Alexey Shelkovich

Software Quality Assurance Methods for Enterprise Applications

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa

Institute of Software Technology

Faculty of Computer Science and Biomedical Engineering

Graz, November 2020

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date, Signature

Abstract

Enterprise software is one of the most complex, comprehensive and wide-used software families worldwide. Today it not only merely supports, but to a much greater extent makes it possible to process and seamlessly accomplish within a uniform technology architecture and information system from the very beginning to the end business processes of various nature for companies of different scales, simultaneously providing sophisticated online data management und data analytics possibilities. To be able to do it efficiently and adequately enterprise applications' actual state should be synchronised with all the current trends, forms and practices in industry and various business areas, constantly incorporating in applications and infrastructure all the relevant latest technical advances. As more and more vital functions, including those that have direct influence on human lives, are becoming reliant on the stable, correct and sustainable functioning of enterprise applications, the problems, concerns and tasks of achieving and securing strict long-term quality goals, which in their turn rely on the development and correct application of the appropriate quality evaluation and assurance methods, acquire the utmost importance.

This thesis makes a review of the quality conception views, followed by the analysis of evolution and comparison of software quality models. The details of the most actual and mature standards series in software quality ISO/IEC 25nnn: Software Product Quality Requirements and Evaluation (SQuaRE) are explained, emphasising the possibilities of extension and modification of the provided software quality models and going though the properties and validity criteria of the new or modified quality measures. The latest and the future development trends of the enterprise software, which have a big impact on the quality goals, are demonstrated on the example of SAP solutions as the biggest enterprise software provider. On the basis and as a result of the conducted analysis the important characteristics of hybrid applications are examined and summarised, top-level models of external quality and quality in use for enterprise application have been developed and proposed including a general view of their integration into a software lifecycle and possible re-engineering strategy. The objective is to create the basis for the further development of the evaluation and assurance methods of enterprise applications aligned with and based on the international software quality standards ISO/IEC 25nnn.

Kurzfassung

Unternehmenssoftware ist eine der komplexesten, umfassendsten und am weitesten verbreiteten Softwarefamilien weltweit. Heute unterstützt es nicht nur, sondern ermöglicht es in viel größerem Umfang, innerhalb einer einheitlichen Technologiearchitektur und eines einheitlichen Informationssystems von Anfang bis Ende Geschäftsprozesse verschiedener Art für Unternehmen unterschiedlicher Größenordnungen zu verarbeiten und nahtlos durchzuführen, und bietet gleichzeitig anspruchsvolle Möglichkeiten zur Online-Datenverwaltung und Datenanalyse. Um dies effizient und angemessen tun zu können, sollte der tatsächliche Status von Unternehmenssoftware mit allen aktuellen Trends, Formen und Praktiken in der Industrie und in verschiedenen Geschäftsbereichen synchronisiert werden, wobei ständig alle relevanten technischen Fortschritte in Anwendungen und Infrastruktur einbezogen werden. Da immer wichtigere Funktionen, einschlielich solcher, die direkten Einfluss auf das menschliche Leben haben, auf das stabile, korrekte und nachhaltige Funktionieren von Unternehmenssoftware angewiesen sind, gewinnen die Probleme, Bedenken und Aufgaben der Erreichung und Sicherung strenger langfristiger Qualitätsziele, die wiederum von der Entwicklung und korrekte Anwendung der geeigneten Methoden zur Qualitätsbewertung und -sicherung abhängen, die äußerste Bedeutung.

Diese Masterarbeit gibt einen Überblick über die Ansichten zur Qualitätskonzeption, gefolgt von der Analyse der Evolution und dem Vergleich von Softwarequalitätsmodellen. Die Details der aktuellsten und ausgereiftesten Normenreihen in Softwarequalität ISO/IEC 25nnn: Anforderungen und Bewertung der Softwareproduktqualität (SQuaRE) werden erläutert, wobei die Möglichkeiten der Erweiterung und Änderung der bereitgestellten Softwarequalitätsmodelle sowie die Eigenschaften und die Gültigkeit Kriterien der neuen oder geänderten Qualitätsmaßnahmen hervorgehoben werden. Die neuesten und zukünftigen Entwicklungstrends der Unternehmenssoftware, die einen großen Einfluss auf die Qualitätsziele haben, werden am Beispiel von SAP-Lösungen als größtem Anbieter von Unternehmenssoftware demonstriert. Auf der Grundlage und als Ergebnis der durchgeführten Analyse werden die wichtigen Merkmale von Hybridanwendungen untersucht und zusammengefasst. Es wurden Top-Level-Modelle für externe Qualität und Qualität im Gebrauch entwickelt und vorgeschlagen, einschließlich eines allgemeinen Überblicks über deren Integration in einem Software-Lebenszyklus und mögliche Re-Engineering-Strategie. Ziel ist es, die Grundlage für die Weiterentwicklung der Bewertungs- und Sicherungsmethoden von Unternehmensanwendungen zu schaffen, die an den internationalen Softwarequalitätsstandards ISO/IEC 25nnn ausgerichtet und auf diesen basieren sind.

Contents

List of Figures

List of Tables

List of Acronyms and Symbols

SAP SAP SE is a German multinational software corporation that makes enterprise software to manage business operations and customer relations. Also used to reference the whole range of software development by SAP SE
ISO International Organization for Standardisation
IEEE Institute of Electrical and Electronics Engineers
FCM Factor, Criteria, Metrics Quality Model
FURPS Functionality, Usability, Reliability, Performance, Supportability Quality Model
GQM Goal Question Metric Quality Model
SQuaRE Systems and software Quality Requirements and Evaluations (ISO definition)

1 Introduction

Development, maintaining and using an enterprise software of high quality are complex multistage processes, which imply carrying out a big number of different tasks and solving countless unpredictable problems, challenges and issues. Enterprise applications are used in and supporting, sometimes even controlling to the extent of dependence on them, almost all areas of enterprise activities, many of which are of extremely importance for the existence of a company itself, including those that might be potentially dangerous and threatening for human lives. In the development of such a software a lot of people, organisations and resources are involved and huge amount of money is spent.

The recent trends, such as globalisation, strong international market competition and ubiquitous digitalisation, led to the shift of focus from mostly pure technical and functional view on the enterprise software to the human-centred view, to the end user, who comes form all business areas around and stays now in the middle of attention, influencing all processes starting with methods of interaction with a software and even the ways how the software should work. The software is starting to show its "human face".

This turn together with an explosive spread and development of technologies brought with itself many new, before unknown, types of users into the area of enterprise software: today not only long-familiar technical and business experts, possessing solid education and many years of experience, but also a lot of specialists from other domains, often completely technically ignorant but still needing the possibilities provided by enterprise applications, have to use it and are dependent on it for the success of their everyday routine. The feedback from end-users and the requests for changes for a new functionality are coming now much faster, in much bigger scale and it is assumed that the results and new solutions will be delivered fast, seamless for already running processes and with achievement of high quality goals. As a response to these challenges in the last decade completely new agile methods and practices were incorporated and adapted into a software engineering practice. Nobody wants and has enough time anymore to wait before the complete thorough specification would be written and compete development cycle would be finished and integrated into the production: all this together could take years. Agile methods are aimed at rapid development and deployment in production ready-to-use versions is short development cycles.

And exactly at this time the quality of these applications becomes of utterly importance, not only to ensure that all business processes run correctly and efficiently, implemented solutions are satisfactory and do exactly what is excepted from them, but also to protect the whole extreme complex system from potential risks, errors and inconsistencies.

It is rather impossible to define exactly and describe precisely what the quality is because of its ubiquitous and inborn nature. There are many ways of treating it and the

influencing factors include the scope of application, personal perception, cultural and industry differences, context in which the quality concept is used and also many others. For the area of interest and tasks of this thesis under the quality would be understood a degree to which a set of inherent characteristics of an object fulfils requirements [Sta15a], and this definition applied to the enterprise software applications quality implies the capability of software product to satisfy stated and implied needs when used under specified conditions [Sta14].

The before mentioned challenges, importance and complexity of the tasks solved by enterprise applications discover und highlight the urgent need in research aimed at development of quality models, algorithms and methods of enterprise applications quality assurance. Quality assurance of enterprise software is a multi-criteria process, including but not limited to developing of the quality models, the definition of set of quality criterias, choosing the etalon values, measurement and evaluation of real values, evaluation and assurance of the enterprise software quality. In practice quality evaluation processes are often carried out by experts or are based on personal judgements of the staff involved in the development or support, what in most of the cases gives the results which are not objective enough. Chosen criterias are mainly descriptive, the selection of base and etalon values is rather intuitive and measurement and comparison processes are difficult to formalise.

To archive the quality goals and to assure the quality of enterprise applications it is necessary to evaluate the quality constantly, a set of criterias used for the measurements and their appropriate base values should be regularly reconsidered and appropriately adjusted in accordance with the progress of development, testing and real use in production environment. These might be a very ambitious and demanding tasks taking into account the variety of users, business cases and staying behind them multiple business roles with many business-processes and applications, interacting with each other and running on a heterogeneous set of end-user devices in a countless number of possible contexts. In this thesis a number of tasks were set to come closer and tackle the mentioned objectives:

- Analyse the quality conception, the evolution and development of approaches and software quality models used in quality assurance processes;
- Review and analyse the most mature and recent standards in the area of software quality ISO/IEC 25000 SQuaRE series;
- Show the trends of the enterprise software development directions in the last decades on the example of SAP as the largest enterprise software supplier in the world;
- Develop and propose the top-level external quality model and top-level quality in use model for enterprise applications based on the actual SQuaRE standard series.

The second chapter of the thesis presents the broad range of quality definitions, which were constantly changed and clarified with time, and shows the ways of how these various quality interpretations were formalised in the form of software quality models, giving their short analysis and a final comparison of the properties and parameters.

Third chapter gradually continues this overview by getting into details of the latest mature international standards in the software quality ISO/IEC 25000 series. Starting with the description of the standards' structure, the conceptions of internal, external quality and quality is use are explained followed by the presentation of the corresponding reference quality models. Shortly the basics of quality qualimetry are explained and the important question of extension and adaptation of the reference models, including measures and such important concerns as their desired properties and criteria for the validity, is clarified.

Fourth chapter deals with the enterprise software, firstly presenting how SAP enterprise solutions have evolved, what are the current and future trends of changes and developments, which, in their turn, will have a big impact on how the software is being used, on future quality goals and on quality assurance methods. The analysis of characteristics of hybrid enterprise applications highlights the important aspects needed to be considered while development, usage and quality assurance processes. Finally, based on all of the conducted researches, top-level models of external quality and quality in use of enterprise applications quality are proposed, shortly touching the aspects of their integration into software lifecycle and possible re-engineering strategy.

2 Evolution of quality conception and software quality models

2.1 Quality definition

One of the challenges of quality assurance is that the quality definition itself is not exact and obvious. It happens that quality can be understood differently, context-dependent, sometimes subtle or false. The main reasons for this are:

- Quality is not an isolated idea, but it a complex multi-level conception.
- Different people perceive the quality with different abstraction level. In some cases quality is seen as a broad idea, in others as a concrete values of defined separate characteristics.
- The term quality is used frequently in all areas of human life, not only in software area, which makes is difficult to focus of exact definition.

If we try to analyse how the idea of quality if used in most of the cases (for example expressions or, better to say, broad evaluations as "good" or "bad" quality) it becomes clear, that the majority treats the quality as something abstract und undefined, uncontrollable and therefore not really measurable. However, the professional and scientific approach to the quality allows not only to measure and control it, but also manage and improve it.

One more common misconception is that quality inextricably connected with high complexity, high costs and sophistication of the product. According to this view a simple, inexpensive and effective program may rare be assessed as one having a high quality level.

As such perception of the quality makes it impossible to use it for constant improvement of the processes of software development, specialists and scientist gave a number of more precise and convenient definitions.

Juran has defined quality as "fitness for use" [JJ88]. What is important that this definition takes into consideration the requirements and expectations of the users of the software.

Later Crosby [Cro79] described quality as "conformance to requirements". This definition is based on maximal clear detailed requirements. Unconformity of the product to this requirements is considered by Crosby as defect and otherwise full conformance means maximal level of quality of the product.

Afterwards a number of definitions of quality were given. Hamphrey [W.S89] defined quality as achievement of excellent level of suitability for use, IBM introduced the

"market-driven quality" and in Baldrige National Quality Program "customer-driven quality" idea is used.

Pressman [R.97] gave the definition of the quality as conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software. Later [R.09] he has given one more definition: software quality can be defined as an effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

The definition given in [AG15] is: *quality* is a property representing a set of those and only those properties that characterise the consumption results of an object, both desirable and undesirable, excluding the cost of their creation and consumption. That is to say, this set includes only properties associated with the results achieved in consuming an object, and does not include ones associated with the cost of providing these results. The *property* in turn is defined as: a feature, characteristic or peculiarity of an object, that becomes apparent during its consumption/operation/use/application (henceforth, all these terms are used interchangeably) according to the purpose of its use (e.g., the mean lifetime of a community).

Fitzpatrick [R.96] defined software quality as the extent to which an industry-defined set of desirable features are incorporated into a product so as to enhance its lifetime performance. It worth mentioning that in this definition quality has been given a time dimension.

The most accurate definitions of quality have been formulated and recorded in international standards ISO (International Organization for Standardisation). Standard ISO 9000:2015 [Sta15a] defines quality as "**degree** to which a **set** of **inherent** characteristics of an **object** fulfils **requirement**s". From this definition one can deduce that quality is a complex, multi-dimensional concept:

- **degree** implicate that quality is a variable and not fixed;
- **set** means that quality is not an atomic characteristic;
- **inherent** means that quality exists in object itself, cannot be detached from the object as another independent object and ist not assigned by external force;
- **object** means anything perceivable or conceivable therefore the term quality can be used relative to both tangible and intangible objects, for example quality of software;
- **requirement** means a need or expectation that is stated, generally implied or obligatory.

To put it all in simple words - to what extent the object corresponds to our expectations.

In relation to the area of information technologies and software, the exact definition of the software quality was first given in standard ISO/IEC 9126-1 [Sta01] and afterwards in the newest standards of series 25000 SQauRE [Sta14]. The quality conception there is analysed from different perspectives, it is described in detail in 3.

data quality

degree to which the characteristics of data satisfy stated and implied needs when used under specified conditions

quality in use

degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use

software quality

capability of software product to satisfy stated and implied needs when used under specified conditions

The historical approach to the quality definition mainly focuses on the compliance of the software to the functional requirements. However, since the publishing of international standards 9126- and later 2500-series, the focus of the definition has been shifted. In according to the standards the quality is defined thought the set of characteristics, which to a greater extent *reflects the expectations and needs of the real users* than the technical conformity to the documentation and requirements.

2.2 Development of software quality models

The main research domain in the area of software quality assurance is the formalisation of quality attributes and methods of their evaluation. To support, facilitate and regulate these processes in the middle of 1970 the first quality models were created and have been constantly evolving since then. The most mature and modern international standard of software quality ISO/IEC 25000 [Sta14] characterises quality model as a set of indicators, i.e. attribute categories which are related to the software quality, and relations between them, what provides the basis for quality requirements and quality evaluation specification.

Further follows the overview and analysis of the most important and known quality models, approaches to quality concepts, which have been developed in the last four decades. Some of them aimed at solving some specific corporate tasks, others were used in heterogeneous projects of software developing and the other ones inspired and became part of the industry, state or international standards of quality.

2.3 Factor-Criteria-Metric Models

Models of software quality possess hierarchical structure in their nature. The main advantage of this is the possibility to decompose every quality attribute into the number of factors, which, in their turn, can be further decomposed to the number of criterias. Criterias can be afterwards matched with the set of appropriate metrics and therefore evaluated. Such models are named after the first letters of their components as **FCM** (**F**actor, **C**riteria, **M**etrics) Models.

One of the first wide-known quality model became the work of McCall [MJ77]. Here the quality indicators are divided into three groups:

- factors - describe the software from the users' point and provide the requirements, external software view

- criterias - describe the software from the developers' point and provide the targets, internal software view
- metrics - used for qualitative description and measurement of quality.

Criterias of the quality (overall 11) are combined into three factor-groups in accordance with the different methods of interaction of users with software. Revision identifies the ability to undergo the changes, Transition characterises the adaptability to new environments and Operations are operational characteristics. The resulting structure can be represented in so-called triangle of McCall (figure 2.1).

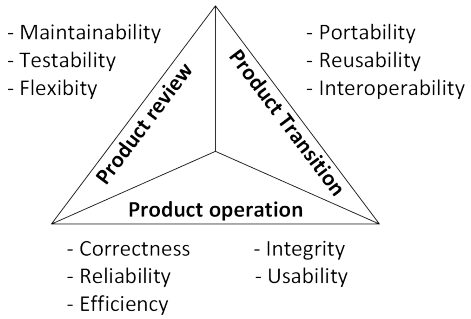

Figure 2.1: The triange of McCall (from [MJ77])

Criterias of quality are the quantitative values of the factors, which are set as targets for the software development. It is difficult to make objective evaluation of the quality factors directly, so the metrics of the quality were introduced to allow measurement and valuation. The values of metrics are getting values from 0 to 10, each metric can influence several quality factors and the value of the factor is calculated as a liner combination of the related metric values. Appropriate coefficients are defined differently for different organisations, development teams, types of software, used processes etc. The main aim of the model application in real project is *the achieving of the measurability of the software quality attributes.* The model is shown on the figure 2.2.

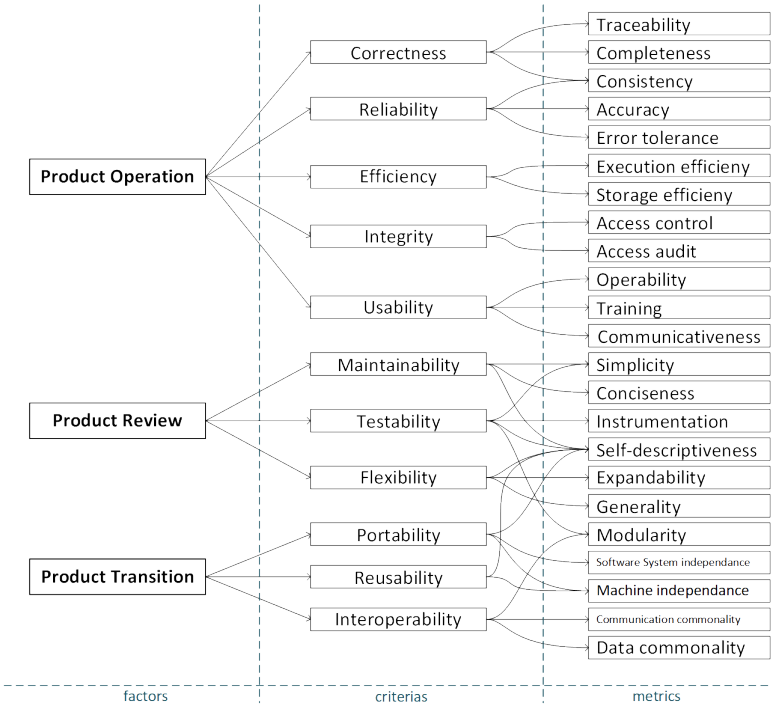

Figure 2.2: McCall Quality Model (from [MJ77])

Boehm [BB78] has introduced the model, which is basically the extension of McCall model. It introduces high-level factors such as as-us utility, maintainability and portability, and additional characteristics (criterias) also including those dealing with the performance of hardware. The model is shown on the figure 2.3.

