe*nce has been moved as a sub-characteristic to the new characteristic *Compatibility*

--- *→Compatibility*.

The new characteristic has 2 sub-characteristics: current *Co-existence* has been moved from *Portability* and *Interoperability* from *Functionality.*

--- *→Security*.

The previous sub-characteristic of *Functionality* has been moved to the level of characteristics and 5 new sub-characteristics are defined for it: *Confidentiality, Integrity, Non-repudiation, Accountability,* and *Authenticity*

The comparison of models shows that a significant changes are not offered. The terms have been searched that seems more important to stakeholders and appropriate for software quality description and evaluation.

3 Problem Solution

The main goal of our research is to find a form of quality definition, allowing to use a single approach to quality assessment during the entire software life cycle. In order to justify that a transformed quality model of standard ISO / IEC 9126 can be used in different cases of software life cycle, it is necessary to investigate for evaluation of what type of artefacts this model has been developed.

3.1 Software intermediate products

Evaluation of software products in order to satisfy software quality needs is one of the processes in the software development lifecycle. Software product quality can be evaluated by measuring internal attributes (typically static measures of intermediate products), or by measuring external attributes (typically by measuring the behaviour of the code when executed) [1].

Let us see what are the main intermediate products is software development and what are their characteristics.

Software development starts with understanding what needs to be done. This is valid both for a level of large and complex project and for solving a

simple problem report or change request. That means looking for the answer to a question *What to Do*?

For that purpose information is collected, analysed and classified. A variety of methods and tools has been developed: research of industrial activities, studying of documents and real process, interviewing, data and information flow modelling, and so on. Possible forms of documentation of results are also very diverse.

In software engineering looking for answer to the question *What to Do?* is called requirements specification. Often requirements specification is understood as development of *Software requirements specification* document in accordance with the software engineering standards. In fact it is only a one possible way of requirements documentation.

In accordance with software requirements specification standard a good SRS should has characteristics described in Table 1.

Table 1 Characteristics of a good SRS [13]

Quality characteristic	Software/System Requirements Specification
Correct	An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet. There is no tool or procedure that ensures correctness. The SRS should be compared with any applicable superior specification, such as a system requirements specification, with other project documenttation, and with other applicable standards, to ensure that it agrees
Unambiguous	An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.
Complete	An SRS is complete if, and only if it includes a) all significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces; b) definition of the responses of the software to all realizable classes of input data in all realizable classes of situations; c) full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure
Consistent	An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict
Ranked for importance and/or stability	An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement
Verifiable	An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.
Modifiable	An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to a) have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross- referencing; b) not be redundant (i.e., the same requirement should not appear in more than one place in the SRS); c) express each requirement separately, rather than intermixed with other requirements.
Traceable	An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended: a) b*ackward traceability (i.e., to*

Quality characteristic	Software/System Requirements Specification
	previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents; b) f*orward traceability (i.e., to all documents spawned by the SRS).* This depends upon each requirement in the SRS having a unique name or reference number

The next stage in software development is to invent how the defined requirements will be implemented in a program. It means to find an answer to the question *How to Do*? The situation is similar that with the answer to the question *What to Do?* Traditionally, it is a software design development task. Also, there exists a wide variety of methods and tools for design development, and diverse forms for its documentation.

Each user of a design description may have a different view of what are considered the essential aspects of a software design. The proportion of useful information for a specific user will decrease with the size and complexity of a software project. Hence, a practical organization of the necessary design information is essential to its use.

Information about entity attributes can be organized in several ways to reveal all of the essential aspects of a design. In such case the user is able to focus on design details from a different perspective or viewpoint. A *design view* is a subset of design entity attribute information that is specifically suited to the needs of a software project activity.

In accordance with Recommended Practice for Software Design Descriptions (SDD) 10 design entity attributes should be described (see Table 2).

