QUALITY MANAGEMENT IN DISTRIBUTED SOFTWARE DEVELOPMENT

Application within a highly innovative and security sensitive software development process

Abstract

The field of software quality management has been subject to extensive studies. This is due to the fact that software is getting more and more important in every situation of life. However, in consequence of disparate requisites according to the respective development setting, as well as product attributes and – finally – diverging understandings of quality itself, there is no such thing as a silver bullet for achieving a high quality software product. This paper describes a quality management system that has been especially designed for assessing and improving the quality of a security sensitive software product (software for performing elections over the Internet) under the constraint that there are geographically and organizationally separate partners involved in the software development process.

Keywords: Quality Model, Quality Management, Online Voting.

1 INTRODUCTION

It is an undisputable fact that computer systems in general and software in particular have become an intrinsic part of economic, public as well as private sector in recent years. Online shopping, electronic tax returns or mobile phones – software nowadays can be found virtually everywhere. On the one hand, this can be appreciated since software is intended to ease our lives in various ways (e.g. by saving time or reducing effort/costs). On the other hand, however, this trend must be observed with greatest care. Along with this continuously growing importance of software it is also the software dependency which is growing at least in equal measure. As in many other areas, it is rather dangerous to get into too strong dependencies. This might lead to severe problems in case that something is going wrong.

Often software failures are comparatively minor and thus need no further discussion. However, this situation is completely different if you think about the software used in flight management systems, nuclear power plants or, as further addressed in the following, in electronic voting systems. In these cases even the smallest software bug might have disastrous consequences. Having a look at the literature, dozens of software related incidents can be found, which have caused high financial or even human losses. The Therac-25, for example, was a machine for radiotherapy whose protection mechanisms against excessive radiation doses were controlled by software. Caused by some errors in this software at least four people lost their lives and many others got severely injured (Leveson et al. 1993). Another example is the rocket Ariane-5 which exploded on its maiden flight in 1996. It turned out that parts of its software were inherited untested from the Ariane-4. Unforeseen incompatibilities resulted in defective flight control commands, self-destruction and a loss of about 500 million dollars (Pfleeger 2001, p. 37). As a result, huge effort is being raised during the software development stages to completely eliminate or at least minimize software bugs.

The previous explanations have shown that correctness is a fundamental factor that distinguishes a good software product from a bad one. One now might argue that this “good” software is of a high quality since it is working correctly and does not contain any bugs. This sounds evident, but unfortunately it is not that simple. In fact, (software) quality is a rather abstract term that still lacks a commonly agreed upon definition (NIST 2002, p. 1-3). Instead, quality more or less must be seen as a subjective term which is understood and interpreted differently by different people respective within different contexts or situations.

In addition to correctness, it might also be important that a software product is working efficiently in terms of response times or hardware requirements. Similarly, a quality valuation on a software product might be heavily influenced by its properties with regard to security (e.g. access control, data protection). As can be seen on these examples, quality is a very comprehensive concept that needs to be further refined in order to make it usable. More precisely, stating that a (software) product is of a high quality means that this product has a number of specific (quality) characteristics and fulfils a set of (quality) requirements.[1] This is where quality models come into play. A quality model offers a well proven way to further formalize the abstract term quality and thus to make it easily applicable within a specific context. Moreover, quality models must be regarded as the basis of any quality management activity. Thus, they are a major key to developing high quality software.

In the following we will describe our quality model as well as the related quality management processes which especially target distributed development of security sensitive software products. At first, chapter 2 gives a short overview over the fundamentals regarding quality and how it can be

[1] This complies with ISO 9126 (1991) in which software quality is described as the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs. Furthermore, ISO 9000 (2000) defines quality as the degree to which a set of inherent characteristics fulfils requirements.

achieved. For this purpose well-known approaches in this area are briefly presented. Based on these fundamentals, we describe our approach defining a comprehensive quality model as well as the necessary organizational structure and responsibilities in chapter 3. In addition to that, chapter 4 demonstrates the realization and execution of our quality management approach within the EU-funded project "e-VOTE" as an example. The goal of this project is to develop an Internet based electronic voting system. Finally, a conclusion will be drawn in chapter 5.

2 RELATED WORK

The field of software quality management by far is not as new as one might think. The first findings that dealt with quality aspects of software date from the time of the software crisis and the formation of Software Engineering in the early 1970s. Since that time a lot of research has been put forward. According to De Oliviera et al. (2002), software quality nowadays can be divided into two main areas. On the one hand there are many techniques and approaches that are trying to assure the quality of software products by closely examining their significant characteristics (*product*-oriented approaches). On the other hand, many approaches are focusing on the definition, evaluation and improvement of software development processes, basically assuming that a high quality development process results in a high quality product (*process*-oriented approaches). Since previous experiences have shown that both product and process orientation are very promising, we briefly overview their most important approaches in the following subchapters. In chapter 3 we indicate the influences of the existing approaches on the quality management methodology developed within e-VOTE.