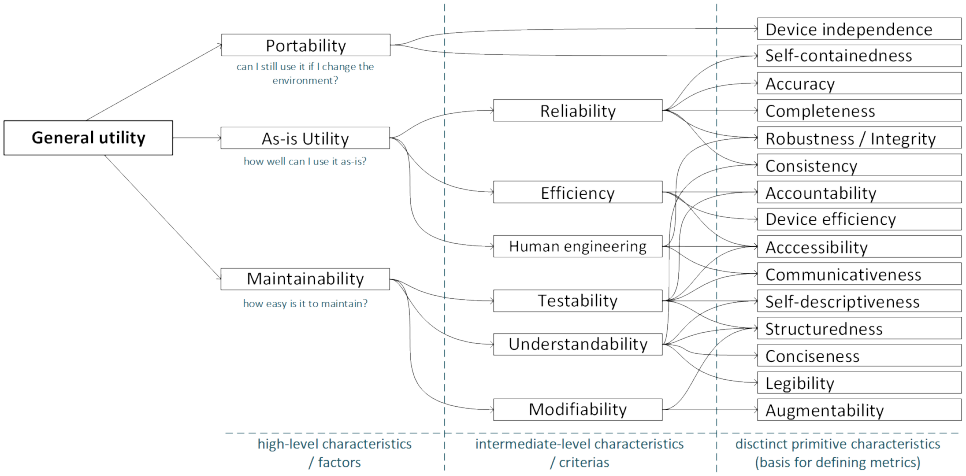

Figure 2.3: Boehm Software Quality Characteristics Tree (from [BB78])

Hewlett Packard company in 1987 proposed **FURPS** software quality model [GR87], which is an industrial interpretation of McCall and Boehm models. It differentiates between *functional requirements*, which include the input and excepted output (**F**unctionality), and *non-functional requirements*, which are defined by several attributes (**U**sability, **R**eliability, **P**erformance, **S**upportability). These 5 high-level attributes are detailed with 27 low-level attributes. Laster the model was extended by IBM Rational Software to FURPS+ which includes additional non-functional constraints: design constraints, implementations constraints, interface constraints, physical constraints. The model is shown on the figure 2.4.

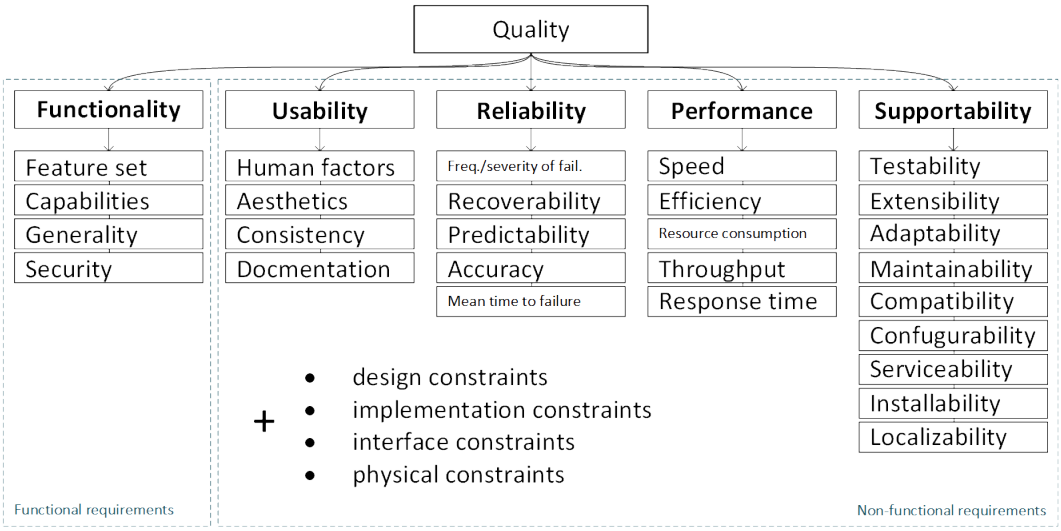

Figure 2.4: FURPS+ Software Quality Model (from [GR87])

The model from Gilb [T.88] corresponds in general the concept of FCM-Model but has some major differences. The quality model is incorporated into project documentation, and every attribute must be measurable and must be further specified in the processes of software lifecycle. Apart from quality attributes the models also includes restriction attributes. The model is based of four quality and four resource attributes and it is possible to extend them (see figure 2.5).

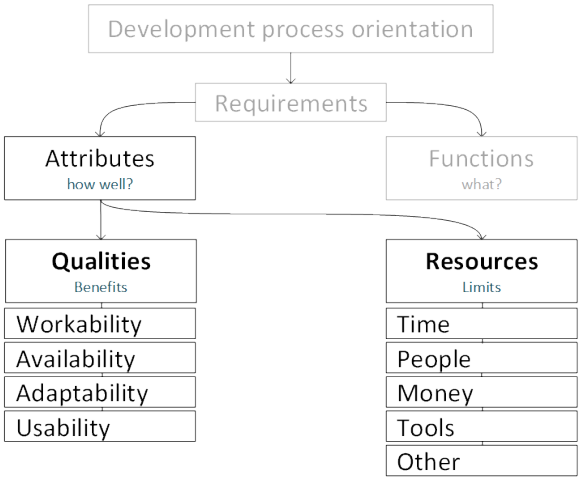

Figure 2.5: Gilb Quality Model, fragment (from [T.88])

Gilb proposed seven principles of model use, main from them is the principle of measurability: all attributes may and must be measurable on practice. Gilb says, that it is always possible to find the appropriate measurement for an attribute. If no measurement for some attribute can be found, it means, that it is not an attribute but an indicator, which needs to be further decomposed.

2.4 Goal-Question-Metric Models

The Goal-Question-Metric model **GQM** was developed to identify defects of the software in center of space flights at NASA [BV94]. The main purpose of the model is to facilitate making decisions and controlling the processes of software development on the basis of measurements. The GQM Model provides the basis for translating the goals of the software development to the set of questions and metrics. This model also has 3-level hierarchical structure (see figure 2.6).

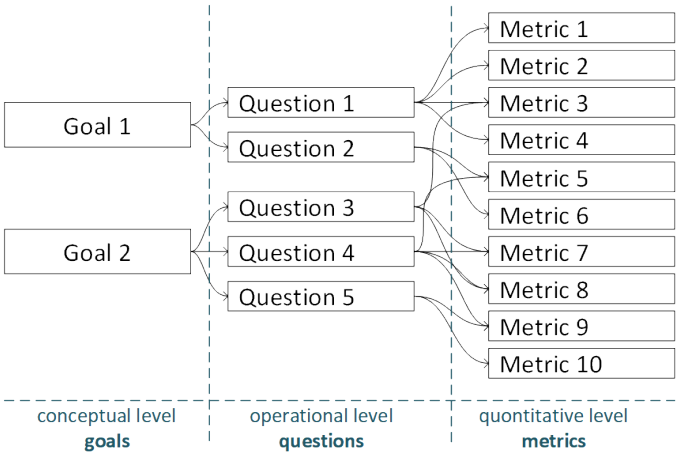

Figure 2.6: GQM Quality Model (from [BV94])

1. conceptual level (goals) - goals are the abstract attributes, which define the direction of the development and can be matched to different objects (software, development processes, resources of development etc.). For example reliability, testability, portability etc.
2. operational level (questions) - questions are used to describe the methods to achieve the goals. The questions characterise the objects for measurement (software, processes, resources etc.) in accordance with the chosen quality factor and describe the quality from some definite point of view.
3. quantitative level (metrics) - metrics are the procedures, formulas or algorithms, wich can be used to answer the question in a quantitative way. Metrics can be objective or subjective.

In accordance with the model every question can be matched only to one goal, but metrics in their turn can be matched to different questions.

The models GQM and FCM can be compared with each other. Goals and questions correspond to factors and criterias. It needs to be mentioned that FCM are general-purpose models which implies their usage in different projects, whereas with GQM-type models different projections of the same project are created. Several GQM models for one project can have common questions and metrics, if the are different projections of the project evaluation. There are a number of model application in different business areas [BA02], [HL96], [V.R93].

2.5 Process-Product Models

Some of the models were developed not only for software quality, but also for controlling the process of software development. This methods are mainly focused on the evaluation of the internal software attributes and their influence and mapping to external attributes.

The main works in this direction are the Dromey model [R.G96] and SQUID method [BJ99].

The Dromey model was first published in 1996 [R.G96]. According to the Dromey's approach the high-level quality indicators such as reliability, maintainability, portability and others cannot be provided and guaranteed directly by the software itself. Instead of this it is necessary to define the set of properties of the software, which, if the are fulfilled, lead in turn to the desired level of values of high-level quality indicators. The Dromey model is also hierarchical in nature but significantly differs from other hierarchical quality models. Hier by building the model the bottom-up approach is applied . In the first turn the set of components, which comprise the software, is fixed. Some of them can be atomic, other be complex and consist of other components. In the second turn the properties of each component, which have the influence on the quality of component, are defined. Finally to describe the quality of software the set of non-intersected, comprehensive and aligned high-level quality attributes is introduced (see figure 2.7).

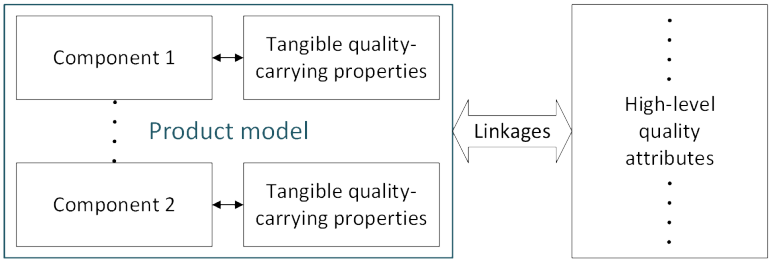

Figure 2.7: Dromey Quality Model, elements (from [R.G96])

The quality model is complemented by linkage between properties of the software and high-level quality attributes. Each such a linkage must be verified experimentally on practice for each property of the software. The linkage between software properties and the attributes of the composite components of the software defines the quality of the software as a whole. So the whole process can be described in the following steps: choose the representative set of high-level quality attributes for evaluation, list components of the software, identify quality-carrying properties for each component, understand how each of these properties affects the quality attributes and finally evaluate the whole model to determine weaknesses.

Because Dromey's approach was focused on supporting the development of software, three quality models were introduced: quality model of requirements, quality model of the project and quality model of the implementation. Each model is related to the appropriate phase of software lifecycle and has own set of attributes.

SQUID method [BJ99] covers the processes of planning, controlling and evaluation of quality. SQUID defines quality as a behaviouristic characteristic of software needed by users. Similar as already before mentioned models SQUID defines quality in terms of high-level attributes, which must be detailed or decomposed until they can be measured. SQUID does not define its own quality model and it is suggested to use any existing model what is the most suitable for the concrete projects requirements (see figure 2.8).

Figure 2.8: SQUID Quality Model, elements (from [BJ99])

Quality attributes which define the behaviour of the software are called external quality attributes, for example reliability, simplicity of use. The attributes which are related to the software development and software itself are called internal attributes, for example test coverage, structural complexity, size. Before the method can be used it is necessary to define how internal attributes affect external attributes by discovering the linkages between them in the quality model.

SQUID method describes quality requirements for software by setting goals for the external quality attributes. Afterwards these goals are realised by defining and tracking the goals for the internal quality attributes. In turn this is achieved by measurement of the internal attribute values.

2.6 Standard IEEE 1061

Standard IEEE 1061 was first published in 1998 by the Institute of Electrical and Electronics Engineers [EE98]. The standard describes relative open and flexible hierarchical structure of the quality model, which has quality factors on the top level and which are decomposed to sub-factors and metrics. Each level can be decomposed to sub-levels (see figure 2.9).

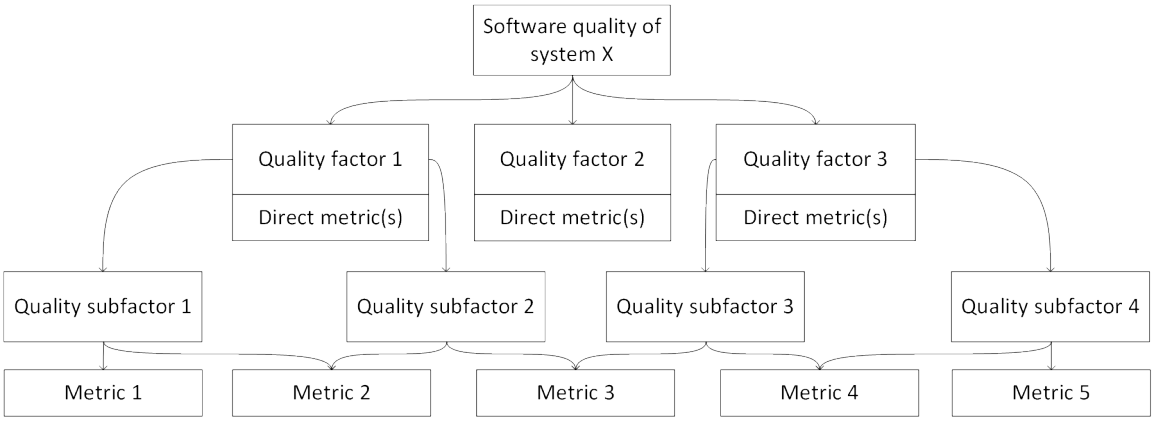

Figure 2.9: IEEE1061 Software Quality metrics framework (from [EE98])

Opposed to the standards ISO/IEC 9126-1-4 [Sta01], [Sta03b], [Sta03a], [Sta04], where the set of quality attributes is fixed, this standards does not limit the number of factors and sub-factors and allows to add or remove an arbitrary number of them. The standard also contains the description of method of using it together with the GQM model (see 2.4). Interesting feature is the possibility of direct evaluation with the help of the metrics of top levels' factor and sub-factors in the hierarchy. It also means that it is possible to decompose high-level elements of the quality model to the sets of multi-rank components.

2.7 Standards ISO/IEC 9216, ISO/IEC 25000

Today the most mature and actual standards for software quality is ISO/IEC 25000 series, which replaced standards of ISO/IEC 9216 series. ISO/IEC 2500 is described and analysed in 3, in this section is given the description of ISO/IEC 9126 series.

The standards use the concept of external quality, internal quality and quality in use.

External quality defines the ability of the software system to show such a behaviour, which would satisfy the implicit and explicit needs by using the system in a given context.

Internal quality is the ability of the set of statical attributes of software system (i.e. inherent properties or characteristics of the essence, which can be characterised qualitative or quantitative by the human or automatically) to assure the implicit or explicit needs by using the system in a given context.

Quality in use in the extent to which particular software users can achieve their goals in a context of effectiveness, productivity, security and reliability.

The model of external and internal quality divides the software quality into six main characteristics, each of which is detailed by subcharacteristics (see figure 2.10).

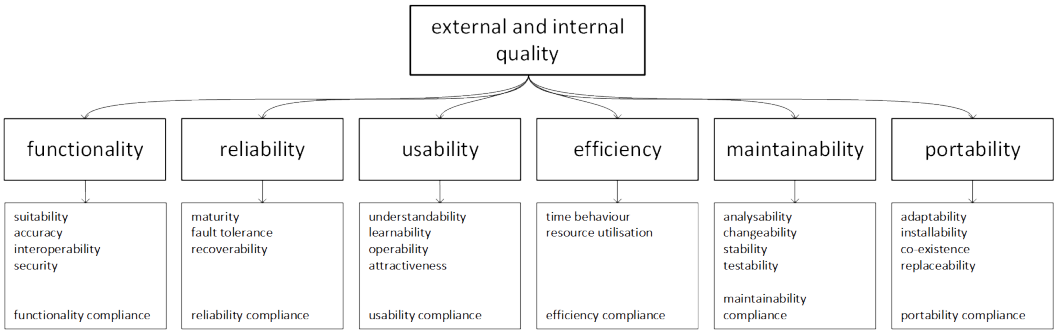

Figure 2.10: ISO/IEC 9216-1 - Quality model for external and internal quality (from [Sta01])

The subcharacteristics in turn can be measured by the appropriate internal or external metrics.

Table 2.1: ISO 9216-1 - external and internal 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.

To describe the quality in use other model and characteristics are used - see figure 2.11.

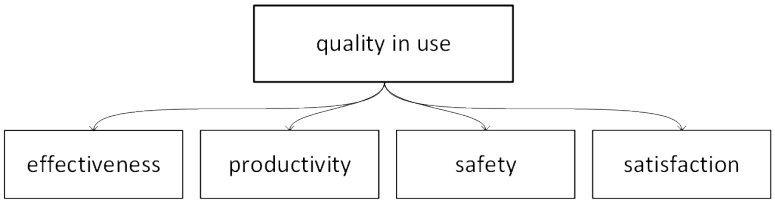

Figure 2.11: ISO/IEC 9216-1 - Quality model for quality in use (from [Sta01])

Table 2.2: ISO 9216-1 - quality in use characteristics

Effectiveness	The capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use.
Productivity	The capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use.
Safety	The capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.
Satisfaction	The capability of the software product to satisfy users in a specified context of use.

2.8 Comparison of the structure of the quality models

The structural comparison of the described quality models is given in the Table 2.3. All of these model have a tree-like structure formed by high-level characteristics which are detailed by low-level characteristics until the decomposition reaches the atomic and measurable attributes or metrics. Despite of the different terminology used it is not difficult to identify the equivalent components of the models. In different models the number of hierarchical levels and characteristics and attributes are different and can be also fixed or flexible. The quality models are being developed as the basis for quality evaluation and they have in their structure not only conceptual elements - characteristics of different levels - but also measurable - metrics. The early models are more qualitative that quantitative in their nature because the knowledge about metrics was not enough developed. Later models show more strict approach to the evaluation and measurement of quality. The main direction of quality model development is to make all of the quality attributes measurable and to split between different levels qualitative and quantitative attributes.

2.9 Comparison of the build methods of the quality models

In accordance with a standard or an approach a quality model can be rigid, where the set of attributes and connections between them are fixed, or flexible, where it is specially

build for some project depending on the the view of customers, users and developers [FN00]. The comparison is given in the Table 2.5.

2.9.1 Strict approach and rigid quality models

In case of strict approach the same final rigid model is used for all software projects for quality evaluation. It is assumed that the model contains the whole set of all the necessary quality indicators and for the evaluation of some particular software an appropriate subset can be chosen. Examples of such models are McCall (see 2.3), Boehm (see 2.3), FURPS (see 2.3). The advantage of the rigid models is:

- it is possible to get comparable common views of quality for potentially different software.

There are also some disadvantages:

- lack of flexibility and reliance on the fact, that all quality attributes required for quality evaluation of some special software must be the subset of the attributes of the model. This is not always achievable because in reality in spite of the universal nature of some attributes other ones are strongly dependent on the software type;
- difficulty in definition and distinction of quality attributes which lead to partial match of criteria related to different factors.

2.9.2 Agile approach and flexible quality models

Agile approach allows to create own flexible quality models for one or more software projects. As the basis the attributes of existing models are used and if necessary own attributes can be introduced. There are no limitations on the number of hierarchy levels or connections between attributes. Examples of such models are Gilb (see 2.3), GQM (see 2.4) und SQUID (see 2.5). The advantages of the flexible models are:

- possibility to reach maximal consideration and coverage of the project features and requirements in quality model;
- more precise and adequate evaluation of software quality by using more detailed and specialised attributes on low-levels of the model.

There are also some disadvantages:

- additional costs for development of quality model for each project;
- potentially lower level of results' correctness because the model is developed by local groups of project specialists.

2.9.3 Mixed approach and overlapping of levels

It is possible to combine the advantages of both approaches (see 2.9.1, 2.9.2). The models of the standards ISO/IEC 9126 and ISO/IEC 25000 can be treated as mixed ones. One can use a subset of top-level characteristics from the standard and introduce own project-specific attributes on low-levels.