Table 2 Attributes of the design entity [13]

Design entity attributes	Attribute description
Identification	The name of the entity
Type	A description of the kind of entity
Purpose	A description of why the entity exists
Function	A statement of what the entity does
Subordinates	The identification of all entities composing this entity
Dependencies	A description of the relationships of this entity with other entities
Interface	A description of how other entities interact with this entity
Resources	A description of the elements used by the entity that are external to the design
Processing	A description of the rules used by the entity to achieve its function
Data	A description of data elements internal to the entity

A recommended organization of the SDD into four separate design views to facilitate information access and assimilation is given below.

Table 3 Recommended design views

Scope	Entity attributes	Example representations
Decomposition description		
Partition of the system into design entities	Identification, type, purpose, function, subordinates	Hierarchical decomposition diagram, natural language
Dependency description		
Dependency description	Identification, type, purpose, dependencies, resources	Structure charts, data flow, diagrams, transaction diagrams
Interface description		
List of everything a designer, programmer, or tester needs to know to use the design entities that make up the system	Identification, function, interfaces	Interface files, parameter tables
Detail description		
Description of the internal design details of an entity	Identification, processing, data	Flowcharts, Program Design Language (PDL)

The analysis of Standard 9126 shows that *Requirement specification* and *Design Description* are used for input to internal measurements. These metrics give an indication of the expected quality of the software to be developed. The information in Tables 1 and 2 show the formal requirements of software engineering standards to SRS and SDD quality. These are requirements which must be met

in order to consider such documents as a reliable source of information for assessment of software product internal quality.

This means that quality model of a similar structure can be used in the software life cycle in any other case for quality assessment of a non-executable product (the internal quality).

3.2 Extended life cycle of software quality

We offered to extend the life cycle of software product quality proposed by ISO/IEC 9126 standard by inclusion also the study processes (Fig 3). The study period may be considered as one of the preventive measures for the improvement of software product quality.

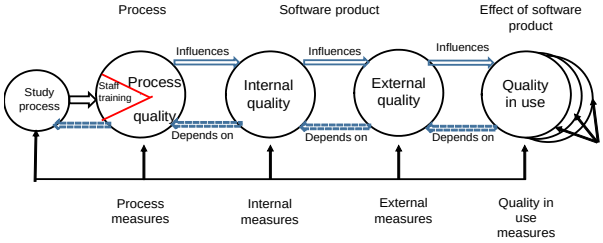

Fig.3 Extended life cycle of software quality

The role of personnel in the development of software product is prescribed also at the level of standard *Software Life Cycle Processes* [15]. The ISO/IEC 12207-2008 standard establishes the human resource management process aimed to provide an organisation with the necessary human resources and competence of human resources suitable for business needs. The basic tasks for the mentioned process include identification of personnel professional skills, skills development plans, assurance of mastering skills, and knowledge management. Therefore teaching the significance and role of quality characteristics in different software product development processes during studies is a kind of preventive actions for the development of quality products in future.

The single quality model has been developed and proposed for usage in the entire software product quality life cycle. It could enhance the establishment of single view on quality through all its stages.

3.3 Development of single quality model

By examining the experience of current software quality model development, it is evident that a much broader set of terms has been used to characterize the quality, than it is included in the standard. Having regard to the fact that, in any case, it is important that all stakeholders have better understanding of the essence of quality the idea to offer an extended set of characteristics was proposed. The most popular variants of software quality model development were investigated and all the used terms were grouped according to their frequency of use in different models. A list of quality attributes used in all quality models during their development is given in Table 6 at the end of the article.

The structure of single quality model was developed (Fig. 4). It represents a step-by-step choice of quality characteristics and sub-characteristics until a satisfactory agreement on the definition of the quality has been reached.

The model has a two-level hierarchic structure:

Level 1 – basic quality characteristics;

Level 2 – sub-characteristics of quality characteristics that are divided into:

- basic sub-characteristics;
- additional sub-characteristics;
- optional sub-characteristics.

One and the same term may be used in different levels and in different relations between characteristics and sub-characteristics.