2.1 Product-oriented Approaches to Software Quality

In principle it is common for all product-oriented approaches (i.e. quality models) that they refine the rather abstract term software quality into more significant (quality) characteristics. The most important quality models are those proposed by McCall et al. (1977), ISO 9126 (1991) as well as the more general GQM approach developed by Basili et al. (1994).

One of the first quality models was presented by McCall, Richards and Walters in 1977. In their approach software quality is subdivided into the three categories product operation, product revision and product transition which are assigned 11 characteristics, so-called *factors* (e.g. correctness, usability, maintainability). In a next step these factors are further refined into 25 so-called *criteria* (e.g. completeness, consistency, modularity). Finally, the quality model defines 41 *metrics* that can be used to measure the degree to which these different criteria are fulfilled.

The quality model proposed by McCall et al. (1977) highly influenced the further research activities in this area[2]. One of the more recent quality models can be found in ISO 9126 (1991). Within this standard software quality is itemized into six characteristics (functionality, reliability, usability, efficiency, maintainability and portability). In addition to that, the appendix contains a proposal of 21 sub-characteristics that can be used for a further refinement of these main characteristics (e.g. interoperability, fault tolerance, testability). Unfortunately, according measurement methods (metrics) are not defined.

In contrast to the quality models mentioned above a slightly different approach to software quality was introduced by Basili, Caldiera and Rombach (1994). According to Balzert (1998, p. 263), their Goal Question Metric (GQM) approach can be seen as a method that leads to a development-specific quality model. First of all it is necessary to define the relevant *goals* that should be achieved in terms of software quality. In a next step each goal is refined into a number of *questions* whose answers can

[2] Similar approaches comprise the quality model proposed by Boehm (1978), the Software Quality Metrics (SQM) model (Murine 1984) as well as the Software Quality Attributes (SQA) model (Bowen et al. 1985).

be helpful in achieving the goal. Finally, *metrics* are derived which contribute significant quantitative information for answering the questions.

2.2 Process-oriented Approaches to Software Quality

As opposed to product-oriented approaches to software quality, process-oriented ones can be characterized by not directly focusing on the software product itself respectively its quality-relevant characteristics. Instead, they rather try to (indirectly) improve the quality of a software product by assessing and improving the quality of the software development processes. The main approaches in this respect are the Capability Maturity Model (CMM), Bootstrap as well as the Software Process Improvement and Capability dEtermination (SPICE).

The Capability Maturity Model (Paulk et al. 1993) was developed at the Software Engineering Institute (SEI) in 1987. It can be seen as a framework for evaluating the maturity of software developing companies which is especially used as the basis for continuously improving their development processes. The stage of maturity is divided into five levels (initial, repeatable, defined, managed, optimizing). Starting at the initial level, each superior level comprises a set of so-called Key Process Areas that must be completely fulfilled in order to reach it.

A European pendant to the CMM is the Bootstrap method which was developed within an EU-funded project (Haase et al. 1994). Conceptually, Bootstrap is quite similar to the CMM as it basically uses the same maturity levels. In addition to that, each level is further separated into quartile, thus allowing a finer granularity regarding the maturity of the development processes.

More recently, SPICE was developed to form an international standard for software process assessment (ISO/IEC 15504 1998). SPICE has the advantage that it combines the experience and improved understanding gained in developing CMM, Trillium, Bootstrap and other models (Wang et al. 1997). Similar to the approaches mentioned above, it defines six maturity levels which are built up on each other (incomplete, performed, managed, established, predictable, optimising). Furthermore, a maturity level is determined for each single process allowing exactly focussed improvements.

3 GENERIC QUALITY MANAGEMENT METHODOLOGY

The previous chapter has shown that many approaches supporting quality management have been developed over the years. The abundance of research activities in the field of software quality management, however, also indicates that the silver bullet, which guarantees the development of high quality software products in every possible setting, has not been encountered yet. In general, the existing approaches need to be adjusted in order to properly fit into a specific context. In the present case, i.e. with regard to the e-VOTE project, we needed a quality management system that is able to cope with two peculiar conditions:

1) The e-VOTE system is developed by several geographically and organizationally separate partners.
2) The e-VOTE system is a highly innovative and security sensitive product.

The first constraint makes it hard to use process-oriented quality management approaches as the different partners normally do not have common development processes. Instead, each partner is rather working on its tasks and according to its internal processes and procedures. It also might be not advisable to adapt process-oriented models isolated for every partner, as from a higher level perspective interrelations between tasks performed by different partners can be identified. In addition to that, almost all process-oriented approaches aim at a continuous process improvement and therefore are especially useful for software developing companies. In the case of the e-VOTE project this would cause unnecessary effort since it is a nonrecurring development project. Thus, it is advisable to use the product (i.e. the e-VOTE system) as center of reference between the partners and to focus on a product-oriented quality management approach.