On some models, for example early models McCall or Boehm, it is possible to see multiple connections form criteria to factors what actually represents not tree but network structure of the hierarchy of quality characteristics. We can call this an overlapping of levels. This is a natural phenomenon which can be explained by the nature of the software itself [R.G95]. Model FURPS is an example of the model without levels overlapping. The models of the standards ISO/IEC 9126 and ISO/IEC 25000 are mixed ones, because they do not allow overlapping on top-levels but allow it on low-levels.

2.9.4 External and internal quality characteristics

A number of models does not make a difference between internal characteristics, which define the structure and behaviour of the software from the point of view of a developer, and external characteristics, which reflect the point of view of the user of the software. All such characteristics exist in one model and form one set of characteristics of one or another level of hierarchy. An opposite to it are models where exact distinction between external and internal characteristics is made. The importance of such approach can be seen in [BJ99], where the development process, considered by internal characteristics, is controlled by the values of external characteristics. The standards ISO/IEC 9126 and ISO/IEC 25000 define completely different models for external quality, internal quality and quality in use.

2.9.5 Relationships between quality attributes

It is reasonable while developing a quality model to consider the relationships between quality characteristics (factors) on the same level. There are a number of works which have defined such a relationships. In the figure 2.12 the example of relationships "neutral", "direct", "inverse" is shown [W.87].

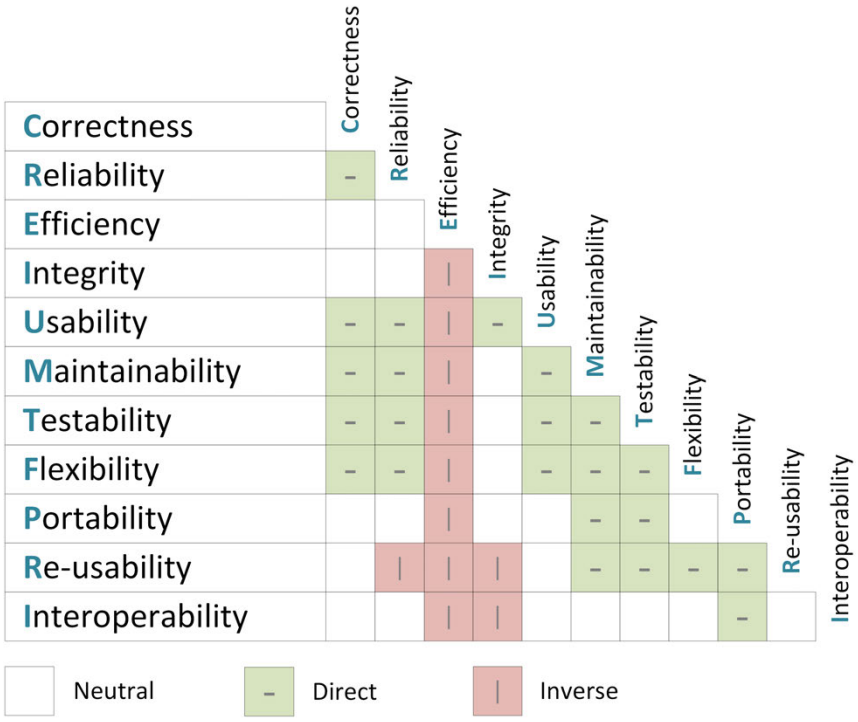

Figure 2.12: Quality attributes relationships (Perry) (from [W.87])

In the figure 2.13 the relationships from standard IEEE 1061 are demonstrated.

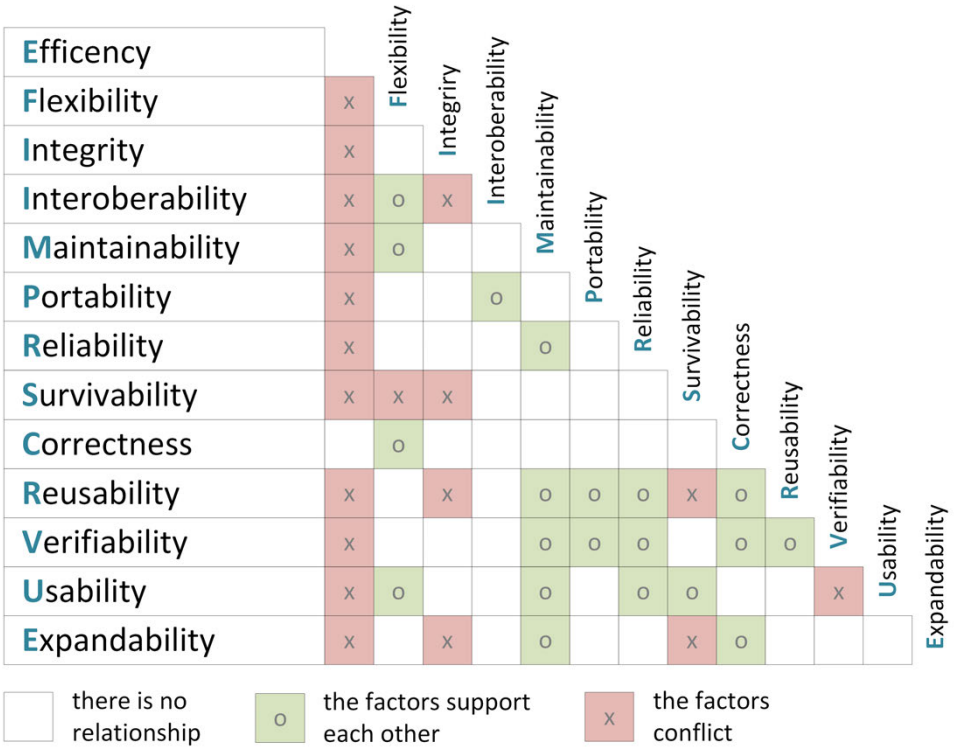

Figure 2.13: Quality factors relationships (IEEE 1061) (from [EE98])

It worth mentioning that there are more studies which define more detailed relations between quality attributes such as "realises", "destroys", "supports", "violates", "unknown", "equivalent", "rather support", "rather violets" [CL03], [L.99].

Table 2.3: Comparison of the Quality Models structure

Quality model	Types of elements of the model	No of levels	Number of elements on each level	Metrics set
McCall	Factors	2	1 level - 3, 2 level - 11	no
	Criterias	1	23	
	Metrics	1	user-defined	
Boehm	High-level characteristics	2	1 level - 1, 2 level - 3	no
	Intermediate characteristics	1	6	
	Primitive characteristics	1	15	
	Metrics	1	user-defined	
FURPS	Quality attributes	2	1 level - 5, 2 level - 27	no
	Metrics	1	user-defined	
GQM	Goals	1	user-defined	no
	Questions	1		
	Metrics	1		
Gilb	High-level attributes	user-defined	user-defined	no
	Low-level attributes	user-defined		
	Metrics	user-defined		
Dromey	High-level attributes	user-defined	user-defined	no
	Quality-carrying attributes	user-defined		
	Components	user-defined		
SQUID	Quality characteristics	user-defined	user-defined	no
	Quality attributes	1		
	Components	1		
IEEE 1061	Factors	user-defined	user-defined	no
	Sub-factors	user-defined		
	Metrics	user-defined		

Comparison of the Quality Models structure, continued

Quality model	**Types of elements of the model**	**No of levels**	**Number of elements on each level**	**Metrics set**
ISO/IEC **9126**, ISO/IEC **25000**	Characterictics	1	6	yes not fixed
	Sub-characterictics	1	27	
	Quality attributes	user-defined	user-defined	
	Metrics	1	user-defined there is a set of standard metrics	

Table 2.5: Comparison of the Quality Models build methods

Quality model	**Types of relation between elements**	**Build method**	**Method exists?**	**External / Internal elements?**	**levels composition allowed?**	**Evaluation objects**
McCall	hierarchical	fixed	-	-	X	criterias
Boehm	hierarchical, dependency between metrics	fixed	X	-	X	primitive characteristics
FURPS	hierarchical	fixed	-	-	-	low-level quality attributes
GQM	hierarchical	flexible	X	-	X on low-level elements	questions
Gilb	hierarchical	flexible	-	-	X	high- and low-level attributes
Dromey	linkages between components and high-level quality attributes	flexible	X	X	X	-
SQUID	hierarchical, linkages between internal and external attributes	flexible	-	X	X	quality attributes
IEEE 1061	hierarchical, conflicting and supporting relations between factors	flexible	X	-	X	elements of any level
ISO/IEC **9126**, ISO/IEC **25000**	hierarchical	mixed	X	X	X on low-level elements	quality attributes

3 Software quality standard ISO/IEC 25000 SQuaRE series

3.1 Software quality standard structure SQuaRE

The most mature and actual standards of quality are the standards ISO/IEC 25nnn: Software Product Quality Requirements and Evaluation (SQuaRE) which have superseded and replaced standards ISO/IEC 9216 series and ISO/IEC 14598 series. The standards of the family are grouped into divisions - the architecture is shown in the figure 3.1.

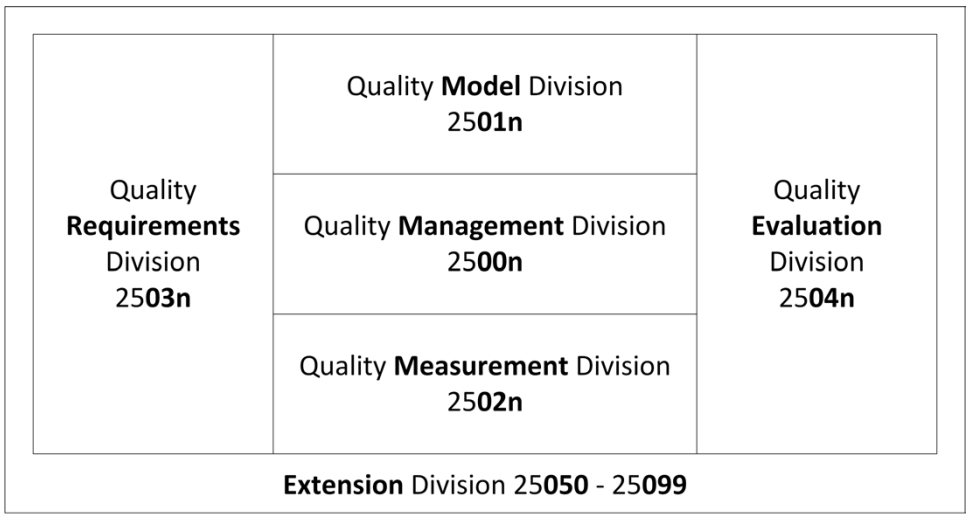

Figure 3.1: Organisation of SQuaRE series of International Standards (from [Sta14])

ISO/IEC 25**00n** - define all common models, terms and definitions. The division also provides requirements and guidance for a supporting function that is responsible for the management of the requirements, specification and evaluation of software product quality.

ISO/IEC 25**01n** - present detailed quality models for computer systems and software products, quality in use, and data.

ISO/IEC 25**02n** - include a software product quality measurement reference model, mathematical definitions of quality measures, and practical guidance for their application.

ISO/IEC 25**03n** - help specify quality requirements, based on quality models and quality measures. These quality requirements can be used in the process of quality requirements elicitation for a software product to be developed or as input for an evaluation process.

ISO/IEC 25**04n** - provide requirements, recommendations and guidelines for software product evaluation, whether performed by evaluators, acquirers or developers.

ISO/IEC 25**050** - 25**099** - currently include requirements for quality of Commercial Off-The-Shelf software and Common Industry Formats for usability reports.

The process of standardisation of software quality concerns has already own history and the standard are constantly evolving taking the recents developments, fixed practices and scientific achievements into consideration. The main reasons of developing and as the consequence the main benefits the SQueRE standard series are: the coordination of guidance on systems and software product quality measurement and evaluation; guidance for the specification of system and software product quality requirements; harmonisation with ISO/IEC/IEEE15939 in the form of Software product Quality Measurement Reference Model (figure 3.3).

Standard ISO/IEC 25010 [Sta11a] defines hierarchical structure for quality models (see figure 3.2). Top level defines the characteristics which in turn can be divided into a set of subcharacteristics. Measurable properties of software related to a quality are called quality properties. Quality metrics in turn are linked with appropriate quality properties. In order to measure the quality of some characteristic of subcharacteristic (if it is not possible to accomplish by direct measurement or this characteristic of subcharacteristic) it is necessary to define properties related to this characteristic of subcharacteristic, calculate the values of all metrics of these properties and unite the measurement results info a derivate measure of this characteristic of subcharacteristic.

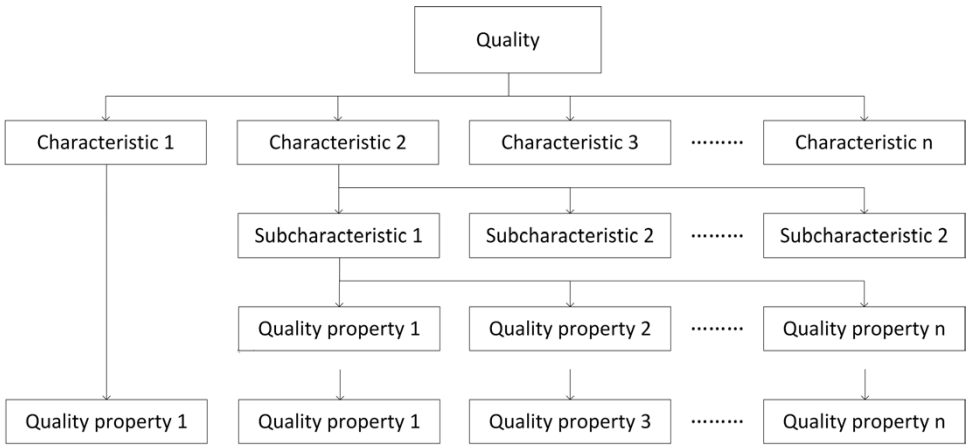

Figure 3.2: ISO25010 - Structure of the quality models (from [Sta11a])

Standard ISO/IEC 25020 introduces the conception of quality measure elements which are used as incoming arguments for measurement functions. In the figure 3.3 are shown the relationships between Quality measure elements, Quality measures and quality char-

acteristics and subcharacteristics. Quality measures are constructed by by applying quality measure elements to a measurement function, they can be either basic or derived measures. Quality measure elements are constructed in accordance with the guidance from ISO/IEC/IEEE 15939 [Sta19], [Sta17].

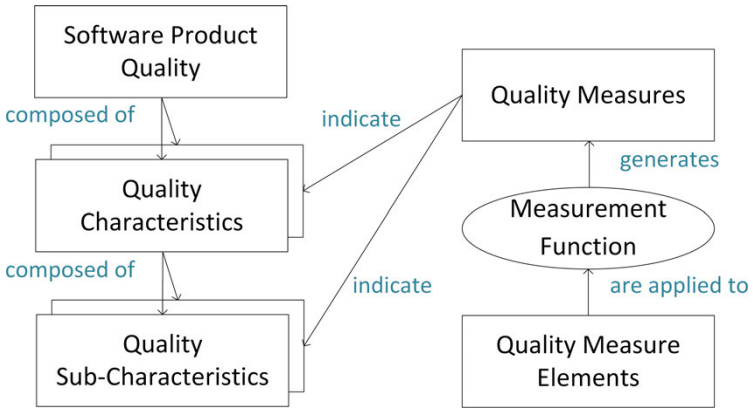

Figure 3.3: ISO25020 - Software product quality measurement reference model (from [Sta19])

Standard defines the System/Software Quality Life Cycle Model (see figure 3.4). It is used to address and differentiate between three principal phases of software lifecycle:
- software being developed undergoes internal measures of software quality;
- software in testing phase undergoes external measure of software quality;
- software in use phase is the subject of quality in use.

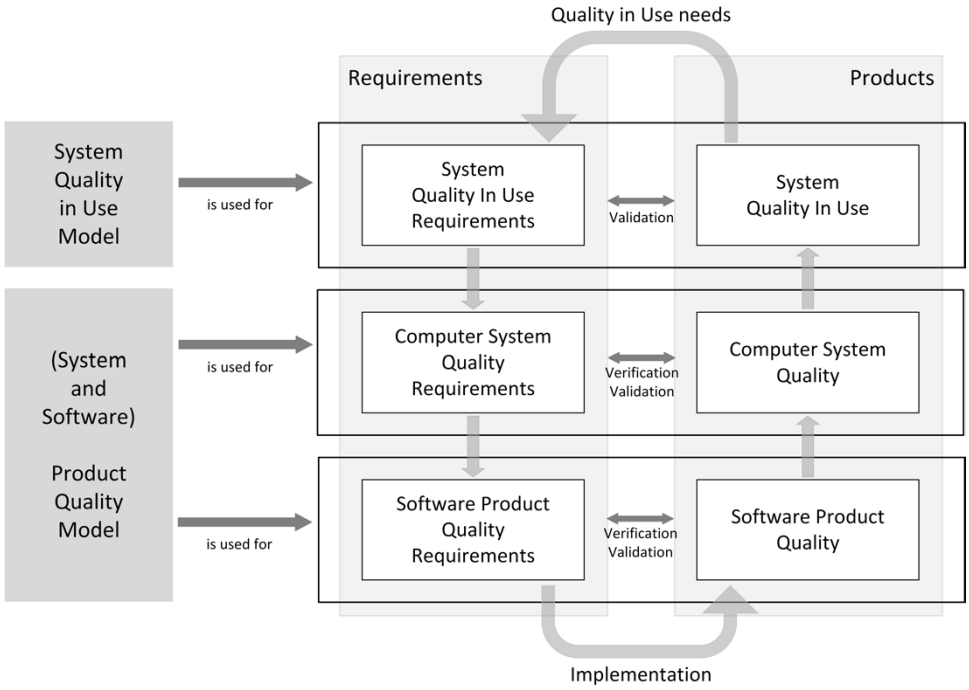

Figure 3.4: ISO25010 - System / Software Quality Life Cycle Model (from [Sta11a])

The quality of software product is therefore treated by standards on three phases of lifecycle: development phase is defined by internal quality, testing and technical verification and validation phase is defined by external quality and use of software is defined by quality in use. The relationships between these different quality aspects are shown in the figure 3.5.

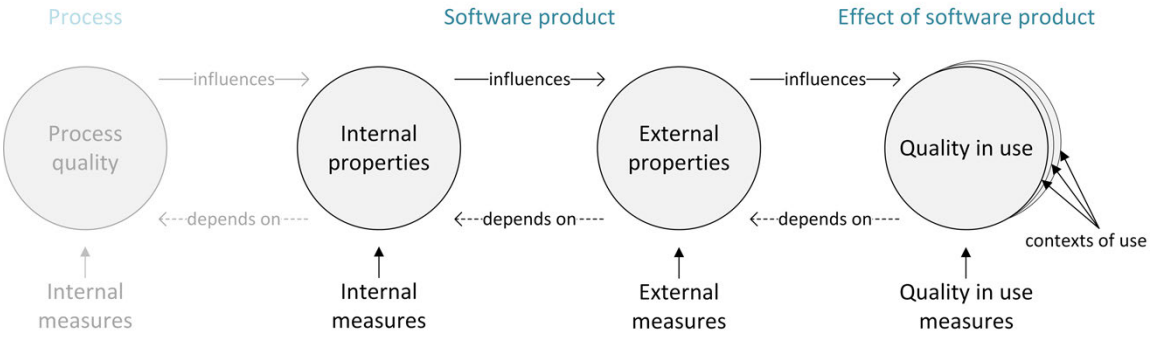

Figure 3.5: ISO25010 - Quality in the lifecycle (from [Sta11a])

Internal quality is a set of software characteristic from internal point of view, which is measured by internal measures and is evaluated against the requirements to the internal quality. The improvement of some quality elements can be archived by coding,

verification or testing, but the fundamental basis of internal quality can be changed only by new design and development.

External quality is a set of software characteristic from external point of view, which is measured by external measures while executing the software product in test modelled environment with modelled data or while ist real usage. Estimated or forecasted external quality - this is estimated or predicted quality of the final software product on each phase of the development process for each quality characteristic or subcharacteristic which is based on the knowledge of the internal quality.