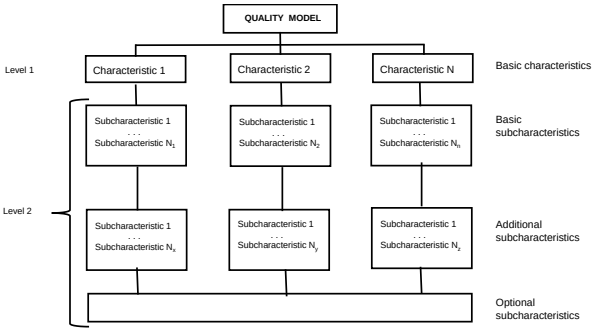

Fig.4 Single quality model

In the quality model relationship of characteristics and sub-characteristics is 1: N, excluding optional sub-characteristics. They have the relationship N: M. This can be explained by the fact that the optional sub-characteristics may be chosen for representation of the various quality characteristics. For example, sub-characteristic *Readability* can be used for both *Usability* and *Maintainability* characteristics.

Functionality, *Reliability*, *Usability*, *Efficiency*, *Maintainability* and *Portability* were chosen as characteristics of the single quality model. These quality characteristics are the same that defined in the ISO 9126 model. The choice of these characteristics is also motivated by the information in the publications of a number of scientific studies that have been carried out on software product quality evaluation based on the ISO 9126 model [16, 17, 18, 19, 20].

For optional sub-characteristics of single quality model it is recommended to use a non-exclusive list

of features. Those who defines quality requirements and carry out an assessment may supplement the list by more understandable to them and quality more accurately characterizing features.

3.4 Suitability of study programme to single quality model

As can be seen from Figure 3, the study process is a part of the full life cycle of software quality. Knowledge and skills acquired by next IT professionals during their studies has significant impact on quality of software products developed by them. So, studies are one of the objects for quality assessment of which the single quality model have been adapted.

As mentioned above, a set of different documents is prepared during the software development. These documents are used also for the evaluation of product quality. Basic documents are requirements specification, design description and source code. A similar set of descriptive documents are prepared for the study programme. They are the curriculum, study plan, and descriptions of study courses. Mutual similarity can be seen in the general documents in both cases.

The study program includes all requirements necessary for an academic degree or professional qualification. The study program is regulated by description of the study content and implementation. According to educational level and type it defines the goals, objectives and expected results of implementation of the specific program. These documents describe the offered curriculum, amount of compulsory, optional and elective parts of the program, learning time allocation, educational evaluation criteria, examination forms and procedures [21].

The course is a description of a study subject or a part of it that is specially organized at a certain level, scope and duration [22].

Software requirements specification and design description are taken as input item for measurements when evaluating internal quality characteristics of software product. Every design entity is evaluated to predict the quality of software end product. As a study course is the smallest unit of the study programme, the total evaluation of internal and external quality of the study programme can be obtained by evaluation of each course separately.

General quality requirements may be defined for documents of study programme similarly to software requirements specification or other software program documents. First of all, they refer to the course descriptions. These requirements shall certify that study programme documents have been prepared at a level sufficient for usage of evaluation of the internal quality (compare Table 1 and Table 4).

Table 4 Characteristics of good study course

Quality characteristic	Study programme and / or course
Correct	A course is correct if, and only if, every topic stated therein is one that corresponds to required scope and level of the particular course.
Unambiguous	A course is unambiguous if, and only if, every described topic is not in mutual conflict with any other topic. It requires at least usage of unified terminology
Complete	A course is complete if, and only if, it includes all significant topics; definition of all practical or laboratory works; describes all kinds of control and contains references
Consistent	Consistency refers to internal consistency. If a course does not agree with some higher-level document, such as a study programme or other courses, then it is not correct
Ranked for importance and/or stability	A course is ranked for importance and/or stability if each topic in it has been identified to indicate either the importance or stability of that particular topic
Verifiable	A course is verifiable if, and only if, every topic stated therein is verifiable. A topic is verifiable if, and only if, there exists some finite cost-effective process with which a person can check that the studying of the particular topic facilitates meeting the goal of the course. In general any ambiguous topic is not verifiable
Modifiable	A course is modifiable if, and only if, its structure and style are such that any changes to the topics can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an course to a) have a