However, the second constraint makes it hard to simply adopt predefined product-oriented approaches (i.e. quality models). It was already mentioned that quality models need to be adjusted according to the specific context. This is especially true in the case of the e-VOTE project as there are hardly any quality models that are focusing on security aspects. In our case special care is necessary, because the software product is very specific and makes heavy use of complicated and security relevant features. Therefore we finally decided to develop our own generic quality model which is mainly influenced by the spadework of McCall (1977) as well as the GQM approach (Basili et al. 1994).

3.1 The Generic Quality Model

As can be seen in Figure 1, our quality model is hierarchically structured, having one root node representing the overall quality of the software product to be developed. This node has several child nodes whose number may vary depending on the respective software product. These child nodes constitute quality characteristics, which correspond to the factors in McCalls quality model. Each quality characteristic represents an intrinsic aspect of the software contributing to its overall quality (e.g. usability, security).

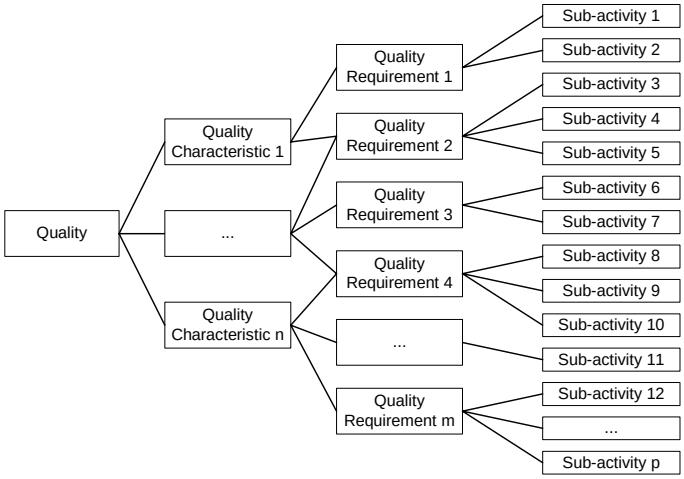

Figure 1: General Structure of the Quality Model

In contrast to McCall we did not refine the characteristics into sub-characteristics respective criteria. Instead of this, we associated the quality characteristics with a variable number of quality requirements which represent concrete demands towards the according software product (e.g. “The system should support different languages and the user must be able to change them.”). Thus, quality requirements more or less can be compared to the goals within the GQM approach. Using (quality) requirements as one of the layers in our quality model has the advantage that these can be easily adopted from the requirement analysis, functional specifications, etc.[3] to a large extend. Thus, requirements can be directly linked to the according quality characteristic.

Finally, we needed a mechanism to evaluate the fulfilment of the specified quality requirements. In general, all existing quality models are using metrics for this purpose. The disadvantage of metrics is that they are still poorly defined (NIST 2000, p. 1-6). On the one hand, this means that specific software attributes can be measured using different metrics which are possibly leading to different evaluation results. On the other hand, there are many attributes that are nearly impossible to evaluate using metrics. Moreover, normally the metrics that are easiest to measure are also the ones which are least important with regard to software quality (NIST 2002, p. 1-8).

[3] These are documentations that are commonly prepared within a software development process.

As a consequence, we decided not to solely focus on metrics but to rather use a more general evaluation approach that gives a detailed description of the actions to be performed. We called this the concept of "sub-activities". Each sub-activity defines a concrete action which must be performed by the respective evaluator. As can be seen in Figure 1, each requirement has one or more sub-activities assigned which can be handled like a checklist. This detailed definition of how to perform the evaluation of a certain requirement also assures the integrity of the evaluation results over consecutive evaluation rounds. In the case of the e-VOTE project we realized this checklist approach by using specific "Quality Requirement Evaluation Forms" (cf. Figure 5).

Everything put together, the presented quality model as well as its evaluation procedures are highly flexible and thus can be easily customized according to specific needs. Therefore, it can be regarded as a viable approach for assessing and improving the quality of software products.

3.2 Organizational Structure and Responsibilities

The set-up of an adequate organizational structure for quality management along with the assignment of responsibilities is a crucial factor. This holds especially true, when quality management has to be performed across organizational boundaries. Therefore, a specific structure for the assignment of personnel has to be developed that reflects this organizational peculiarity.

In our approach, we have singled out one partner, who holds a central role for the quality management, i.e. the Quality Assurance Coordinator (QAC). He defines the overall quality strategy, manages the quality model and is responsible for the conduction of the necessary quality evaluations. Furthermore, he can be seen as the supreme controlling entity.