Quality in use is the quality of software product used in a given environment in a given context of use from the user point of view. The quality is evaluated by using quality of use measures and in the fist place the degree to which the user achieves his goals in this environment is measured. The user evaluates the attributes of the software which are related to and are applicable in his tasks and not the properties of the software itself. Estimated or forecasted quality in use - this is estimated or predicted quality of the final software product on each phase of the development process for each quality characteristic or subcharacteristic which is based on the knowledge of internal and external quality.

The reciprocal interaction and consistent integration and application of quality models allows to estimate, evaluate, control and assure the required quality level during the whole lifecycle of the software product - see figure 3.6

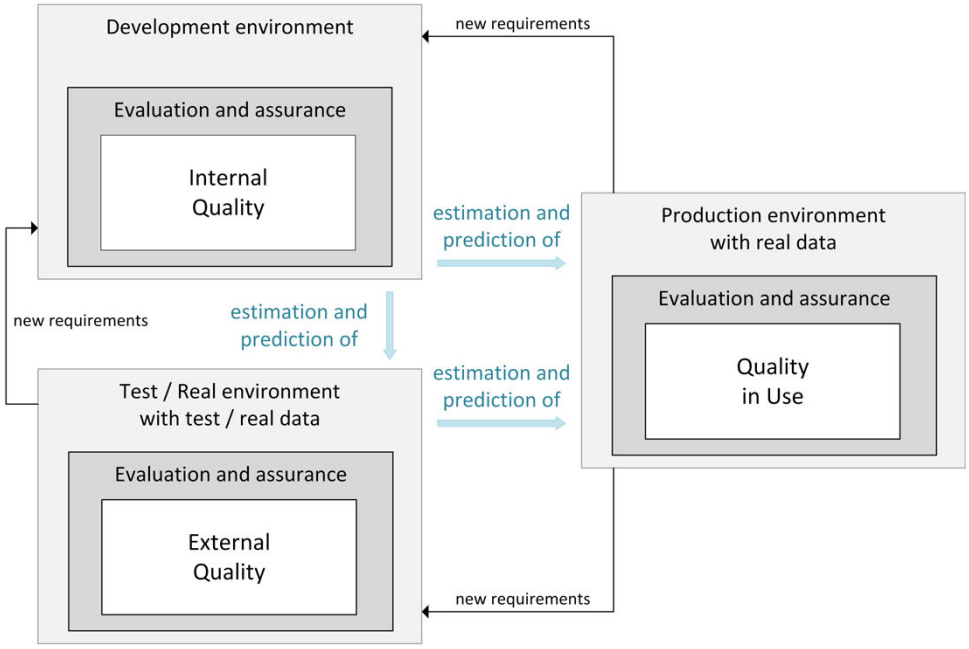

Figure 3.6: Quality prediction in the lifecycle

3.1.1 External / Internal Quality Model

Standard ISO/IEC 25010 [Sta11a] defines the hierarchical internal / external quality model containing eight characteristics which is shown in the figure 3.7.

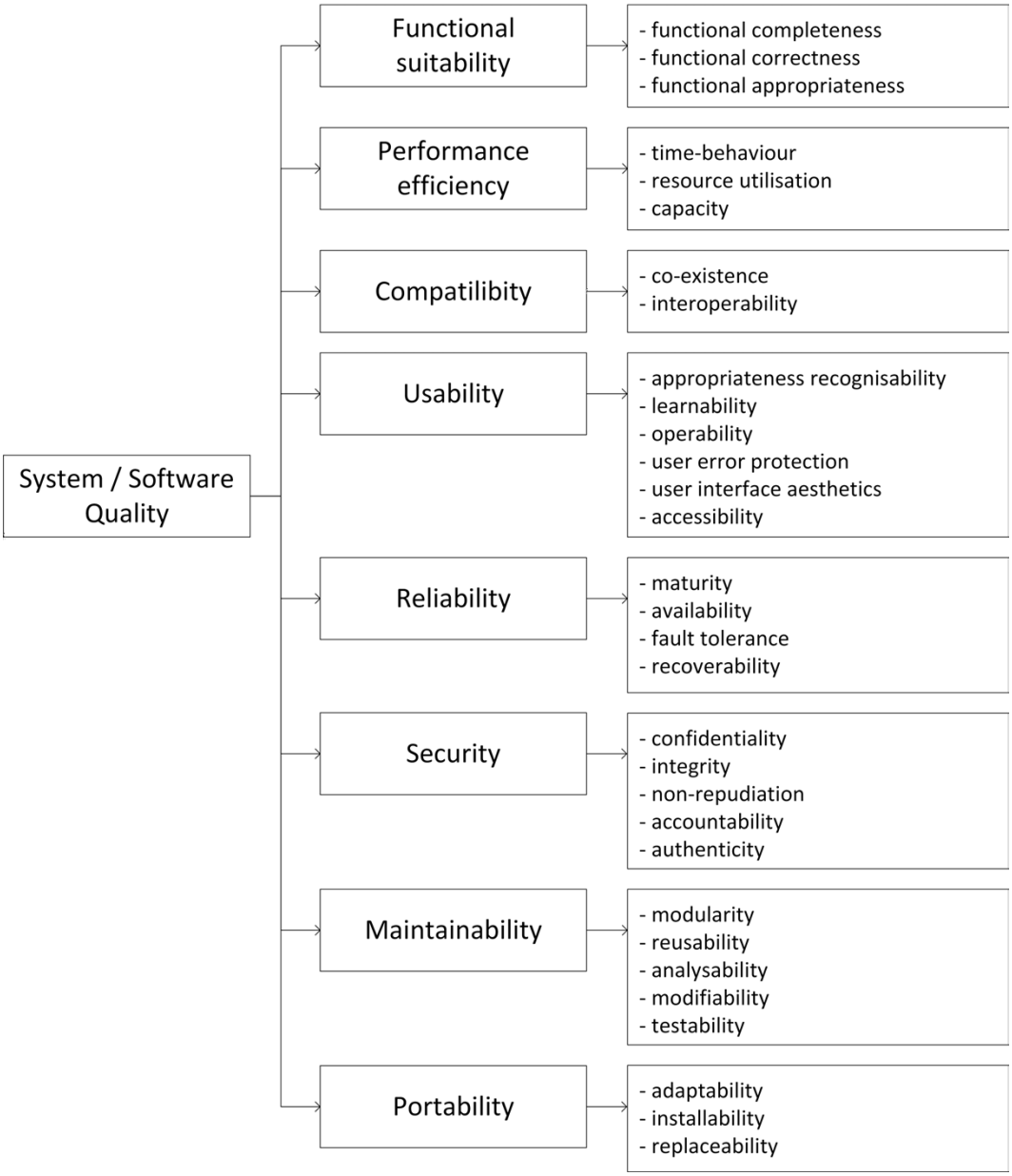

Figure 3.7: ISO25010 - System/Software External / Internal Quality Model (from [Sta11a])

In the table 3.1 the characteristics of the model with their definitions are listed.

Table 3.1: ISO 25010 - external and internal quality characteristics

Functional suitability	degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions
Performance efficiency	performance relative to the amount of resources used under stated conditions
Compatibility	degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment
Usability	degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use
Reliability	degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.
Security	degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization
Maintainability	degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers
Portability	degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another

3.1.2 Quality in Use Model

Standard ISO/IEC 25010 [Sta11a] defines the hierarchical quality in use model containing five characteristics which is shown in the figure 3.8.

Figure 3.8: ISO25010 - Quality in Use Model (from [Sta11a])

In the table 3.2 are listed the characteristics of the model with their definitions and examples of primary user needs interacting with the software.

Table 3.2: ISO 25010 - quality in use characteristics

Effectiveness	accuracy and completeness with which users achieve specified goals *how effective does the user need to be when using the system to perform their task?*
Efficiency	resources expended in relation to the accuracy and completeness with which users achieve goals *how efficient does the user need to be when using the system to perform their task?*
Satisfaction	degree to which user needs are satisfied when a product or system is used in a specified context of use *how satisfied does the user need to be when using the system to perform their task?*
Freedom from risk	degree to which a product or system mitigates the potential risk to economic status, human life, health, or the environment *how risk free does using the system to perform their task need to be for the user?*
Context coverage	degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in both specified contexts of use and in contexts beyond those initially explicitly identified *to what extent does the system need to be effective, efficient, risk free and satisfying in all the intended and potential contexts of use?*

3.2 Software qualimetry basics

In the software evaluation a lot of used measures are quantitative ones unlike the qualitative measures which mainly were used earlier. This leads to the higher degree of objectivity, accuracy and enables the automation of the evaluation process. Measurements form the basis for understanding of development processes and of software itself, which are used for evaluations of qualitative properties of the software and the processes of its development. In works [AA02], [AA05] is proposed the generalised model of measurement, slightly modified version of which together with the software product quality evaluation general reference model from standard ISO/IEC 25040 [Sta11b] are shown in the figure 3.9.

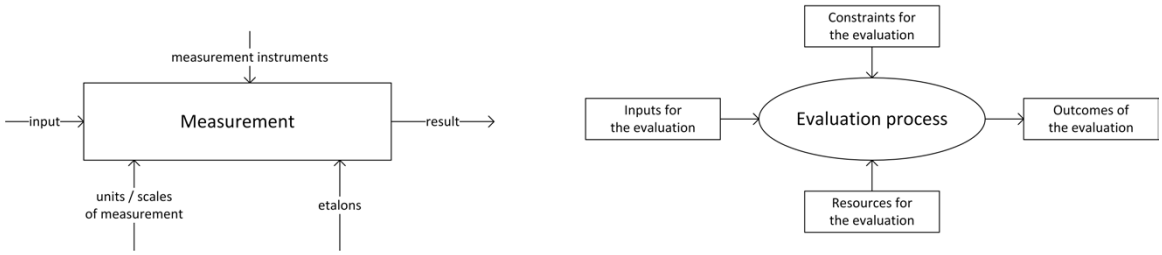

Figure 3.9: ISO25040 - Generalised measurement process and software product quality evaluation general reference model (from [Sta11b])

In the beginning 1970 the scientific discipline qualimetry started to form [AG15], which includes the measurement theories and is itself part of scientific discipline about quality - qualitology. Qualimetry which acts as an interconnected system of theories, differing in the degree of generality, means and methods of measurement and evaluation.

The main terminology and definitions from standards SQuaRE related to qualimetry and to the software quality evaluations are listed in table 3.3.

Table 3.3: SQuaRE - Terminology related to the theory of measurement

Terminology	**Terminology semantic**
measure (noun)	variable to which a value is assigned as the result of measurement. The term "measures" is used to refer collectively to base measures, derived measures, and indicators. To harmonise the terminology with the standard ISO/IEC 15939 the term "measure" and not "metric" is used - equivalent term from the standard ISO/IEC 9126.
base measure	measure defined in terms of an attribute and the method for quantifying it. A base measure is functionally independent of other measures.
derived measure	measure that is defined as a function of two or more values of base measures. A transformation of a base measure using a mathematical function can also be considered as a derived measure.
external measure	measure of the degree to which a software product enables the behaviour of a system to satisfy stated and implied needs for the system including the software to be used under specified conditions. Attributes of the behaviour can be verified and/or validated by executing the software product during testing and operation. [Sta11a]
internal measure	measure of the degree to which a set of static attributes of a software product satisfies stated and implied needs for the software product to be used under specified conditions. Static attributes include those that relate to the software architecture, structure and its components. Static attributes can be verified by review, inspection, simulation and/or automated tools. [Sta11a]

quality in use measure	measure of the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk, satisfaction and context coverage in specific contexts of use [Sta12]
evaluation method	procedure describing actions to be performed by the evaluator in order to obtain results for the specified measurement applied to the specified product components or on the product as a whole
evaluation module	package of evaluation technology for measuring software quality characteristics, subcharacteristics or attributes. The package includes evaluation methods and techniques, inputs to be evaluated, data to be measured and collected and supporting procedures and tools.
measurement	set of operations having the object of determining a value of a measure. Measurement can include assigning a qualitative category.
measurement function	algorithm or calculation performed to combine two or more base measures
measurement method	logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale
measurement procedure	set of operations, described specifically, used in the performance of a particular measurement according to a given method
measurement process	process for establishing, planning, performing and evaluating software measurement within an overall project or organisational measurement structure
rating	action of mapping the measured value to the appropriate rating level. Used to determine the rating level associated with the software product for a specific quality characteristic
scale	ordered set of values, continuous or discrete, or a set of categories to which the attribute is mapped
rating level	scale point on an ordinal scale, which is used to categorise a measurement scale. The rating level enables software product to be classified (rated) in accordance with the stated or implied needs. Appropriate rating levels may be associated with the different views of quality i.e. Users', Managers' or Developers'.
software product evaluation	technical operation that consists of producing an assessment of one or more characteristics of a software product according to a specified procedure

A scale of measurement is a conventionally accepted procedure for determining and denoting values of a certain quantity, which is a set of values onto which the measured quality attribute is mapped. Using a certain measurement method the value of the measured attribute is mapped on certain value of a scale. In the figure 3.10 is shown a sample of a rate level from the standard ISO/IEC 14598-1 [Sta99].

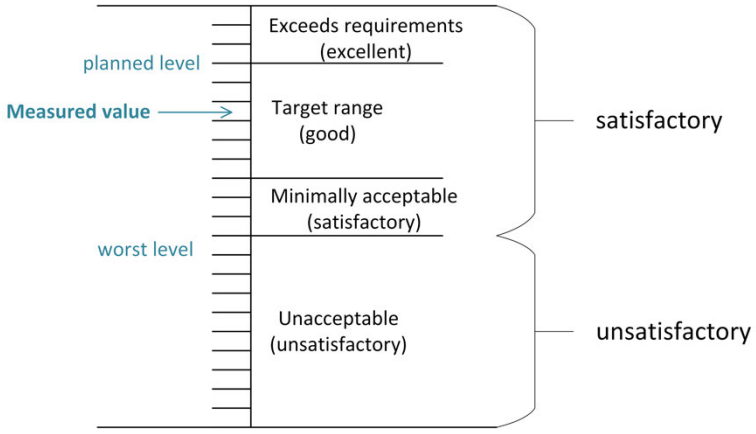

Figure 3.10: ISO/IEC 14598-1 - Possible ranking of measured values of quality measures (from [Sta99])

There are several approaches to the definition and properties of measurement scale, but in general it is possible to differentiate between eight scales of measurement: nominal, ordinal, interval, power, logarithmic, difference, ratio und absolute.

Nominal scale represents the qualitative properties; its elements are characterised by the equivalence and similarity ratios of specific qualitative manifestations of the property. This scale does not have zero and units of measurement, no comparisons as less than, more than are possible, no arithmetic operations are possible, can possibly arise the uncertainty in the measurement result. The measurement itself is reduced to comparing the measured object with the reference one (etalon) and choosing one (or two neighbouring ones) that coincides with the measured one.

Ordinal scale allows comparisons of one size with another on the basis of which is larger or which is better. This scale can be thought of as an extension of the nominal scale by introducing partial order relations. The operation of sizing in ascending or descending order to obtain measurement information is called ranking. On an ordinal scale, sizes are compared with each other, which at the same time remain unknown. The comparison operation results in a ranked series. The mathematical model for comparing two sizes of one measure on a scale of order is the inequality $M_j \leq M_i (M_j \geq Mi)$, and the result of the comparison is the conclusion about which size is larger than the other or they are equal to each other. Such measurements are not informative, since they do not give an answer to the question of how much or how many times one size is larger than the other. On the scale of order, only some logical operations can be performed - for example, if the first dimension is greater than the second, and the second is greater than the third, then the first is greater than the third. If two sizes are less than the third, then their difference is less than the third. Such properties of the scale are called transitivity properties. At the same time, arithmetic operations are not defined on it.

Figure 3.11: Ordinal scale

Interval scale shows the difference between the dimensions. Figure 3.12 shows the principle of constructing an interval scale for dimensions that form a ranged series $M_1 < M_2 < M_3 < M_4 < M_5 < M_6 < M_7$. The mathematical model for comparing two sizes of one measure with each other is the expression $M_i - M_j = \Delta M_{ij}$. By constructing a scale of intervals all sizes M_i are compared with a size M_j. Figure 3.12 shows an interval scale with M_4 selected as M_j. Choosing M_5 as M_j would shift zero to the right, and M_3 to the left. The reference point on the interval scale can be chosen arbitrary. The interval scale defines the mathematical operations of addition and subtraction. The intervals can be added to each other and subtracted from each other, taking into account the signs, and therefore it is possible to determine how much one size is larger or smaller than the other: $M_7 - M_5 = \Delta M_7 - \Delta M_5$, $M_5 - M_2 = \Delta M_5 - (-\Delta M_2) = \Delta M_5 + \Delta M_2$, $M_3 - M_1 = -\Delta M_3 - (-\Delta M_1) = \Delta M_1 - \Delta M_3$. Additive operations in these expressions are performed with the sizes of the intervals obtained according to the formula $M_i - M_j = \Delta M_{ij}$, that is, on an ungraded scale. In the case of a graduated scale, the sizes of units are expressed in specific units of measurement and the expressions of equality given continue to apply. Due to the uncertainty of the reference point, multiplicative operations on the interval scale are not defined.

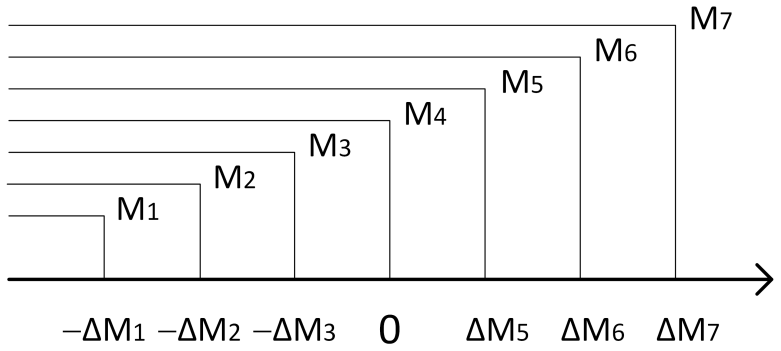

Figure 3.12: Interval scale

Ratio scale is obtained from an interval scale by adding the concept of zero to it. On this scale, you can count the absolute value of the size and determine how many times one size is larger or smaller than another. The relative scale is the most informative one, it defines all mathematical operations: addition, subtraction, multiplication and

division. The value of the measured quantity M is determined by its numerical value g and some size $[M]$ taken as a unit of measurement: $M = g[M]$, where M is the measured value; $[M]$ - unit of measurement; g is a numeric value. Increasing or decreasing $[M]$ results in an inversely proportional change in g. Therefore, the value, like the size of the measured value, does not depend on the choice of measurement units.

Power scale differs from the interval in the type of transformations: proportional transformations are used in the power scale whereas in the interval - linear.

Difference scale uses shift transformations: $M = [M] + g$, and *logarithmic scale* uses exponentiation: $M = [M]^g$.

Absolute scale has all the properties of a ratio scale. Its purpose is to reflect the proportional relationship between values measured in the same units. Units of absolute scales are natural, not chosen by convention, but these units are dimensionless (times, percentages, fractions, etc.). The absolute scale can be limited and unlimited, it is not linear and has no units of measurement.

The figure 3.13 shows the relation between different scale types, ordered by the degree of information they contain.