Quality characteristic	Study programme and / or course
	common structure of description, and explicit cross-referencing to literature and other courses; b) not be redundant (i.e. the same topic should not appear in more than one place in the course); c) express each topic separately, rather than intermixed with other topics
Traceable	A course is traceable if the origin of each of its topics is clear and if it facilitates the referencing of each topic in the future development of practical or laboratory works. The following two types of traceability are recommended: a) backward traceability (i.e. to previous stages of development). It depends upon each course explicitly referencing its source literature; b) forward traceability (i.e. to all documents spawned by the course)

In case of study programme, a study course is an analogous element as a software design entity. To ensure implementation of study programme study courses are arranged in groups according to their compliance with sections A, B or C of the study plan. It is necessary to arrange teaching of individual courses in a certain order. In order to build such a structure, the course descriptions require a number of general attributes. Table 5 outlines the attributes of a study course analogues to design entities described in Table 2.

Table 5. Attributes of a study course

Course attributes	Attribute description
Identification	Name of a study course and its code in the information system
Type	Place of a study course in the study plan – Parts A, B or C, and type of control
Purpose	Aim of a study course
Function	Description of the content of a study course
Subordinates	Division of a study course into terms and forms of study work
Dependencies	Relations of a course with other courses
Interface	Preliminary knowledge required to acquire the course, necessity of the particular course for other courses
Resources	Division of lectures and practical work, technical resources necessary for the course
Processing	Requirements of a course for obtaining the credit points
Data	Bibliography necessary for a course content, electronic materials of lectures

The described similarity of the study program and the software product allows to use also similar quality models for evaluation the both objects. Study program quality model can be derived from the developed quality model following the procedure of tailoring the single model.

3.4 Usage of the single quality model

The single quality model shall be tailored to the context of usage and requirements of stakeholders involved in the quality evaluation in each individual case. In all cases, the tailoring shall be done in accordance with the usage procedure of the single quality model (Figure 5)

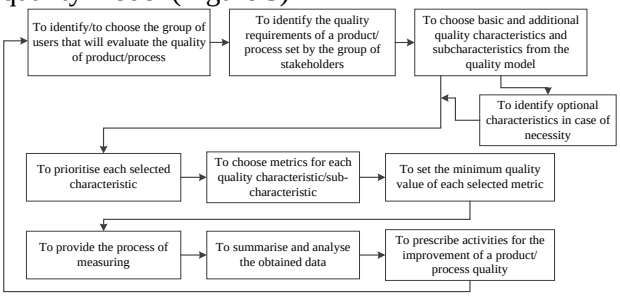

Fig. 5 Procedure of usage of the single quality model

The single quality model is developed so that it may be used in any stage of the entire life cycle of software product. As the study process has been included in this cycle, the model was approbated evaluating the quality of study programme and demonstrating how to include the model into the content of study programme in information technologies. The model was approbated at Latvia University of Agriculture.

The resulting quality model of a study programme is shown in Figure 6. The model encompasses 5 quality characteristics and 21 sub-characteristics. *Functionality, Usability, Efficiency, Maintainability, and Portability* were selected as the basic characteristics

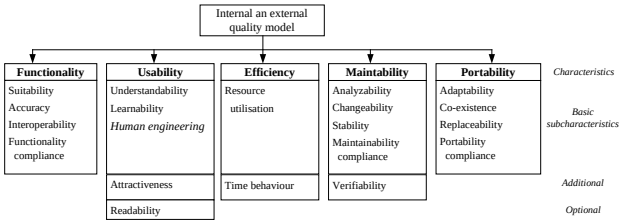

Fig. 6 Quality model of a study programme

In the model of study programme explanations of included characteristics and sub-characteristics are used, which is slightly different from a similar explanations of the software product characteristics. A question is shown for each characteristic to which its use provides the answer.