In addition to that, every other partner is assigned quality evaluation responsibility as well. On each partner side one person is identified who is responsible for the enforcement and completion of the assigned evaluation processes. As compared to the QAC, these persons constitute Local Quality Managers (LQMs). They have the necessary insight in their organization in order to locate adequate evaluators and can be regarded as link between the overall quality management (represented by the QAC) and the final evaluators.

For the final evaluators we defined roles, which serve to classify testing competences. The roles are assigned depending on the tasks of a certain partner i.e. programming or user tests. This serves to address the right partners, respectively Local Quality Managers, with evaluation forms according to their testing competencies. The assignment of roles also provides additional guidance for the Local Quality Managers to forward their respective forms to the final testers.

Depending on the software product and its development process, different roles can be identified for a testing environment. First there is always the role developer. Second there will be one or more roles in a productive setting (users). Figure 2 exemplifies such an organizational structure within the e-VOTE project.

4 QUALITY MANAGEMENT WITHIN THE E-VOTE PROJECT

After having described our approach for a generic quality model as well as the suggested organizational structure for enforcing it, we will now demonstrate its proper usage considering the e-VOTE project as an example. e-VOTE[4] is an IST research project, which is funded by the European Union within FP5 (IST-XXXX-XXXXX). Its aim is to design, implement and validate an Internet based electronic voting system that provides all required services for organizing and conducting a

[4] The full title of the project is "e-VOTE – An Internet Based Electronic Voting System". Additional information can be found on the projects website under http://www.XXXXXXX.XX/XXXXX

voting process. This process can be anything ranging from referenda and simple polls up to general elections. The project started in October 2001 and it will be concluded on schedule in March 2004. The project consortium consists of seven partners from five European countries (Denmark, Germany, Greece, Luxembourg and Slovakia). According to their different background, i.e. University, Company or Municipality, each partner is assigned specific tasks.

From the very first moment the project partners have been aware of the fact that an electronic voting system needs to be highly secure and that it will be challenged with an enormous amount of requirements (both from government and voter respectively user side). Therefore one main part of the development activities is especially dedicated to quality management. One of the partners is assigned the role of the Quality Assurance Coordinator who has to gather all quality related issues within a Quality Handbook (XXXXXX 2002, XXXXXX et al. 2003). This is different to similar projects since quality management usually is not dedicated an own work package. The Quality Handbook is the central document describing the quality strategy as well as all procedures and responsibilities necessary for easily evaluating the quality of the system. Since the e-VOTE system is developed by several distributed partners having different backgrounds (and thus different working practices) it is appropriate to use a quality model which is rather focusing on the software product than on the development process. Finally it was decided not to use an existing product-oriented quality approach, but to base all quality management activities on an own quality management model (as described in chapter 3.1) better supporting all characteristics inherent in the project.

4.1 Initial Set-up of the Quality Management System

Following steps have to be undertaken in order to implement and conduct a quality evaluation process:

- Set-up of an organizational structure for the quality evaluation procedure

At the beginning, an adequate organizational structure has to be set-up according to the abovementioned considerations (cf. chapter 3.2):