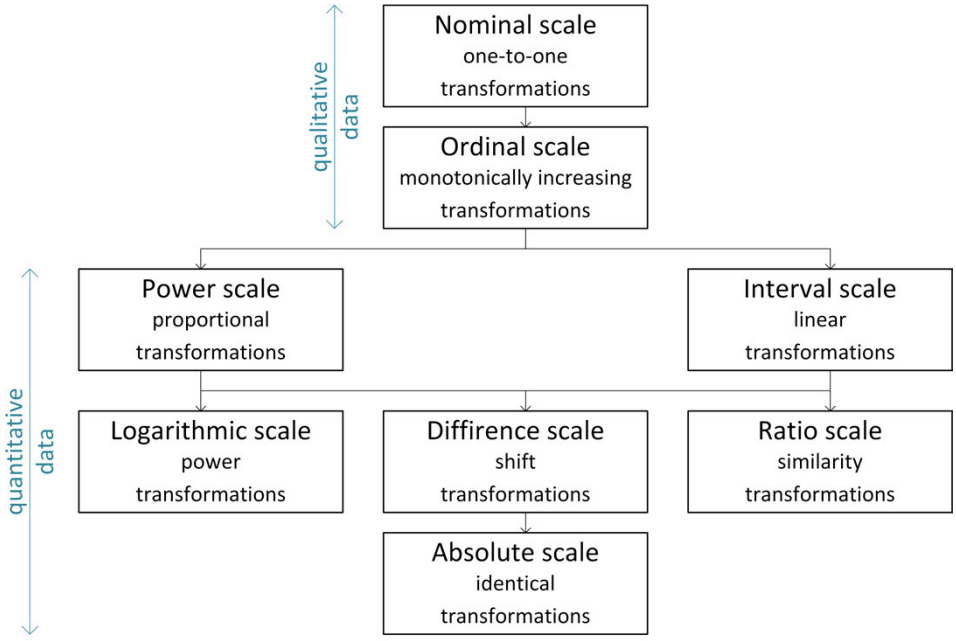

Figure 3.13: Scale types relation

To ensure the possibility of joint use of various measures by quantitative assessment of the software quality, regardless of their physical meaning, units of measurement and ranges of values, the measures can be presented in relative units in the form:

$$X = \frac{A}{B} \quad \text{or} \quad X = 1 - \frac{A}{B} \tag{3.1}$$

where X - the value of the measure;
A - the absolute (measured) value of some property (attribute) of the evaluated object;
B - the base value of the corresponding property (attribute);

For a specific measure one equation from 3.1, which meets the criteria for tracing and consistency, should be selected: with an increase of the relative value of the measure the value of the subcharacteristic and characteristic of the quality should also increase.

Calculation of measures according to equation 3.1 allows bringing their relative values into the range:

$$0 \leq X \leq 1 \tag{3.2}$$

which simplifies their joint use in the integral assessment of the quality of software.

3.3 Extension and customisation of the quality model and measures

The SQuaRE standards define the nominal quality model with the open possibility for further extensions and modifications.

The decomposition of software product quality may go through several levels although the nominal model contained in ISO/IEC 25010 [Sta11a] only contains two levels [Sta19]. So the top levels are fixed but it is possible to perform further decomposition if necessary. In addition to that the selection and usage of measures is completely free provided the essential constraints are met. It is allowed modify the measures or use measures which are not included in SQuaRE standards. [Sta19] When using a modified of new measure not identified in the ISO/IEC 25022 [Sta16a], ISO/IEC 25023 [Sta16b]or ISO/IEC 25024 [Sta15b], it is necessary to specify how the measure relates to its corresponding quality model and how it is to be constructed from quality measure elements.

3.3.1 Properties and criteria for the validity of quality measures

The standards ISO/IEC 25022 [Sta16a], ISO/IEC 25023 [Sta16b], ISO/IEC 25024 [Sta15b] define respectively internal, external measures and quality in use measures. Additionally, ***desirable properties*** for measures are defined [Sta19].

- *Reliability* (of measure): is associated with random error. A measure is free of random error if random variations do not affect the results of the measure.
- *Repeatability* (of measure): repeated use of the measure for the same product using the same evaluation specification (including the same environment), type of users, and environment by the same evaluators, should produce the same results within appropriate tolerances. The appropriate tolerances should include such things as fatigue, and learning effect.
- *Reproducibility* (of measure): use of the measure for the same product using the same evaluation specification (including the same environment), type of users,

and environment by different evaluators, should produce the same results within appropriate tolerances.

It is recommended to use statistical analysis to measure the variability of the results.

- *Availability* (of measure): the measure should clearly indicate the conditions (e.g. presence of specific attributes) which constrain its usage.
- *Indicativeness* (of measure): capability of the measure to identify parts or items of the software which should be improved, given the measured results compared to the expected ones.
- *Correctness* (of measure): the measure should have the following properties:
  1. *Objectivity* (of measure): the measure results and its data input should be factual: i.e., not influenced by the feelings or the opinions of the evaluator, test users, etc. (except for satisfaction or attractiveness measures where user feelings and opinions are being measured).
  2. *Impartiality* (of measure): the measurement should not be biased towards any particular result.
  3. *Sufficient precision* (of measure): Precision is determined by the design of the measure, and particularly by the choice of the material definition used as the basis for the measure. The measure user will describe the precision and the sensitivity of the measure.
- *Meaningfulness* (of measure): the measurement should produce meaningful results about the software behaviour or quality characteristics.

The developer of the measure must proof its validity. The measure must meet at least one of the following ***criteria for validity*** [Sta16b]:

- *Correlation*: the variation in the quality characteristics values (the results of principal measures in operational use) explained by the variation in the measure values, is given by the square of the linear coefficient.

  It is possible to predict quality characteristics without measuring them directly by using correlated measures.
- *Tracking*: if a measure M is directly related to a quality characteristic value Q (the results of principal measures in operational use), for a given product or process, then a change value $Q(T_1)$ at the moment of time T_1 to $Q(T_2)$ at the moment of time T_2, would be accompanied by a change measure value from $M(T_1)$ to $M(T_2)$, in the same direction (for instance, if Q increases, M increases).

  It is possible to detect the movement of quality characteristic along a time period without measuring directly by using those measures which have tracking ability.
- *Consistency*: if quality characteristics values (the results of principal measures in operational use) $Q_1, Q_2, ..., Q_n$, corresponding to products or processes $1, 2, ..., n$, have the relationship $Q_1 > Q_2 > ... > Q_n$, then the corresponding measure values would have the relationship $M_1 > M_2 > ... > M_n$.

  It is possible to notice exceptional and error prone components of software by using those measure which are capable of being consistent.

- *Predictability*: If a measure is used at time T_1 to predict a quality characteristic value Q (the results of principal measures in operational use) at time T_2, prediction error, which is $\frac{predictedQ(T_2)-actualQ(T_2)}{actualQ(T_2)}$, would be within the allowed prediction error range.

  It is possible to predict the movement of quality characteristics in the future by using those measures which are within the allowed prediction error range.
- *Discrimination*: a measure should be able to discriminate between high and low quality for software characteristics and subcharacteristics.

  It is possible to categorise software components and rate quality characteristics values by using those measures which have the capability to discriminate between high and low quality.

4 Enterprise applications quality assurance

4.1 Enterprise applications definition

In the last decades globalisation and total digitalisation set themselves as the main influencing trends worldwide. Many companies are being forced to face with completely new challenges and tasks, which have never existed before, and must quickly transform themselves to be able to react properly to all these transformations or even should try to become faster and be more predictive in order to set themselves as pioneers and inventors of the products and services of the a nature. Enterprise software today not only supports the companies in their common routine business processes but, more importantly, makes the digital business transformation and digitalisation possible by providing technical advances and appropriate environment for all of these changing processes.

It is not possible to give a precise definition of what the enterprise applications exactly are because this family of software includes a wide range of possible solutions of diverse scale and complexity. For example, wikipedia gives the concise and abstract definition: enterprise software, also known as enterprise application software, is computer software used to satisfy the needs of an organisation rather than individual users [Wik]. What is possible to do is to define common properties, features and a range of functions of typical enterprise software.

First, it is focused on the needs and, therefore, requirements and tasks of large organisations, such as enterprises, rather middle or big companies, governments, different groups. It aims at modelling the company-wide structure with all the inherent complexity, supporting all of the business processes from the very start to the end and creating the uniform and solid enterprise environment.

Second, enterprise software works intensively with a huge amount of persistent transactional data, which must be available for different applications and be stored for a long time; provides solutions for diverse analysis and processing capabilities of the data. Different company divisions might require many advanced analytics and different knowledge extracted from the same array of stored data. The data must be available for concurrent access, not only for reading but also for modifications.

Third, much of the data requires many interfaces to handle it properly, including GUI interfaces for end-users of the software as well as the technical interfaces and protocol definitions for the exchange with external systems. Summing up it can be stated, that enterprise software becomes a central hub of the corporate data.

Last but not least, enterprise applications are often distributed systems where multiple software modules or components are executed on different calculation units in a networks, which in turn can be local network of the company of a worldwide cloud-solution.

To give a more exact understanding of which software solutions can be classified as

enterprise software the following list shows the main types of enterprise applications [Wik]:

- Business Intelligence BI comprises the strategies and technologies used by enterprises for the data analysis of business information.
- Business Process Management BPM is a discipline in operations management in which people use various methods to discover, model, analyse, measure, improve, optimise, and automate business processes.
- Customer Relationship Management CRM is one of many different approaches that allow a company to manage and analyse its own interactions with its past, current and potential customers.
- Content Management System CMS is a computer software used to manage the creation and modification of digital content.
- Database Management System DBMS software system that enables users to define, create, maintain and control access to the database.
  - Master Data Management MDM is a technology-enabled discipline in which business and information technology work together to ensure the uniformity, accuracy, stewardship, semantic consistency and accountability of the enterprise's official shared master data assets.and
  - Data Warehousing DW, DWH, EDW is a system used for reporting and data analysis, and is considered a core component of business intelligence.
- Enterprise Resource Planning ERP is the integrated management of main business processes, often in real time and mediated by software and technology.
- Enterprise Asset Management EAM involves the management of the maintenance of physical assets of an organisation throughout each asset's lifecycle.
- Human Resource Management HRM is the strategic approach to the effective management of people in a company or organisation such that they help their business gain a competitive advantage.
- Knowledge Management KM is the process of creating, sharing, using and managing the knowledge and information of an organisation.
- Low-code Development Platforms LCDP is software that provides a development environment used to create application software through graphical user interfaces and configuration instead of traditional hand-coded computer programming.
- Product Data Management PDM is the business function often within product lifecycle management PLM that is responsible for the management and publication of product data.

- Product Information Management PIM is the process of managing all the information required to market and sell products through distribution channels.
- Product Lifecycle Management PLM is the process of managing the entire lifecycle of a product from inception, through engineering design and manufacture, to service and disposal of manufactured products.
- Supply Chain Management SCM the management of the flow of goods and services, involves the movement and storage of raw materials, of work-in-process inventory, and of finished goods as well as end to end order fulfilment from point of origin to point of consumption. Interconnected, interrelated or interlinked networks, channels and node businesses combine in the provision of products and services required by end customers in a supply chain.
- Software Configuration Management SCM - such as Version Control System VCS - is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management.
- Networking and Information Security
  - Intrusion Detection Prevention IDS - is a device or software application that monitors a network or systems for malicious activity or policy violations.
  - Software Defined Networking SDN - technology is an approach to network management that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring, making it more like cloud computing than traditional network management.
  - Security Information Event Management SIEM - is a subsection within the field of computer security, where software products and services combine security information management and security event management.

There are also additional solutions which do no relate directly to some business area or spread over many other business functions providing common infrastructure services such as Backup software, Billing Management, Accounting software.

4.2 SAP Enterprise software evolution from R/1 to HANA

SAP is the world's largest provider of Enterprise Applications. Founded in 1972 in Waldorf in Deutschland SAP has today more than 100.000 employees worldwide [SE20a]. SAP solution has been evolving already more than 40 years starting with the fist release of SAP R/1 in 1972. It has succeeded for SAP all these year to anticipate, analyse and provide appropriate solutions for the current market situation and demand. The evolution of SAP product family is shown in the figure 4.1.

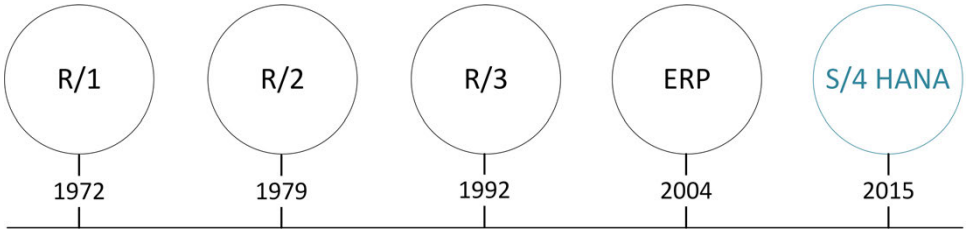

Figure 4.1: SAP product family evolution

With the ERP version in 2004 SAP introduced SAP NetWeaver software stack for the applications and ECC - ERP Central Component - which is the evolutionary successor of SAP R/3.

The latest technological platform of SAP is HANA. SAP HANA was first released in 2010 as in-memory database which has dramatically increased the speed of data processing. Afterwards it has gradually grown to a flexible, data source-agnostic, in-memory data platform with the possibility of large data volume analysis in real time for all solutions offered by SAP Enterprise software. It was called SAP S/4 HANA (SAP Business Suite 4 SAP HANA). It delivers a lot of simplifications as customer adoption, user experience and others as well as innovations such IoT, Big data, predictive analysis etc. The evolution of SAP HANA solutions is shown in the figure 4.2.

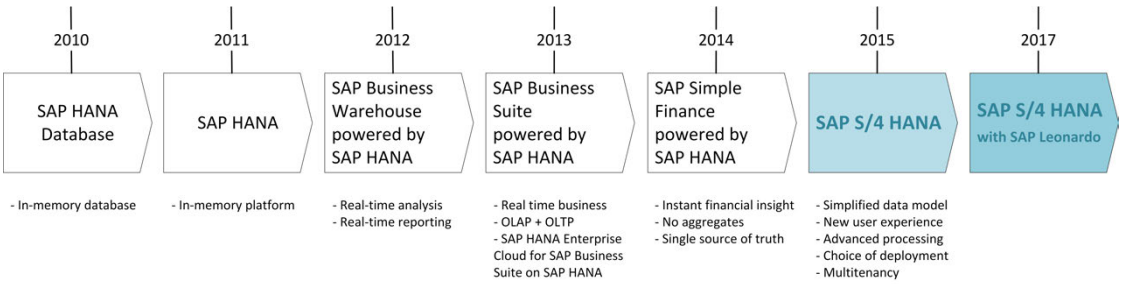

Figure 4.2: SAP HANA - evolution (from [BJ19])

The speed achieved by the usage of technology advances such as multiple Multi-Core CPUs, RAM in the size scale of Terabytes, fast SSD-storage in the size scale of Terabytes enabled new possibilities for real-time data processing and analysis which made possible completely new business applications and enterprise application types and interaction. OLAP - online analytical processing - and OLTP - online transnational processing - became possible in one place with the almost unlimited possibilities of data processing and analysis (see figure 4.3). Moreover there are also different real-time application domains and solutions are provided by SAP depending on the company business and requirements (see table 4.1).

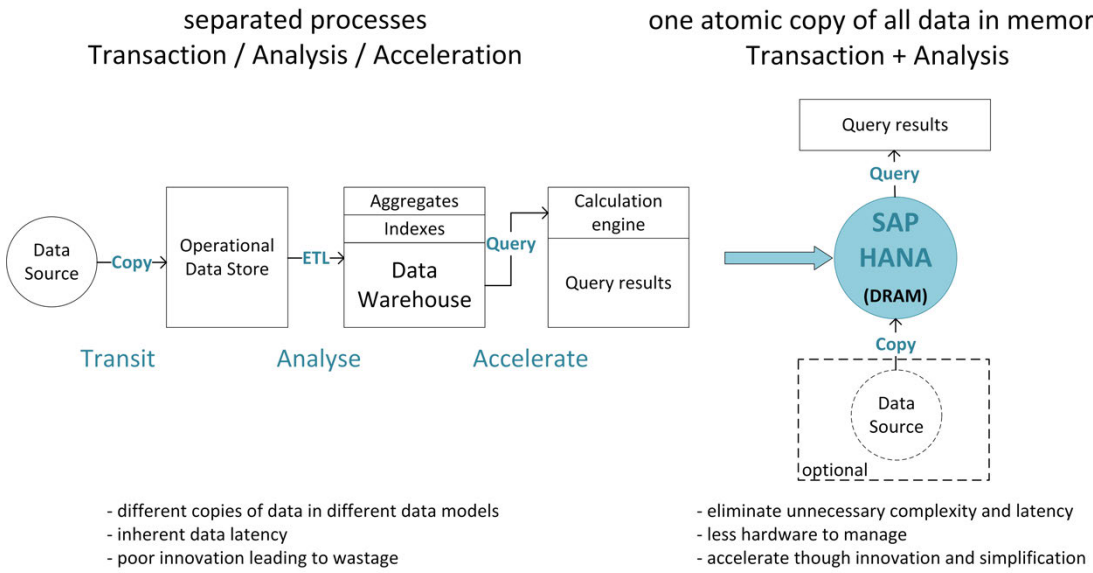

Figure 4.3: SAP HANA - OLTP and OLAP, speed advantage (from [K.K18])

Table 4.1: SAP Real-time domains

Real-Time Analytics	**Real-Time Solutions**	**Real-Time Platform**
Operational reporting	Core Business Acceleration	Database
Data warehousing	Planning and Optimisation	Mobile
Predictive and Text Analysis of large data	Sensing and Response	Cloud

SAP HANA platform is much more than a database and just database services themselves are very advanced and complex in order to allow real-time data transformation, building of aggregates and performing complex calculations - database architecture is shown in the figure 4.4.

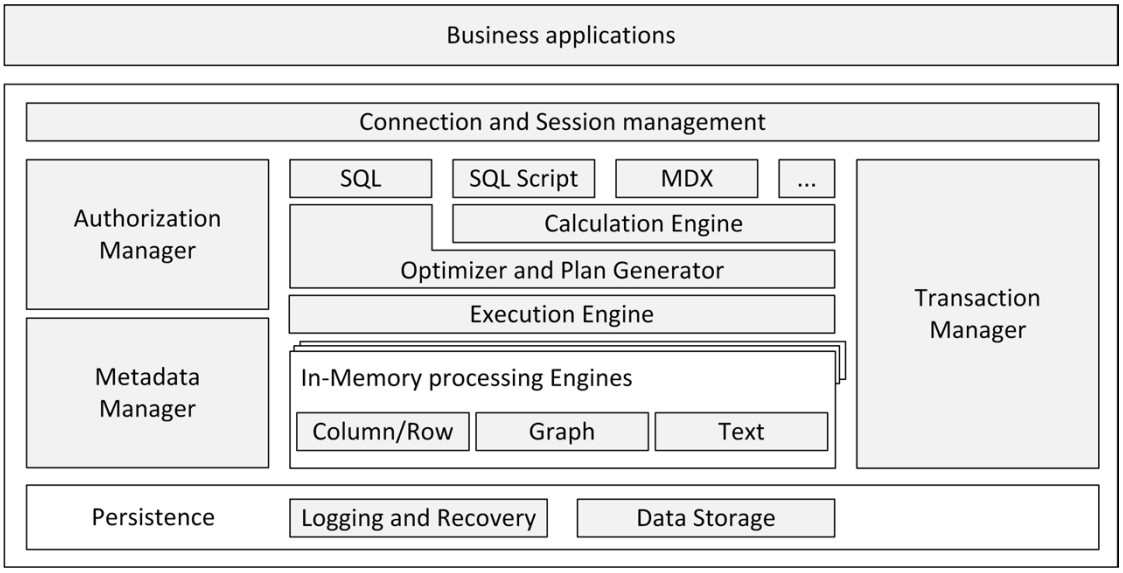

Figure 4.4: SAP HANA DB architecture (from [R.19])

4.3 Focus transformation to user-centred design and technological advances

With the technological advances in SAP HANA the focus was completely reconsidered and shifted towards the end user of the enterprise applications with the completely new way of thinking and transition to the direction of User experience strategy employing design thinking and introducing completely new user-interface and the methods of interaction with the SAP software products. It allowed to put in the first row not the technical expert or developer but a common user from business area who must be able to use the enterprise software to effectively, effortlessly, easily and smart perform the daily tasks to support the business process in the area of his responsibility through its whole lifecycle. The design thinking process is shown in the figure 4.5, it makes it possible to approach and tackle unknown, vague-defined issues by looking at them in human environment (as opposed to technical and engineering) and focusing directly on the humans (user) and their needs and on what is important for them.