- **Functionality** (are the necessary knowledge and skills included into the study programme?)
  - *Suitability* – course is topical, conforms to the aims of study direction and labour market requirements;
  - *Accuracy* – it is possible to acquire knowledge and skills included into the programme and they conform to the requirements of the qualification/degree to be awarded;
  - *Interoperability* – learning outcomes of the programme support participation in exchange studies and ensure the necessary knowledge for further studies;
  - *Functionality compliance* – the programme is prepared consistent with the legislation and regulations of higher education, it is accredited.
- **Usability** (is the study programme easy to teach and to learn?)
  - *Understandability* – the curriculum plan is balanced and logically structured, the statement style conform to the level of preliminary knowledge;
  - *Learnability* – acquisition of theoretical and practical knowledge of the programme is balanced, teaching aids are defined and available;
  - *Attractiveness* – oratory skills of the teaching staff are sufficient;
  - *Readability* – descriptive documents of the programme are prepared in good readable language consistent with the regulatory rules;
  - *Human engineering* – degree up to which an individual university lecturer can impact the programme implementation.
- **Efficiency** (is the study programme efficient?)
  - *Time behaviour* – conformance of the programme volume to the time determined for the programme acquisition;
  - *Resource utilisation* – resources necessary for the programme implementation.
- **Maintainability** (is it easy to maintain the study programme?)
  - *Analysability* – it is possible to analyse the curriculum plan and content;
  - *Changeability* – it is possible to modify the study programme without worsening its teachability;
  - *Stability* – it is not necessary to update/modify the content of study programme during the academic year;
  - *Verifiability* – it is possible to verify the achievement of study programme aim;
  - *Maintainability compliance* – the necessary regulatory documents of the programme are prepared, quality evaluation of the programme is available.
- **Portability** (is it easy to adapt the study programme to another audience?)
  - *Co-existence* – the programme does not require specific preliminary knowledge;
  - *Adaptability* – adjustment of the programme to other language or other cultural space does not require redevelopment of the programme;
  - *Replaceability* – degree up to which learning outcomes of the programme may be achieved in another programme;
  - *Portability compliance* – conformance to the regulatory documents governing students' exchange programmes

Quality model of the study programme was used when preparing the bachelor's study programmes to accreditation. Best practices are recommending that the expenses to quality evaluation activities should be small enough in comparison with the possible benefits. This means that in each individual quality assessment it is not recommended to use the full quality model, but select only the most important features. For the internal quality evaluation of the bachelor's study programme courses three characteristics and 10 sub-characteristics were selected (Fig.7).

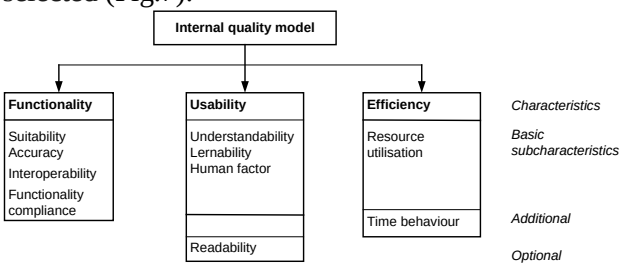

Fig. 7 Internal quality model of study courses

The assessment of internal quality of study courses was carried out during the peer reviews by

all the teachers involved in the program. The participants of the reviews prepared the estimate according to the assessment questions, which were formulated for each sub-characteristic as purpose of the metrics.

For evaluation of the external quality, i.e. the quality of implementation of the programme the model was further modified slightly by reducing the number of quality sub-characteristics. External quality assessment is carried out by students after completion of the course of study. For students a similar survey questions were prepared, the answers to which provides metric values for sub-characteristics included in the model (Fig.8).

In order to provide a more complete understanding of the quality, explanation of the same sub-characteristic may vary slightly in cases of internal and external quality assessment.