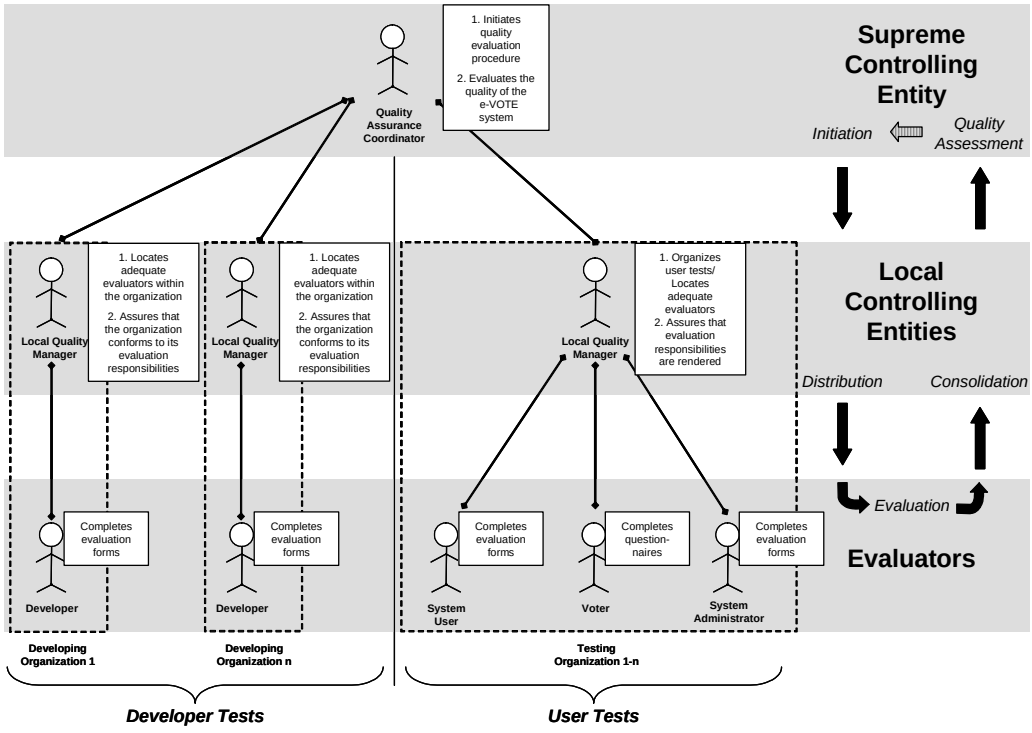

Figure 2: Organizational Structure for Quality Management in the e-VOTE Project

- Definition of roles and according evaluation responsibilities

In the course of setting-up the organizational structure roles have to be defined, too. For e-VOTE we define the roles “developer”, who is occupied with the programming of the system, “system administrator”, who installs and runs the program, “system user”, who mainly has administrative tasks for the program (like content management) and “voter” (i.e. end user). Following this basic classification using roles, we are able to identify two general evaluation environments.

First, there are the developer tests, which are performed by the individuals actually programming the software. They have special insights regarding the technical implementation of the product, e.g. they know the code and the software architecture. They are occupied with test cases having the character of tests commonly referred to as white-box tests.

Second, there are user tests, in which the software is generally evaluated in a productive setting. This testing environment is especially useful for assessing soft factors, e.g. the attractiveness of the user interface or the satisfaction with response times. The people who evaluate the software during user tests usually have no special knowledge about the implementation of the product, thus following a black-box-testing approach.

In a next step, the general quality model has to be applied. Aiming at assessing the quality of the e-VOTE system, quality requirements and quality characteristics have to be formulated.

- Formulation of quality requirements

Several project partners contribute quality requirements to the system, e.g. the partner, who defines the use cases for the e-VOTE system, prepares a catalogue of applicable functional and user requirements. Apart from that, contractual requirements must also be included. All gathered requirements are discussed within the consortium of project partners and commonly agreed on. As a result, there is a list of 80 requirements related to different aspects of the system to be developed.

- Identification of quality characteristics

Furthermore, quality characteristics have to be identified. This task is fulfilled by the Quality Assurance Coordinator. Following nine quality characteristics are identified as most important for the system at hand:

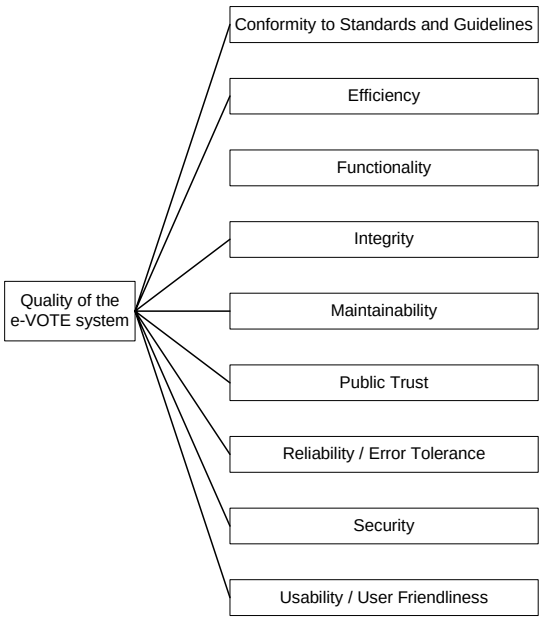

Figure 3: Quality Characteristics identified in the e-VOTE Project

Especially the quality characteristics “Conformity to Standards and Guidelines” as well as “Public Trust” show the peculiarity of the venture, as general purpose software products usually need not cope with these kinds of demands. Security is an ever increasing claim; however, in the present environment of building a product to support democratic (voting) processes and dealing with extremely sensitive information, it is of vital importance.

- Assignment of quality requirements to quality characteristics

In a following step, the collected requirements have to be assigned to the quality characteristics according to their scope. It must be noted that - depending on their influence - requirements may be linked to one or more characteristics, i.e. representing a 1:n setting (cf. Figure 1). As an example, the requirement “Voters’ and candidates’ data stored by the system should not violate the data protection legislation” has been set in relationship to the quality characteristics “Public Trust” as well as “Conformity to Standards and Guidelines”.