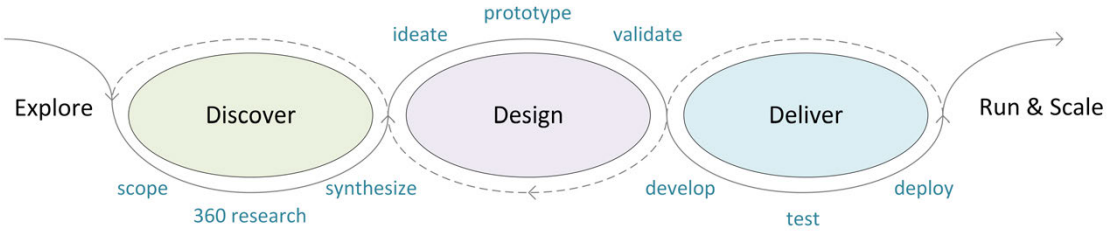

Figure 4.5: Design thinking process (from [SE20b])

4.3.1 SAP Classical GUI

SAP traditionally used proprietary graphical user interface technology called SAP-GUI. Interface is developed in own programming language ABAP, screens in accordance with the technology are called classical screens Dynpros and communicate with SAP-application server using own protocol (see figure 4.6). First it required installation of the local client (normally available for Windows OS) and later SAP has introduced Web Dynpro technology which could be employed to render the classical GUI-screens on the web-interface, what actually is only a transformation of display surface, not the way a user interacts or uses the enterprise software. The classical GUI has evolved during several decades, was created by technicians for technical and expert users and the way of working and interacting with it is sometimes not only inconvenient or illogical but even counterintuitive.

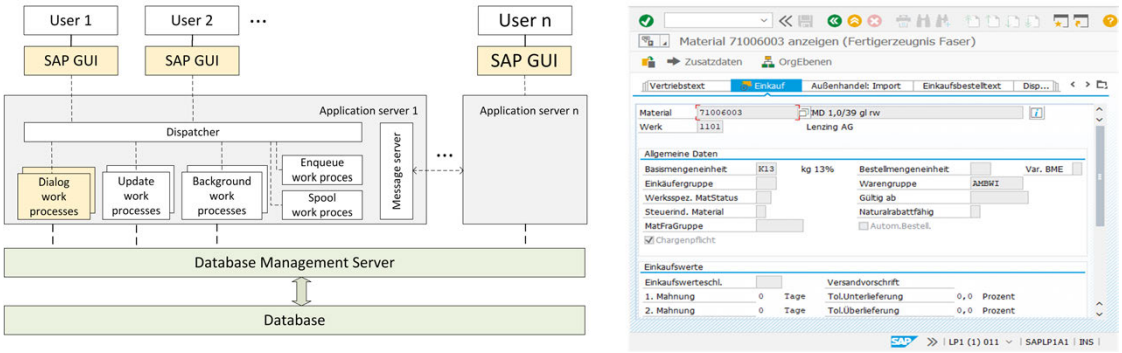

Figure 4.6: SAP Classical-GUI components

4.3.2 SAP Fiori

As a result of complete re-thinking and technological transformation SAP has developed a completely new way of interaction with enterprise applications and a new user interface called **Fiori** [SE20b]. Fiori has renewed the most widely-used scenarios and is aimed at keeping things simple. Frontend part is based of the web-technology using HTML5 and own javascript UI library SAPUI5. Fiori is a place where people, business and technology

meet together (see figure 4.7). This is a SAP's user experience that applies modern user-centric design principles. It provides a consistent and role-specific experience across all tasks, for all lines of business. Fiori is simultaneously simple, can be personalised and runs on any device, helping people to get the job done easily with an intuitive experience.

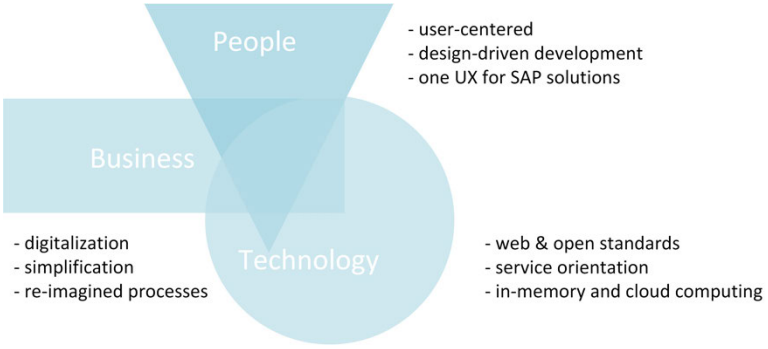

Figure 4.7: SAP Fiori - UX Direction (from [K.K18])

It is not the new collections of enterprise applications, it is the new face of SAP business users and the user experience paradigm which upholds certain values, principles and practices listed in the table 4.2 [SE].

Table 4.2: SAP Fiori Design: values, principles and practices

Values	Consistency	Offer design solutions that can be adopted by all UI technologies and scenarios across the entire range of the SAP portfolio, leveraging the expertise of the entire SAP design community.
	Integration	Provide solutions to integrate different independent products and technologies into an environment that is coherent and easy to use.
	Intelligence	Establish machine learning and artificial intelligence as an integral part of the user experience, with a focus on enabling the user rather than taking away control.
Principles	Role-based	Provide the right information at the right time.
	Adaptive	Enable users to work where they want, on the device of their choice.
	Simple	Help users focus on what is important.
	Coherent	Provide the same intuitive and consistent experience across the enterprise.
	Delightful	Enrich the user's work experience.
Practices	Design-led development	Put user experience at the heart of the product lifecycle.

First version of Fiori was released in 2013, had only 25 most frequent applications. The second version came to the market in 2016 and had already 600 applications and

7022 visually harmonised classic applications. The actual third version 2018 started in 2018. The features and functionality development between versions are shown below:

SAP Fiori 1.0

Figure 4.8: SAP Fiori - Interface 1st generation (from [SE])

- break down monolithic transactions into role-based and task-focused apps
- enable a responsive experience across all devices
- make use of modern and mobile-inspired interaction patterns

SAP Fiori 2.0

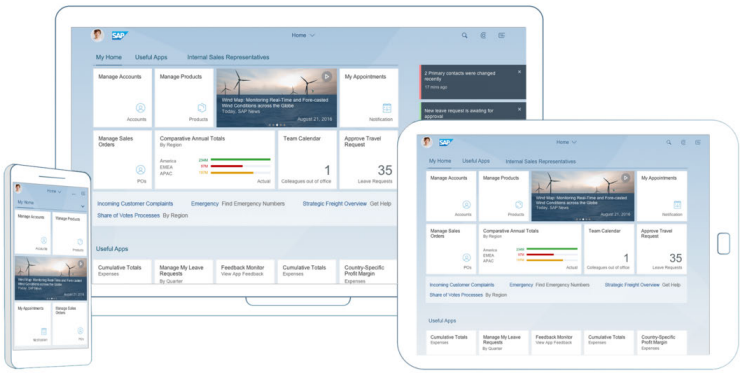

Figure 4.9: SAP Fiori - Interface 2nd generation (from [SE])

- scale the role-based and task-focused design approach to complex enterprise scenarios and native mobile platforms
- offer a powerful productivity environment for business users with new scalable floorplans and patterns, including embedded analytics, notifications, search and improved navigation
- introduce conversational interaction and machine intelligence

SAP Fiori 3

Figure 4.10: SAP Fiori - Interface 3rd generation (from [SE])

- establishes basic rules for consistency across all SAP products
- new Quartz theme
- new aligned shell bar for all SAP products
- greater flexibility for home pages
- integration of machine intelligence, focusing on a human-centric approach
- adoption across a broader range of technologies

4.3.3 SAP Transactions transformation

One important note: in SAP ***transaction*** *means* just a separate ***application*** which is executed in the environment of enterprise software (an application can have a transaction code attached to it; entering this transaction code in special field just starts this application) - it does not mean a technical database transaction or something else (in this sense in SAP other terms / conceptions are used like LUW - logical unit of work).

Classic SAP Transactions were designed mainly from technical and utility point of view and constructed in the way to provide in one application maximal amount of functions, options and settings to cover all possible business-situations and their combinations. This view brought over time a number of serious flaws:

- Many ways to come the same functionality, as within one transaction as well as the same business tasks can be executed with different transactions. This leads to the situations where the same functionality is executed in different contexts with different business data and deliver therefore different results for the same process, what might be confusing and be too technical for the business user;
- Extreme complex user-interface and GUI-screen composition of transactions - many functions are hidden in deep menus, layouts, buttons and in many cases it is counterintuitive and prevents users from focusing on the task;

- Difficult to remember all the necessary transactions / sequence of them, especially for complex, multi-step business processes;
- Some transactions only partially correspond to the business-process and business task the user should execute and provide much more functionality that it is necessary at the moment;
- User often is forced to execute and navigate between multiple transaction to complete singe business task.
- Transactions are not specific to the business role / business tasks, therefore many users from different business areas and with different roles can use the same transaction.
- Learning to effectively work with SAP enterprise applications takes a lot of time and holds the new users from being productive.

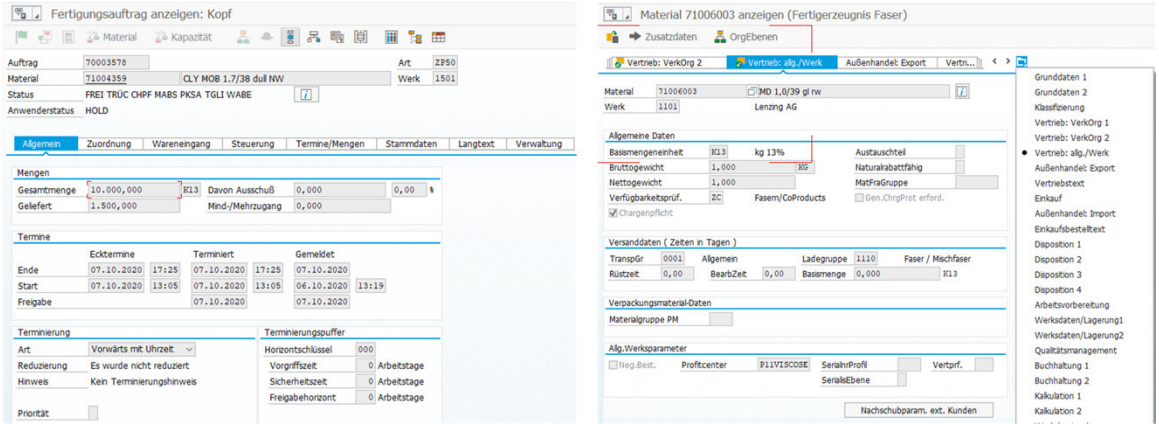

Figure 4.11: SAP Monster transactions exacmple

All such transactions can be named monster or mega transactions, the most functionality of which is complete overload or irrelevant for every single business user which have to use them. From technical point of view one transaction can call (and in most of the cases calls) other transaction(s) at its runtime, which in turn call multiple function modules (globally available in the enterprise application environment function libraries) and / or methods of classes (see figure 4.12).

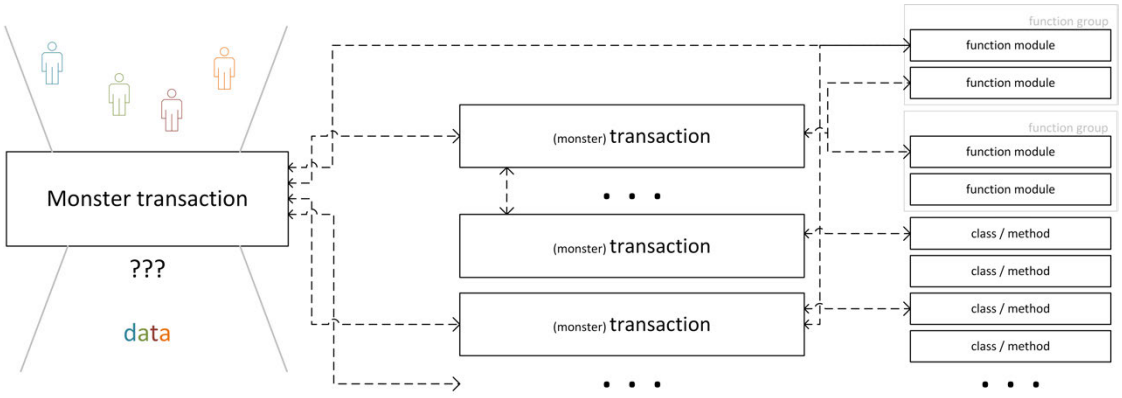

Figure 4.12: SAP Monster transaction composition

The concepts and ideas that lay behind Fiori (see 4.3.2) aimed also at addressing and resolving these problems and anomalies.

4.3.3.1 De-composition of transactions

SAP-Transaction functionality may be broken down into a set of multiple Fiori apps, each of which is aimed at separate user-role and task and presents only what is relevant now in this particular context. This process is called de-composition [K.K18] (figure 4.13).

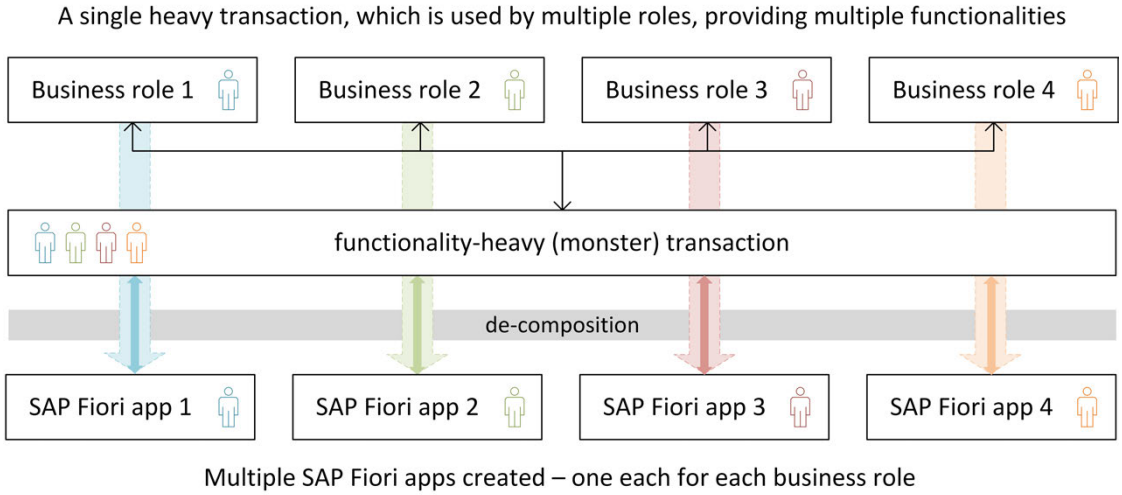

Figure 4.13: SAP de-composition process

4.3.3.2 Re-composition of transactions

Fiori app is developed with the intent of performing one business transaction which combines the related functionality from multiple SAP-Transactions. This process is called re-composition [K.K18] (figure 4.14).

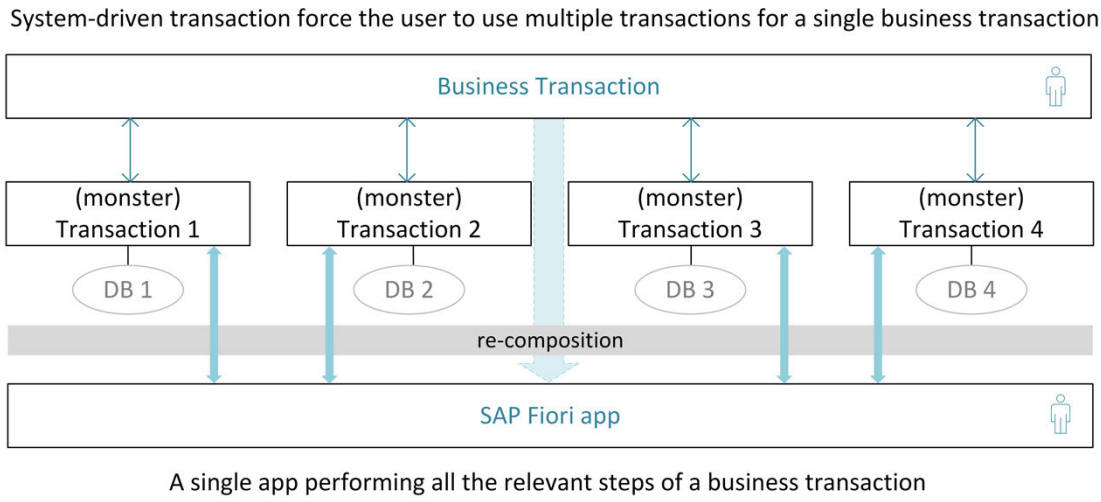

Figure 4.14: SAP re-composition process

4.3.3.3 Role-based applications

The described processes result in a conceptual transformation from technical, function-based transactions (SAP Classic UI) designed from a system perspective to the role-based, business-process oriented, user-experience focused applications (SAP Fiori UX) designed from a user perspective. The purpose is to support a persona (or multiple personas with very similar needs) to complete a task in their use context, design app in such a way to suit specific business environments, specific roles and the way people work. Users and tasks are now the main focus in the enterprise applications which creates a much more efficient and easy system.

4.3.4 Data model simplification and application code push-down

SAP HANA has introduced many new technical advances and it brings many new possibilities and innovations to the application development cycle. One of them is a much simpler data model for the business and database objects. The reason for that is the re-design of the database schema where, thanks to HANA data processing capabilities, a number of data tables and structure became unnecessary. The figure 4.15 [SE19] shows that many additional objects such as aggregates, indexes etc are superfluous with HANA, because all the necessary aggregations and calculations are constructed on the fly in real time from the transactional data.

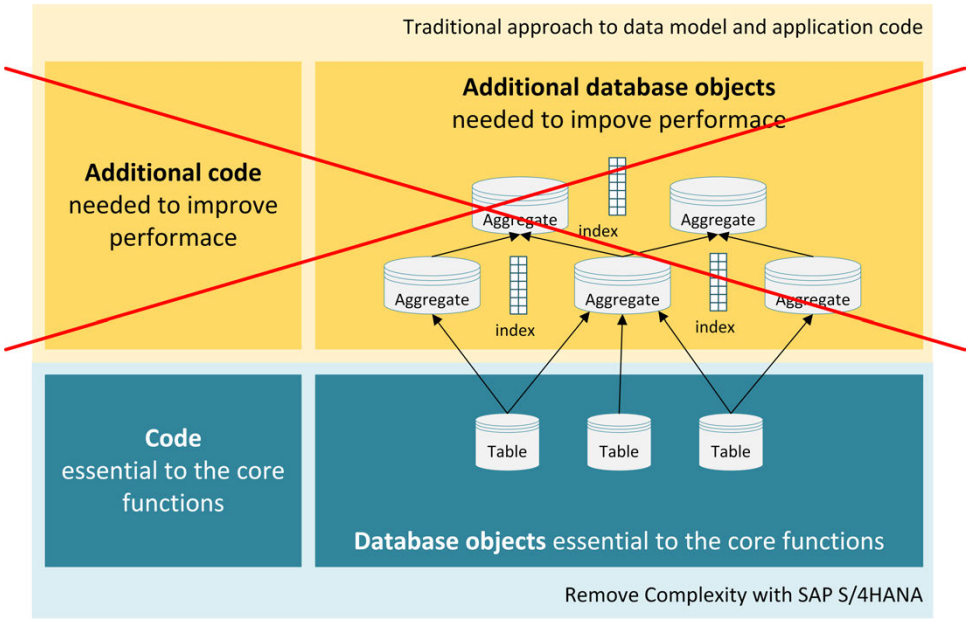

Figure 4.15: Remove complexity with SAP S/4HANA (from [SE19])

As an example in the figure 4.16 the simplification of database-schema and table structure for several SAP-Modules as sale, finance is demonstrated [BJ19].