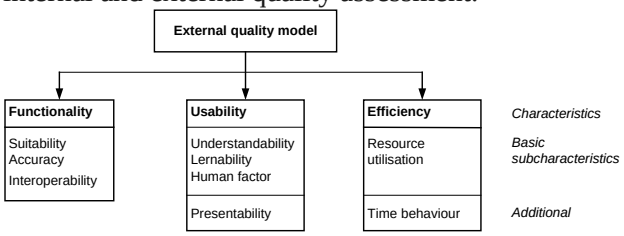

Fig. 8 External quality model of study courses

For example, sub-characteristic *Accuracy* of characteristic *Functionality* for the teachers' evaluation has been explained as a "course is not very fragmented, the number of topics and presentation of the course meets the purpose and scope, the description of independent work is given". For the external quality evaluation (student survey) *Accuracy* is seen as "topics of course meets the defined purpose and content of the course."

4 Conclusion

The quality in the software life cycle begins with the training of next information technology specialists during their studies. It is being developed gradually until, together with an end product creates the software quality in use. The developers must evaluate quality consciously from the very beginning, because it is impossible to put the quality in the end or intermediate product. At least it is much more expensive.

It would be useful for all personnel involved in software development to consider the development as a continuous decision-making process where quality characteristics should be used as decision-making criteria

A hierarchical form of quality model proposed by Standard 9126 may be used to define quality in all other phases of software development, and for other types of products, including the study program.

For harmonization general description of the quality between all stakeholders involved in each quality assessment it may be appropriate to use a wider range of terms than are contained in the generally accepted and standardized quality models. If it can enhance mutual understanding, the same term can be used at different levels and with different interpretations. The most important thing is to reach mutual agreement on the quality definition between all stakeholders at the very beginning of development and to comply with it all the time.

References:

[1] *ISO/IEC 9126–1:2001 Software Engineering – Product Quality – Part 1: Quality Model*. International Organization for Standardization.

[2] J. McCall, P. Richards, G. Walters, *Factors in software quality. volume i. concepts and definitions of software quality*. GENERAL ELECTRIC CO SUNNYVALE CA, 1977

[3] B. Boehm, J. Brown, H. Kaspar, M. Lipow, G. McLeod, M. Merritt, *Characteristics of Software Quality*. North Holland, 1978

[4] *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, 2004, online at - http://www.computer.org/portal/web/swebok/htmlformat

[5] S. J. Khalaf, M. N. Al-Jedaiah, Software Quality and Assurance in Waterfall model and XP - A Comparative Study. *WSEAS TRANSACTIONS on COMPUTERS*, Vol. 12, No 7, 2008, pp. 1968-1976.

[6] K. Lapin, S. Ragaisis, Integrating team projects into the SE Curriculum. *WSEAS TRANSACTIONS on ADVANCES in ENGINEERING EDUCATION*, Vol. 3, No 5, 2008, pp. 104-110.

[7] R. Al-Qutaish, Quality Models in Software Engineering Literature: An Analytical and Comparative Study. *Journal of American Science*, Vol. 6, No 3, 2010, pp. 166–175.

[8] ISO/IEC 14598-2: 2000 Software engineering - Product evaluation - Part 2: Planning and Management. *International Organization for Standardization*.

[9] B. Behkamal, M. Kahani, M. K. Akbari, Customizing ISO 9126 quality model for evaluation of B2B applications. *Information and Software Technology*, Vol. 51, No 3, 2009, pp. 599–609.