To get a clearer picture, the following figure provides an exemplary overview of some requirements that are joined to the quality characteristic “Security”:

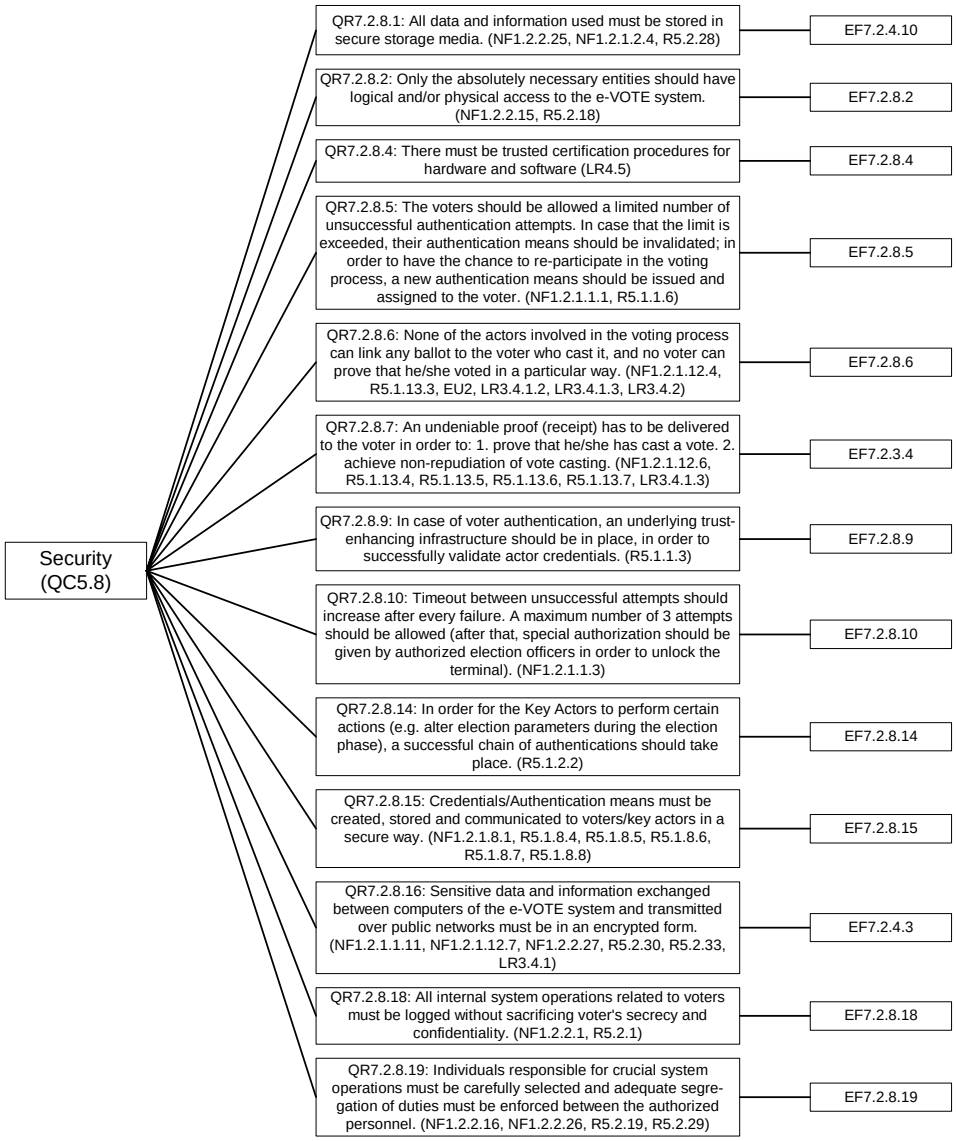

Figure 4: Exemplary Subset of Quality Requirements related to Security

- Formulation of sub-activities for each quality requirement

Finally, the necessary sub-activities are formulated in order to properly evaluate the requirements. For each requirement the according sub-activities are put together in a specific Quality Requirement Evaluation Form (cf. Figure 5).