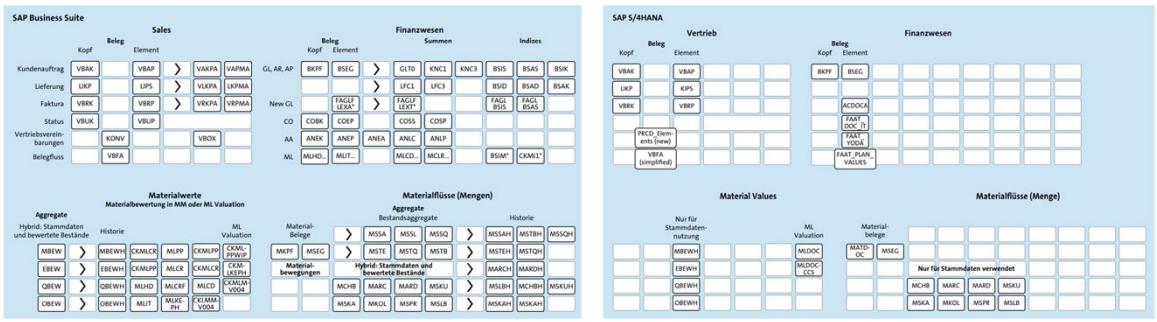

Figure 4.16: Simplification of database-schema and table structure with SAP S/4HANA (from [BJ19])

Additionally SAP HANA platform provides much more solutions and intellectual technologies for data handling, analysis and transformation including multiple application, processing, integration and database services (see figure 4.17).

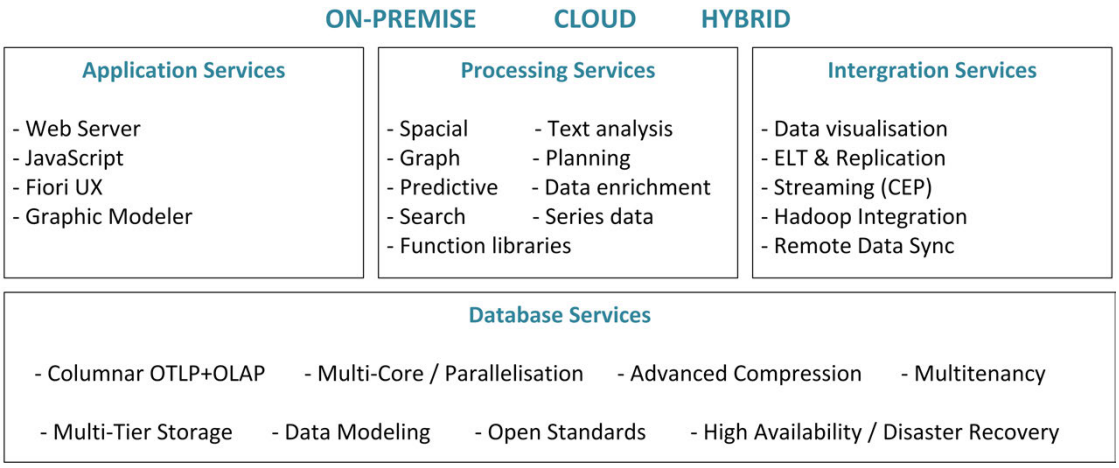

Figure 4.17: SAP HANA - Platform SAP S/4HANA (from [SE19])

All these resulted in the development paradigm shift, where the complex calculations and additional data processing services, which can be prepared and provided by on the database level in real time with SAP HANA, should be done there and application only gets the necessary results, so the calculations are pushed down from application code to the database (see figure 4.18). This approach is called code to data as opposed to the previously used one, data to code, where the transactional or intermediate data was in big volumes returned to the application server and all the calculations were performed by application code by application server computing resources.

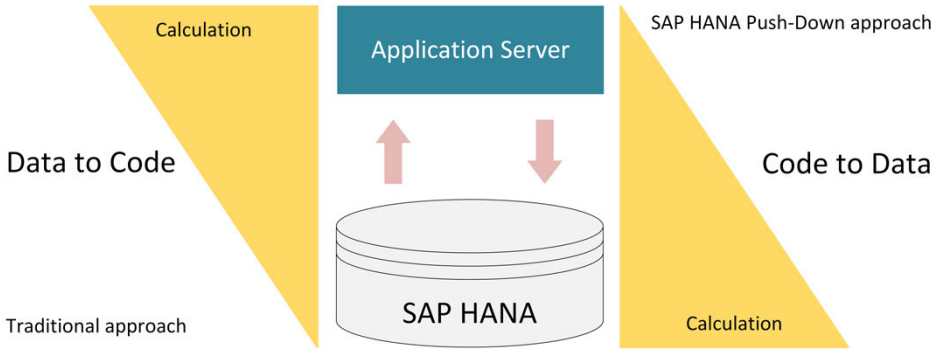

Figure 4.18: SAP HANA Code push-down paradigm (from [SE19])

4.3.5 Hybrid enterprise applications

In this thesis under hybrid enterprise applications are understood the applications which have a clear focus on end-user scenarios and context, have been designed and developed and use the appropriate technologies in such a way to run everywhere, effectively support business roles tasks and be technically efficient, including development, deployment,

operation and maintenance. In SAP the hybrid applications can have the following properties:

- Run on any device (for example Fiori technology, see 4.3.2).
- Can be developed using one technology for all devices (i.e. HTML5) or different technologies may be employed to maintain a set of applications with the same functionality for different device families (i.e. IOS, Android etc)
- MTA (Muti-Target-Application)

In the industry in became a good approach to separate applications into three-tiers: front-end, application (back-end) and a database. The combination of such multiple application and service instances make a full-stack application. SAP with HANA has introduced the concept of MTA - Multi-Target-Application. This is an application which consists of a number of separate applications, each of each in their turn is independent and is responsible for own domain of functionality, and which work together as one solution while the operation. In the MTA project different modules of different types are created (i.e. HTML5, Java, Node.js, SAP HANA DB) and white deployment every module of the project becomes a separate application (for example in Cloud or in on-premise installation infrastructure), each module has its own build-pack, development language and runtime environment.

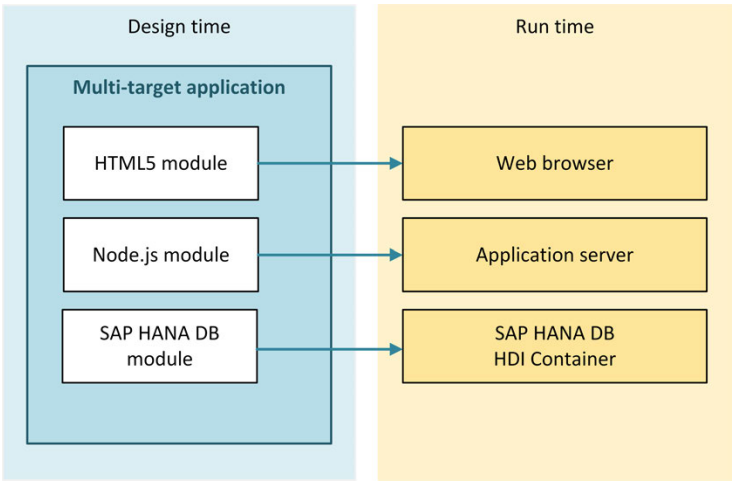

Figure 4.19: SAP HANA Multi-target application (from [SE19])

For example HTML5 modules are served as static files and are executed in Web-browser, Node.js or Java modules are executed on the applications server and SAP HANA DB modules generate appropriate objects in SAP HANA database.

4.4 Analysis of characteristics of hybrid enterprise applications while design, development and usage

Constant availability of business data and vast possibilities for its on-line analysis created, from one side, the enormous demand of having it. From the other side new technological achievements led to invention of completely new delivery and presentation channels of this data and enabled hight-speeds at which it can be transferred. The reciprocal phenomenon arose with the total spread of mobile technologies and powerful devices: because these mobile devices are always at hand and are always connected to the network, the demand for business data should be immediately satisfied at any moment.

Taking these recent trends into account a new paradigm of *design for mobile first* is being employed in design and development processes to assure that application would be smooth, seamless and responsive across platforms. Mobile first approach mainly means:

- look at the whole scenario as the mobile application is being designed fist;
- mobile in this context means that is is necessary to deal with restrictions - build up, do no tear down;
- staying focused on what is important and thinking ahead;
- mobile first might lead to reconsideration of established solutions;
- working with restrictions lead to find new smart ways to reduce, aggregate or group;
- mobile first has responsive or adaptive design in mind from the first moment of designing and development.

In these circumstances it is essential to analyse and consider the characteristics of mobile devices and the context of their usage while designing and developing enterprise applications. The most important features, peculiarities and concerns are summarised below.

- *interface of an application*: interface should take into account the physical parameters of the device, the variety of situations of its use and the set of functions provided for the user. It is possible highlight three important aspects when designing and developing an interface:
    1. *multimodality of the interface*: in some case it is advisable to develop a multimodal interface - an interface that uses several modalities, i.e. inherent human forms of influence on the device, for context-sensitive interaction between the application and the user.
    2. *interface adaptability*: an important aspect of interface design is its adaptability to different devices in terms of their physical properties. Since the application can be used on a number of devices with different physical parameters, for example, screen size and resolution, it is necessary to dynamically adapt the interface for the most effective use of them.

3. *usability*: the issue of usability is especially relevant for mobile device applications in conditions of resource scarcity and physical limitations - the application should allow solving the problem with the maximum convenience for the user in the minimum possible time.

- *contextual awareness of the application*: using the capabilities provided by modern mobile devices (various sensors of location, motion, temperature, illumination, etc.), the application can analyse the conditions of its operation in highly dynamical changing context - the physical environment, computing resources - and automatically adapt the mode of its operation.
- *computing resources*: mobile devices use special computing processors with special parameters. In addition, multitasking mobile operating systems are used, sharing computing resources between all processes in the system. Complex computing processes therefore can take longer and lead to an overall decrease in performance and increased power consumption
- *limited storage space*: mobile devices have limited built-in RAM. Moreover, the operating system may limit the amount of memory available for the application and its service data. The feasibility of storing a large amount of data should be evaluated and, if necessary, the external memory of the device (memory cards) should be used or data should be stored outside the device (for example, use a cloud service).
- *power consumption*: the amount of time a mobile device can operate autonomously without additional charging is a critical parameter for the user. Excessive or inappropriate use of the device's resources (for example, non-disconnecting of data transmission interfaces after the end of the communication session, etc.) can lead to a rapid discharge of the battery of a mobile device and decrease the duration of its autonomous work;
- *limited time of continuous work*: a mobile device is distinguished by a high dynamics of its use, characterised by many often unpredictable events that can occur. It should be borne in mind that the application may terminate its work at any time (for example, the user switched to another task, the battery of the mobile device is discharged, the device's resources have run out, etc.);
- *unpredictability of the moment when the application is interrupted by external events*: the operation of a mobile application at any time can be interrupted by other more important events (for example, a phone call, a received message, etc.), which must be taken into account when developing an application and always, if possible, return the application to the last state before its work was interrupted;
- *multi-interface of the device*: the use of multiple input / output interfaces within one user session, the use of specific interfaces - NFC, compass, gyroscope, light sensor, accelerometer, various sensors (movement, orientation, etc.) that are not available for stationary personal computers;
- *limited / interruptable communication channel*: since a mobile device in most cases uses various wireless communication technologies and also moves in space,

the stability and quality of communication is constantly changing. If the application uses a data transmission channel, then all situations of breakage and loss of communication quality must be correctly processed;

- *multi-platform*: in some cases it turns out to be necessary to develop several applications in parallel for various device families (for example, iOS, Android) to cover the maximum possible number of end users of the application;
- *testing on emulators*: given the large number of different models of mobile devices (of different generations and classes), it is often not possible to test the application on real devices, therefore, in some cases, the only way out is to test the application on available software emulators;
- *mandatory observance of the requirements and development rules for further distribution of the application*: for a number of platforms, special requirements and rules for the development of applications are put forward for their further sale through a centralized application distribution system (for example, Apple), which must be taken into account when designing and developing a mobile application;
- *additional censorship of the application's functionality*: a number of companies (for example, Apple) check the source code of the application and its work before publishing it in the application distribution system;
- *user event notification system*: mobile applications are characterized by frequent changes in the context of use and frequent switching between applications, it is important to inform the user about important events if another application is active at the time of their occurrence;
- *operation of the application in online / offline modes*: if the operation of the application depends on external data stored outside the mobile device, and the availability of the communication channel is limited, depending on the requirements for the application, it may be advisable to preload the necessary information and save it on the mobile device for further use;
- *limited availability of external peripheral devices* (eg, printing, etc.);
- *limited availability of common software technologies* (eg flash, audio / video codecs);
- *building web applications*: for a number of tasks it may be more appropriate to develop a mobile web application (using html5 technology) that can run on multiple platforms instead of developing a native application for a specific platform.

4.5 Enterprise applications quality model

For quality assurance it is essential to formalise and constantly evaluate the quality of enterprise application, trying to define the issues on early stage and initiate appropriate requirements and changes. This is a difficult task because enterprise applications are complex software products and it is necessary to clear define the purposes, models and methods of quality assurance. Especially important in the context of enterprise applications, taking into account the user-centred focus, transformation of ideology from mainly technical to business-orientation, is to measure and assure the quality on the business-role level, where many aspects come into play together. Some business-processes are

a singe step or a single-application process, whereas another ones assume the usage of several apps and different forms of interaction between them.

A business role represents the responsibility for performing specific behaviour, to which an actor can be assigned (i.e. a user), or a part an actor plays in a particular action or event. Business roles with certain responsibilities or skills are assigned to business processes or business functions. A business actor that is assigned to a business role is responsible for ensuring that the corresponding behaviour is carried out, either by performing it or by delegating and managing its performance. A business role maybe be assigned to one or more business processes or business functions, while a business actor may be assigned to one or more business roles [Gro].

As was showed in chapters 2, 3 the model used for quality evaluation and assurance for enterprise application is to be based on the actual SQuaRE series of standards.

While developing the quality models it makes sense to pay attentions to the external quality and quality in use for the following reasons:

- the enterprise software often allows to get on early stages already working solution, because the enterprise applications are in most of the cases already delivered as some basic ready-to-use solution;
- if the customisation processes is required the results can be only seen and evaluated in test environment with test data or in real production environment;
- in case of own developments or / and modifications of the delivered applications or components it is to be done on known-platform using known technologies, frameworks, libraries and integrated development environment. So it makes sense to measure external quality of intermediate versions of already working solutions and not try to predict it is based on the evaluation of internal quality;
- the quality of enterprise applications should be evaluated and assured in the test environment with model data or while the real production use;
- the quality of enterprise applications should be evaluated primarily not from the view of technical users but from the view of real business users in the context of their business-roles taking into account the feedback from them and changing the requirements accordingly;
- for enterprise applications the result of the work of the software is important and not the details of the internal implementation;
- development process of the enterprise application implies the constant release of new versions or the updates for rapidly changing business environment and requirements. Moreover in contemporary software development practice agile development methods are incorporated, what is aimed at fast provision and deployment of working versions to test and production environment;

Because of these considerations the reliable data about the current version of application can be obtained by the external quality and quality in use evaluation.

In this thesis the first ***outlines of the top level*** *of quality models are proposed limiting by the selection of characteristics. The choice of the subcharacteristics, measures and evaluations methods are the tasks for the other detailed research.*

The developed model assume that each characteristic has a conformity subcharacteristic, i.e. the ability of the enterprise application to comply with the standards and conventions associated with this characteristic. Otherwise, if the enterprise application does not meet the accepted standards and agreements, the quality of the enterprise application cannot be assessed.

4.5.1 External quality model

The standard ISO/IEC 25010 shows the influence between quality models by the quality characteristics (see Table 4.3).

Table 4.3: ISO25010 - Influence of the quality characteristics

Software product properties	Computer system properties	**Product quality characteristic**	**Influence of quality in use for primary users**	Influence on quality in use for maintenance tasks	Information system quality concerns of other stakeholders
-	-	Functional stability	x		
-	-	Performance efficiency	x		x
-	-	Compatibility		x	
-	-	Usability	x		
-	-	Reliability	x		x
-	-	Security	x		x
-	-	Maintainability		x	
-	-	Portability		x	

The mentioned relations were taken into consideration because as was stated before enterprise applications are evaluated using external quality and quality in use.

The proposed external quality model for enterprise applications is a hierarchical structure consisting of three levels - characteristics, subcharacteristics and measures of quality. The model is based on model for product quality (see figure 3.7) from the standard ISO/IEC 25010 [Sta11a], would use and adapt the number of measures from the standard ISO/IEC 25023 [Sta16b]. The model includes the following characteristics: functional suitability, performance efficiency, usability, reliability, maintainability, portability and can be represented as a set Q_e:

$$Q_e = FS, PE, U, R, M, P \tag{4.1}$$

where FS - is a set of subcharacteristic of functional suitability;
PE - is a set of subcharacteristic of performance efficiency;
U - is a set of subcharacteristic of usability;

R - is a set of subcharacteristic of reliability;

M - is a set of subcharacteristic of maintainability;

P - is a set of subcharacteristic of portability.

In the model are not included the following characteristics defined by the standard ISO/IEC 25010: *compatibility*, *security*.

Compatibility - degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment. This characteristic is not included while enterprise applications are developed in and for a special enterprise infrastructure (hardware platform, used software technologies, defined communication protocols between component and appropriate services, defined interfaces to the communication with the external system etc). So the concerns and agreements of intercommunication between applications, components, internal and external systems are covered by enterprise software environment.

Security - degree to which a product or system protects information and data so that persons of other products or systems have the degree of data access appropriate to their types and levels of authorisation. In the enterprise software environment the security is a comprehensive topic and the most security concerns are covered by different parts of the infrastructure and not by the applications themselves. Database provides the mechanism for data encapsulation, encryption and isolated data containers with authorisation services, the hardware provides physical security of data (secure datacenters, cloud storage), there also centralised authorisation services which control users and their permissions. So in most of the cases enterprise applications work with the data they were allowed to and special permissions are granted to users which are authorised to run these applications. Special cases for special applications which are specially dealing with security topics can be covered by the extended quality model with the security characteristic and appropriate subcharacteristics, measure and methods of evaluation defined.

The top-level model of external quality for enterprise applications is shown in the figure 4.20.