[10] P. Berander et al., *Software quality attributes and trade-offs*. Sweden, Blekinge Institute of Technology, 2005, p. 100.
[11] C. Rohleder, Quality Control and ISO Quality Compliance in the Product Lifecycle Management at Siemens. *WSEAS TRANSACTIONS on COMPUTERS*, Vol. 3, No 8, 2009, pp. 469-481.
[12] ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. *International Organization for Standardization*.
[13] IEEE Std 830–1998 IEEE Recommended Practice for Software Requirements Specifications. *IEEE Standards Association.*
[14] IEEE Std 1016–2009 IEEE Standard for Information Technology – Systems Design – Software Design Descriptions. *IEEE Standards Association.*
[15] *ISO/IEC 12207:2008 Information technology – Software life cycle processes*. International Organization for Standardization.
[16] H. Jung, S. Kim, C. Chung, Measuring software product quality: A survey of ISO/IEC 9126. *Software, IEEE*, Vol. 21 No 5, 2004, pp. 88–92.
[17] M. Fam, Y. Luo, G. Wu, X. Fu, An Improved Analytic Hierarchy Process Model on Software Quality Evaluation. *In Proceedings of the 2nd International Conference Information Science and Engineering (ICISE)*. Hangzhou, China, 2010, pp. 1838–1842.
[18] B. Chua, L. Dyson, *Applying the ISO 9126 model to the evaluation of an elearning system.* Proceedings of ASCILITE, 2004, pp. 5–8.
[19] B. Zeiss, D. Vega, I. Schieferdecker, H. Neukirchen, J. Grabowski, Applying the ISO 9126 quality model to test specifications. *Software Engineering*, 2007, pp. 231–242.
[20] I. Padayachee, P. Kotze, A. van Der Merwe, ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems. *The Southern African Computer Lecturers' Association, University of Pretoria, South Africa*, 2010
[21] Law on Institutions of Higher Education. Available at: http://www.likumi.lv/doc.php?id= 37967, 05.01.2015 (Latvian).
[22] Study Regulations in Latvia University of Agriculture (Studiju nolikums) (2008). Available at: http://www.llu.lv/getfile.php?id= 5790, 05.09.2009 (Latvian)

Table 6 Attributes of study course

No.	Characteris-tic	Sub-characteristic	No.	Characteristic	Sub-characteristic
5	*Reliability*	Accuracy	3	*Functionality*	Analizability
5		Completeness	3		Changeability
5		Consistency	3		Maintainability compliance
5		Rubustness/Integrity	3		Stability
5		Self containedness	3		Testability
5		Accuracy	3		Conciseness
5		Frequency/severity of failure	3		Modularity
5		Mean time to failure	4		Self-descriptiveness
5		Predictability	3		Simlicity
5		Recoverability	2	*Reusability*	Generality
5		Fault tolerance	2		Machine independence
5		Maturity	2		Modularity
5		Recoverability	2		Self-descriptiveness
5		Reliability compliance	2	*Testability*	Accountability
5		Accuracy	2		Communicativiness
5		Consistency	2		Self-descriptiveness
5		Error tolerance	2		Structuredness
4	*Efficiency*	Accountability	2		Instrumentation

No.	Characteris-tic	Sub-characteristic	No.	Characteristic	Sub-characteristic
4		Acessibility	2		Modularity
4		Device Efficiency	2		Self-descriptiveness
4		Efficiency compliance	2		Simlicity
4		Resource utilisation	1	*Correctness*	Completeness
4		Time behaviour	1		Consistency
4		Execution efficiency	1		Tracebility
4		Storage efficiency	1	*Flexibility*	Expandability
4	*Portability*	Device independence	1		Generality
4		Self containedness	1		Self-descriptiveness
4		Adaptability	1	*Human Engineering*	Acessibility
4		Co-existence	1		Communicativiness
4		Installability	1		Rubustness/Integrity
4		Portability compliance	1	*Integrity*	Access audit
4		Repliaceability	1		Access controll
4		Machine independence	1	*Interoperability*	Communication commonality
4		Self-descriptiveness	1		Data commonality
4		Software-system independence	1		Modularity
4	Usability	Aesthetics	1	*Modifiability*	Augmentability
4		Consistency	1		Structuredness
4		Documentation	1	*Performance*	Efficiency
4		Human factors	1		Resource consumption
4		Attractiveness	1		Response time
4		Learnability	1		Speed
4		Operability	1		Throughput
4		Understandability	1	*Suportability*	Adaptability
4		Usability compliance	1		Compatibility
4		Communicativeness	1		Configurability
4		Operability	1		Extensibility
4		Training	1		Installability
3	*Functionality*	Capabilities	1		Localizability
3		Feature sets	1		Maintainability
3		Generality	1		Portability
3		Security	1		Serviceability
3		Accuracy	1		Testability
3		Functionality compliance	1	*Understandability*	Conciseness
3		Interoperability	1		Consistency
3		Security	1		Legibility
3		Suitability	1		Structuredness