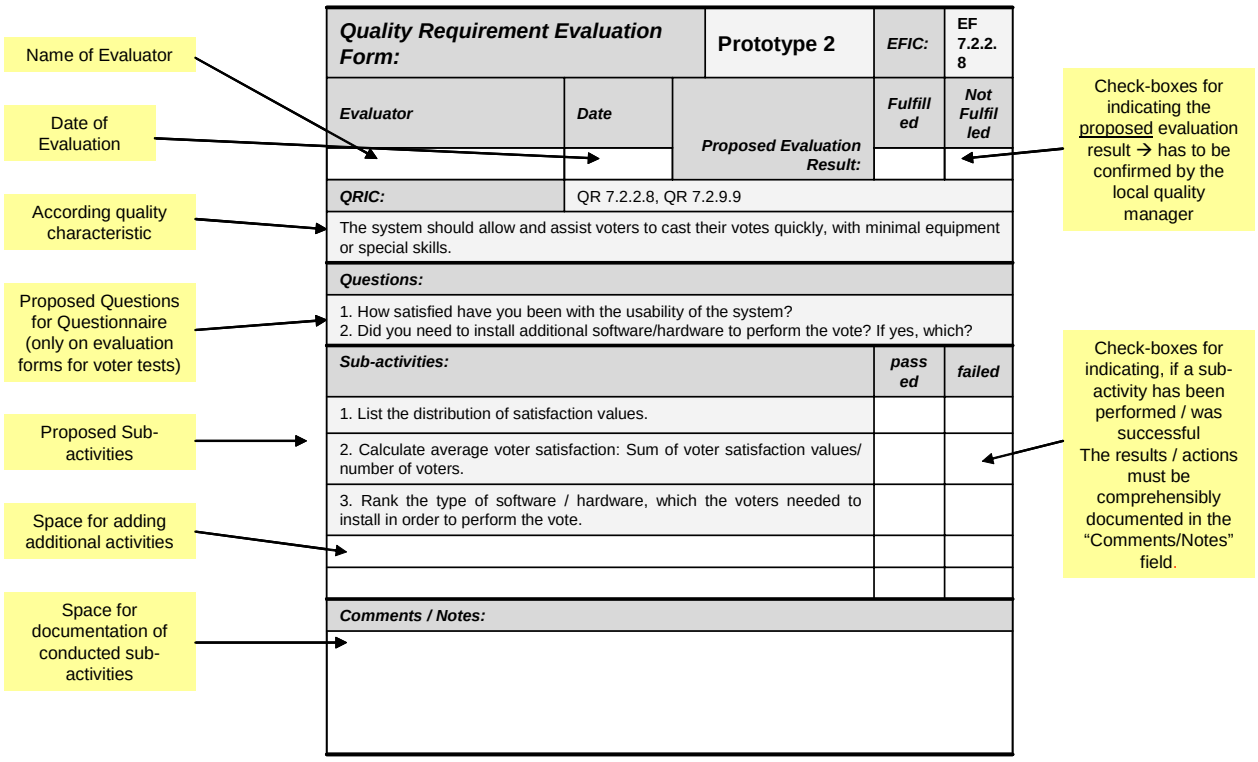

Figure 5: Exemplary Quality Requirement Evaluation Form

4.2 Conduction of the Quality Evaluation Process

- Distribution and processing of quality requirement evaluation forms

In an initial step, the Quality Assurance Coordinator sends the quality requirement evaluation forms to the Local Quality Managers in the participating organizations. They forward these forms to the final evaluators. The respective evaluator performs the actions described under sub-activities (cf. Figure 5), documents his steps and results in the comments field and checks on fulfillment of each sub-activity. Finally, he proposes an evaluation result (requirement is fulfilled or not fulfilled) and hands the completed forms back to the Local Quality Manager, who returns the completed forms to the Quality Assurance Coordinator. The evaluator also has the possibility to propose improvements for existing sub-activities or to add new ones. These changes must be confirmed by the Quality Assurance Coordinator.

- Control of performed evaluations

The Quality Assurance Coordinator examines all forms and controls the work of the evaluators. He either accepts their proposed results or asks for refinement. The sum of all evaluation forms constitutes the basis for the quality assessment of the e-VOTE system.

- Consolidation of information and documentation of results

Finally, the Quality Assurance Coordinator prepares a document which consolidates the complete set of evaluation information. For each quality characteristic the number of completed and failed

requirements is stated. The requirements that have not been fulfilled are especially pointed out. These requirements are further divided into those requirements which were intended to be fulfilled at this development stage (e.g. in prototype 1) and those which apply to a later version (e.g. the pilot system). Afterwards, the document is sent to all Local Quality Managers and not completed requirements are discussed for the purpose of improvement (e.g. on the next meeting).

- Implications on quality assessment

Our evaluation system neglects to compute a value for quality. A percentage of fulfilled requirements could easily be assessed; however, it is questionable if a value of e.g. 90% percent of fulfilled requirements would mean 90% quality. In addition, there are requirements that are unequally important for the realization respectively more or less easy to implement. One can think of a prioritization of requirements but this can hardly be reached following any formal rule and thus would lead to rather arbitrary classifications. Every requirement has its justification for existence within the quality system - otherwise it could be left out. One might argue with the image of a chain, which is only as strong as its weakest link. Therefore every open point has to be addressed individually, taking into account all consequences which are related to it.

- Technical implementation

The e-VOTE software development process follows a spiral model, where in three consecutive rounds two prototypes and, finally, a pilot system are developed. The quality assessment for the former prototypes was done by textual documents, which were sent via email. However, the mass of processed information calls for a more efficient way to handle the quality evaluation procedure. Therefore, an online quality tool is currently under construction for conducting the quality evaluation process for the pilot system.