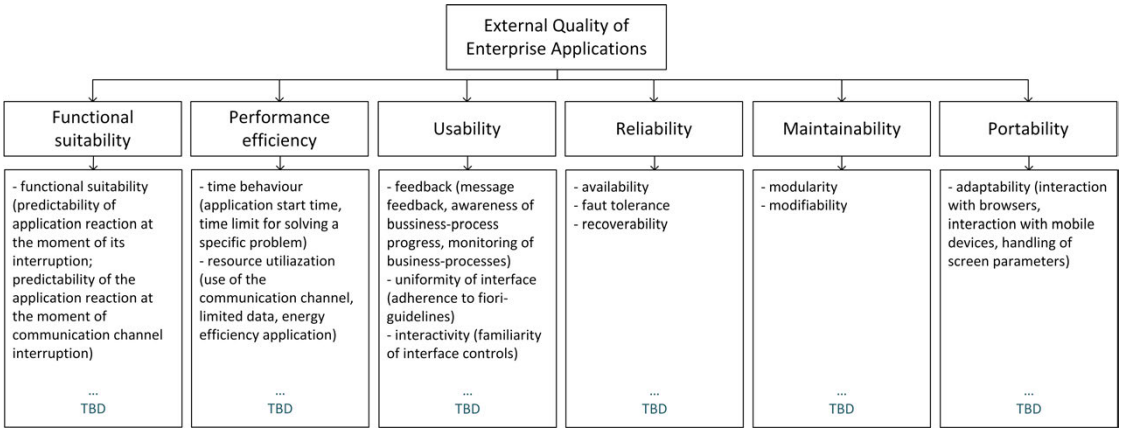

Figure 4.20: External Quality of Enterprise Applications - top level definition

Functional suitability - degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions.

Performance efficiency - performance relative to the amount of resources used under stated conditions.

Usability - degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use

Reliability - degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.

Maintainability - degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers.

Portability - degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.

The development and choice of subcharacteristic and appropriate measures and evaluation methods are the topics of other thorough research. In the figure are already shown as the starting point of their further development some subcharacteristics and metrics which do not exist in the SQuaRE standards.

4.5.2 Quality in use model

The proposed quality in use model for enterprise applications is a hierarchical structure consisting of three levels - characteristics, subcharacteristics and measures of quality. The model is based on model for quality in use (see figure 3.8) from the standard ISO/IEC 25010 [Sta11a], would use and adapt the number of measures from the standard ISO/IEC 25022 [Sta16a]. The model includes the following characteristics: effectiveness, efficiency, satisfaction, context coverage and can be represented as a set Q_u:

$$Q_u = EF, ES, S, C \tag{4.2}$$

where EF - is a set of subcharacteristic of effectiveness;
ES - is a set of subcharacteristic of performance efficiency;
S - is a set of subcharacteristic of satisfaction;
C - is a set of subcharacteristic of context coverage.

In the model are not included the following characteristics defined by the standard ISO/IEC 25010: *freedom from risk*.

Freedom from risk - degree to which a product or system mitigates the potential risk to economic status, human life, health, or the environment. The decision about the use of enterprise software solution is a thorough multi-criteria process taken on the company management level assessing before the possible risks from different perspectives and divisions. For law-regulated ares of activity appropriate valid certifications of compliance must be available. The main risks therefore are evaluated on the higher level and not on the level of separate applications. For especially sensitive data or business-processes enterprise software environment can provide additional protection measures, as the restriction of access, use only of defined, controlled and audited API-functions, restrictions of use of some technologies etc.

The top-level model of quality in use for enterprise applications is shown in the figure 4.21.

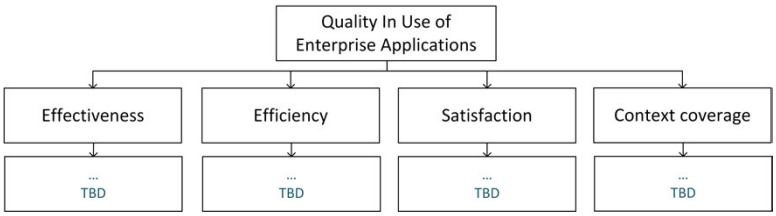

Figure 4.21: Quality in Use of Enterprise Applications - top level definition

Effectiveness - accuracy and completeness with which users achieve specified goals.

Efficiency - resources expended in relation to the accuracy and completeness with which users achieve goals.

Satisfaction - degree to which user needs are satisfied when a product or system is used in a specified context of use.

Context coverage - degree to which a product or system can be used with effectiveness, efficiency, freedom of risk and satisfaction in both specified context of use and in context beyond those initially explicitly identified.

The development and choice of subcharacteristic and appropriate measures and evaluation methods are the topics of other thorough research.

4.6 Quality models in enterprise applications lifecycle

The quality assurance processes should be integrated into the enterprise software lifecycle to enable regular or selective and specific quality evaluations, which would allow to make

appropriate decisions and take measures against detected quality flows in a timely and suitable manner. One of the possible ways to do it is shown in the figure 4.22.

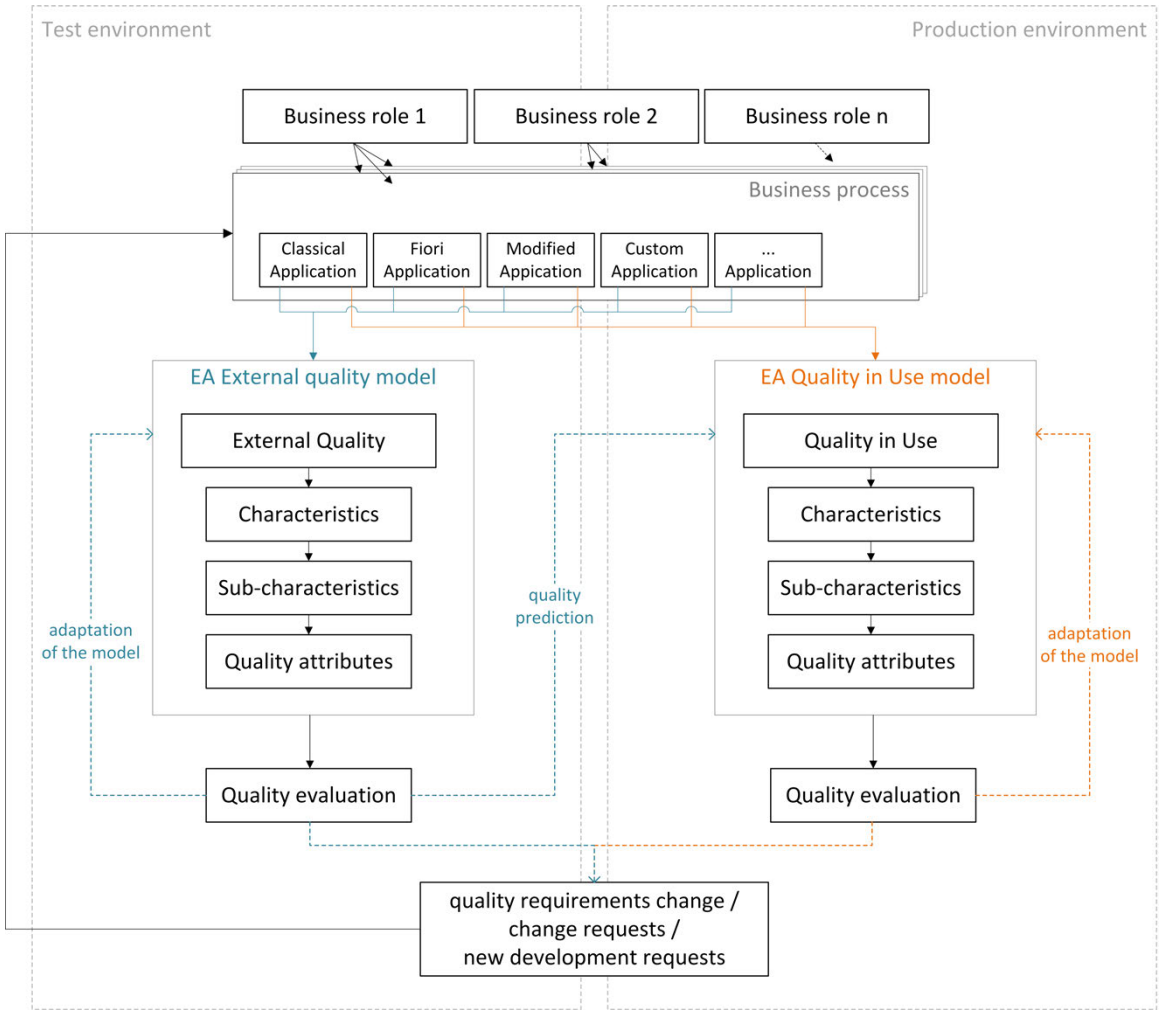

Figure 4.22: Enterprise applications quality models in a lifecycle

To amend the detected quality flaws by implementing the necessary or implied modifications the following re-engineering strategy is proposed - see figure 4.23. The same strategy can be applied in case of migration of some existing custom solutions to a new technology, for example from classic SAP-GUI to SAP Fiori to ensure that required quality goals are remained fulfilled or even became stronger after the transition to the re-engineered solution.

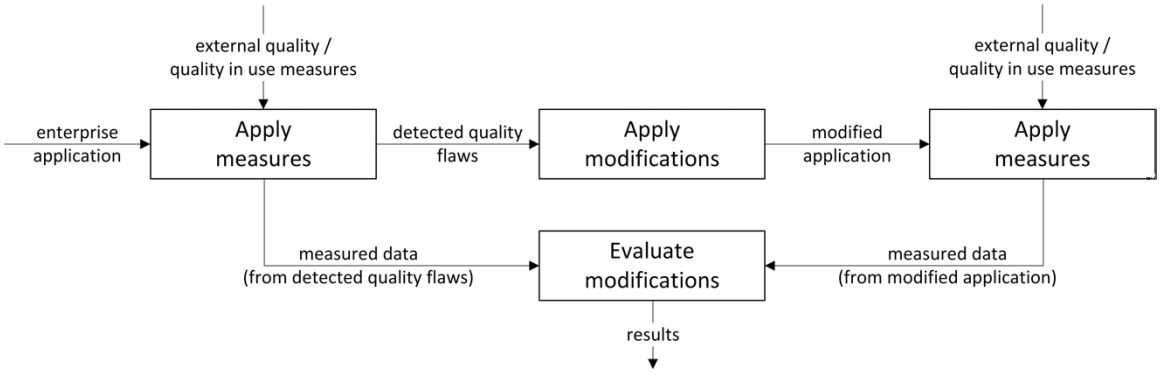

Figure 4.23: Enterprise application re-engineering strategy for quality flaws

The shown strategy is realised in the following three phases:

1. The external quality or / and quality in use (depending on the need only required measures / subcharacteristics / characteristic could be considered) is measured for the application of interest. This can be intentionally initiated evaluation or the result of the evaluation of the regular quality assurance routine revealing the quality flaw(s).
2. Implement the modifications / extensions of the application taking into account the detected quality flaws.
3. Perform exactly the same quality measurements as in the phase 1 by re-applying the chosen measurement criterias to the modified application from the phase 2. At the end compare the results to make the judgement about if the application improvements are compliant with and achieving the required quality goals.

This re-engineering strategy can be applied so many times as necessary until the required quality goals are reached. The strategy might be applied to a single application as well as to the whole business-process consisting of several applications by setting and measurement appropriate quality attributes in a given context.

5 Conclusion

The history of software quality assurance problem is already almost six decades long. In chapter one the long evolution path of this challenging and interesting problem was shown from the fist qualify models to the latest international standards and a multi-parameter analysis was carried out. The ideas, principles and practical applications of all the models and interdisciplinary approach formed the basis of modern mature standards in the area of software quality.

Chapter two gave the description and analysis of the standards ISO/IEC 25000 SQuaRE and showed the possibilities of extension and further development of quality models. Especially important for the quality measurement methods and quality assurance processes are the measures, the analysis of properties and validity criterias of which in accordance with the standards' approach and recommendations was conducted.

Enterprise software is always in the phase of active growth, extension and is in general a rapidly changing software family, adapting and using all of the latest advances in the field of diverse data analysis, including internet of things, big data and machine learning, incorporating latest practices and technologies related to the user-interface and data representation, module and distributed software technologies. These development directions were demonstrated in the third chapter on the example of SAP-software as the major enterprise applications provider worldwide. Taking all these concerns into consideration the properties and features of the modern hybrid enterprise applications were analysed and summarised, what is relevant and important for the development and adaptations of the appropriate quality measures and evaluations methods.

Finally, the top-level external quality and quality in used models based on the SQuaRE standards were developed and proposed. They include top-level characteristics, related to the enterprise applications usage context, and it is the first important step and the solid basis for further development of the models including measures and quality evaluation and assurance methods. Additionally, the concerns of the integration of quality assurance processes into a software lifecycle and applications re-engineering strategy were addressed.

This thesis showed the utmost importance of the quality assurance of enterprise applications problem and the pressing needs for the researches and new developments in this direction. The provided analytics and based on it and on the actual standards in software quality area top-level quality models are to be further developed. The choice of the adequate models' subcharacteristics, appropriate measures and corresponding quality assurance methods are the next consecutive steps, tasks and challenges for future work.

References

[MJ77] Walter G. McCall J. Richards PP. *Factors in Software Quality.* US Rome Air Development Center Reports NTIS, 1977.

[BB78] Lipow M. Boehm B.W. Brown J.R. *Quantitative Evaluation of the Software Quality.* TRW Systems and Energy Group, 1978.

[Cro79] Philip B. Crosby. *Quality is free.* New York: New American Library, 1979.

[GR87] Caswell D. Grady R. *Software metrics: establishing a company-wide program.* NJ Prentice-Hall, 1987.

[W.87] Perry W. *Effective methods for EDI quality assurance.* Prentice-Hall, 1987.

[JJ88] Gryna F.M. Juran J.M. *Quality Control Handbook, 4th ed.* McGraw-Hill, 1988.

[T.88] Gilb T. *Principles of software engineering management.* Addison-Wesley, 1988.

[W.S89] Humphrey W.S. *Managing the Software Process.* Addison-Wesley, Reading, Mass., 1989.

[V.R93] Basili V.R. *Applying the Goal/Question/Metric Paradigm in the Experience Factory.* Institute for Advanced Computer Studies, University of Maryland, 1993.

[BV94] Rombach H. Basili V. Caldiera G. *The Goal Question Metric Approach.* Encyclopedia of Software Engineering, 1994.

[R.G95] Dromey R.G. *A Model for Software Product Quality.* IEEE Transactions on Software Engineering, 1995.

[HL96] Rosenberg L.H. Hyatt L.E. *A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality.* Product Assurance Symposium and Software Product Assurance Worksop, 1996.

[R.96] Fitzpatrick R. *Software quality: definitions and strategic issues.* Dublin Institute of Technology, 1996.

[R.G96] Dromey R.G. *Cornering the Chimera.* IEEE Software, vol. 20, 1996.

[R.97] Pressman R. *Software Engineering: A Practitioner's Approach, 4th Edition.* McGraw-Hill, 1997.

[EE98] Institute of Electrical and Electronics Engineers. *1061-1998. Standard for Software Quality Metrics Methodology.* Software Engineering Standards Committee of the IEEE Computer Society, 1998.

[BJ99] Kitchenham B. Pasquini A. Boegh J. Depanfilis S. *A Method for Software Quality Planning, Control and Evaluation.* IEEE Software, vol. 23, 1999.

[L.99] Tahvildari L. *Non-Functional Requirements in Software Engineering*. Kluwer Aceademic Publishers, 1999.

[Sta99] International Organization for Standardization. *ISO/IEC 14598-1:1999 Information technology - Software product evaluation - Part 1: General overview*. International Organization for Standardization, 1999.

[FN00] Pfleeger S.L. Fenton N.E. *Software Metrics: A Rigorous and Practical Approach*. Thomson Learning; Revised Auflage, 2000.

[Sta01] International Organization for Standardization. *ISO/IEC 9126-1:2001. Software engineering - Software product quality - Part 1: Quality model*. International Organization for Standardization, 2001.

[AA02] Sellami A. Abran A. *Initial Modeling of the Measurement Concepts in the ISO Vocabulary of Terms in Metrology*. Proceedings of the 10th International Workshop on Software Technology and Engineering Practice, 2002.

[BA02] Visaggio G. Bianche A. Caivano D. *Quality Models Reuse: Experimentation on Field*. Proceedings of the 26th IEEE Computer Software and Applications Conference, 2002.

[CL03] Yu E. Mylopoulos J. Chung L. Nixon B. A. *Quality-Driven Object-Oriented Re-engineering Framework. PhD Thesis*. Electrical and Computer Engineering Waterloo, 2003.

[Sta03a] International Organization for Standardization. *ISO/IEC 9126-3:2003 Software engineering - Product quality - Part 3: Internal metrics*. International Organization for Standardization, 2003.

[Sta03b] International Organization for Standardization. *SO/IEC 9126-2:2003 Software engineering - Product quality - Part 2: External metrics*. International Organization for Standardization, 2003.

[Sta04] International Organization for Standardization. *ISO/IEC 9126-4:2004 Software engineering - Product quality - Part 4: Quality in use metrics*. International Organization for Standardization, 2004.

[AA05] Desharnais J. M. Habra N. Abran A. Al-Qutaish R.E. *An Information Model for Software Quality Measurement with ISO Standards*. Proceedings of the International Conference on Software Development (SWDC-REK), 2005.

[R.09] Pressman R. *Software Engineering: A Practitioner?s Approach, 7/e*. McGraw-Hill, 2009.

[Sta11a] International Organization for Standardization. *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, 2011.

[Sta11b] International Organization for Standardization. *ISO/IEC 25040:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Evaluation process*. International Organization for Standardization, 2011.

[Sta12] International Organization for Standardization. *ISO/IEC 25021:2012 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Quality measure elements*. International Organization for Standardization, 2012.

[Sta14] International Organization for Standardization. *ISO/IEC 25000:2014 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE*. International Organization for Standardization, 2014.

[AG15] Padilla Omiste A.E. Azgaldov G.G. Kostin A.V. *The ABC of Qualimetry. Toolkit for measuring the immeasurable*. Ridero, 2015.

[Sta15a] International Organization for Standardization. *ISO 9000:2015 Quality management systems - Fundamentals and vocabulary*. International Organization for Standardization, 2015.

[Sta15b] International Organization for Standardization. *ISO/IEC 25024:2015 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of data quality*. International Organization for Standardization, 2015.

[Sta16a] International Organization for Standardization. *ISO/IEC 25022:2016 Systems and software engineering - Systems and software quality requirements and evaluation (SQuaRE) - Measurement of quality in use*. International Organization for Standardization, 2016.

[Sta16b] International Organization for Standardization. *ISO/IEC 25023:2016 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality*. International Organization for Standardization, 2016.

[Sta17] International Organization for Standardization. *ISO/IEC/IEEE 15939:2017 - Systems and software engineering - Measurement process*. International Organization for Standardization, 2017.

[K.K18] Kammaje K.K. *SAP Fiori Certification Guide*. Rheinwerk Publishing, 2018.

[BJ19] Franke J. Koehler B. Morgenthaler J. Butsmann J. Crumbach M. *SAP S/4HANA Embedded Analytics: Architektur, Funktionen, Anwendung*. Rheinwerk Publishing, 2019.

[R.19] De Louw R. *SAP HANA 2.0 Certification Guide*. Rheinwerk Publishing, 2019.

[SE19] SAP SE. *HA450 - Application Development for SAP HANA*. SAP SE, 2019.

[Sta19] International Organization for Standardization. *ISO/IEC 25020:2019 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Quality measurement framework*. International Organization for Standardization, 2019.

[SE20a] SAP SE. *SAP01 - SAP Overview*. SAP SE, 2020.

[SE20b] SAP SE. *UX100 - SAP Fiori - Foundation*. SAP SE, 2020.

[Gro] The Open Group. *ArchiMate 3.1. Specification, a Standard of The Open Group*. URL: `https://pubs.opengroup.org/architecture/archimate3-doc/toc.html`. (accessed: 25.10.2020).

[SE] SAP SE. *SAP Fiori Design Guidelines*. URL: `https://experience.sap.com/fiori-design/`. (accessed: 16.10.2020).

[Wik] Wikipedia. *Enterprise software*. URL: `https://en.wikipedia.org/wiki/Enterprise_software`. (accessed: 10.10.2020).