The processes of the online quality tool basically follow the defined quality strategy. The online quality tool offers a web interface, which can be used by evaluators to complete the abovementioned evaluation forms easily. Thus, it improves efficiency of the quality evaluation within the distributed software development process.

5 CONCLUSION

In this paper we presented a quality management methodology that is successfully employed within a large software project for the development of an Internet based electronic voting system. The project has two main characteristics influencing quality management:

- Distributed software development
- Innovative and highly security critical application domain

The quality management methodology guarantees that aims defined at the beginning of the project are not getting lost during its course. Especially in a distributed development project there is always the threat that one relies on the other or that no one feels responsible for a particular demand. Besides technical issues, this is addressed by the quality management method as well.

The outcome of the quality management methodology does not lead to one value that expresses the whole aspects of quality in a single number - and we doubt that this is possible at all. We come to the conclusion that expressing quality in a quantitative value implies a mathematical correctness that is not given by any means and creates the illusion that quality itself is quantifiable. However, following a quality strategy in a structured way and with adequate controls, keeps these aspects in the minds of the involved persons. This is of particular importance for security sensitive applications.

From our experience with developing and administering a quality management system, we believe that the most crucial element of such a system must be seen in the definition of adequate quality requirements. Special care is needed for defining requirements in a good manner taking into account various issues like scope, unambiguity etc. from the very beginning.

6 ACKNOWLEDGEMENT

This work is supported by the European Union under grant IST-XXXX-XXXXX (FP5). Their support and cooperation with our project partners XXXXXXXXXXX, XXXXXXXXXXX XX XXXXXXXXXX XXXXXXXXX XXXXXX, XXXXXX self-governing region, XXXXXXX XXX XXXXXXXXXX X.X. and XXXXXXXXX XX XXX XXXXXX is highly appreciated.

References

Balzert, H. (1998). Lehrbuch der Software-Technik, Vol. 2. Spektrum, Heidelberg, Berlin. (in German).

Basili, V.; Caldiera, G.; Rombach, D. (1994). Goal Question Metric Paradigm. In, Marciniak, J.J. (Editior). Encyclopaedia of Software Engineering, Vol. 1, pp. 528-532. Wiley, New York.

Boehm, B.W. (1978). Characteristics of Software Quality. American Elsevier, New York.

Bowen, T.P.; Wigle, G.B.; Tsai, J.T. (1985). Specification of Software Quality Attributes. Rome Air Development Center, RADC-TR-85-37, Vol. I-III.

De Oliviera, K.M.; Rocha, A.R.; Weber, K.Ch. (2002). Workshop on Software Quality. Proceedings of the 24th International Conference on Software Engineering, pp. 671 – 672, ACM Press, New York.

XXXXXX, H. (2002). e-VOTE Project Deliverable D 5, Quality Handbook for the e-VOTE System Services.

XXXXXX, H.; XXXXXXX, J.; XXXXXXX, E. (2003). e-VOTE Project Deliverable D 5-R, Quality Handbook for the e-VOTE System Services (Revised).

Haase, V.; Messmarz, R.; Koch, G.; Kugler, H.J. (1994). Bootstrap Fine-Tuning Process Assessment. IEEE Software, July, pp. 25-35.

ISO 9000 (2000). Quality Management Systems – Fundamentals and Vocabulary. International Organization for Standardization.

ISO 9126 (1991). Information Technology Software Product Evaluation – Quality Characteristics and Guideline for their use. International Organization for Standardization.

ISO/IEC 15504 (1998). Information Technology – Software Process Assessment, Part 1-9. International Organization for Standardization.

Leveson, N.G.; Turner, C.S. (1993). An Investigation into the Therac-25 Accidents. IEEE Computer, July, pp. 18-41.

McCall, J.; Richards, P.; Walters, P. (1977). Factors in Software Quality. NTIS AD-A049-014, 015, 055.

Murine, G.E. (1984). Measuring Software Quality Metrics. Quality Progress, Vol. 17, No. 5, pp. 16-20.

NIST (2002). Planning Report 02-3, The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology.

Paulk, M.C.; Curtis, B.; Chrissis, M.B.; Weber, C.V. (1993). Capability Maturity Model, Version 1.1. IEEE Software, Vol. 10, No. 4, pp. 18-27.

Pfleeger, S.L. (2001) Software Engineering: Theory & Practice. Prentice Hall, New Jersey.

Wang, Y.; Court, I.; Ross, M.; Staples, G.; King, G.; Dorling, A. (1997). Quantitative Evaluation of the SPICE, CMM, ISO 9000 and BOOTSTRAP. Proceedings of the 3rd International Software Engineering Standards Symposium (ISESS '97), pp. 57-68, IEEE.