Effective Quality Management: Risk- and Value-based Software Quality Management

Alexander Poth, TU Berlin
Ali Sunyaev, University of Cologne

Software quality management has to effectively deploy resources for quality assurance activities in order to reflect the achieved product quality. Therefore, quality managers should exploit their creative freedom to direct their courses of action within the economic constraints. We present a method-based approach for better effectiveness. The product risk- and value-based method is applicable in the context of the ordering party’s quality supplier management, and also for the product development of contractors. This is due to the method’s product function orientation and to the independence of the software development procedures.

Keywords: quality management, verification and validation, value-based software engineering, risk management, supplier management

Adequate software quality is a key delivery of software projects. There are various definitions for software quality. The wider definition of software quality includes, besides others attributes, reusability and maintainability. Stewart studied the restrictions of management in general and formulated the findings demands, constraints and choices [1]. In the context of quality management these findings are refined to: the product is delivered with the demanded quality, and the choices to be made within given constraints. Considering companies’ objectives to implement adequate software quality, this article focuses on a transparent and effective resource allocation of quality assurance (QA) activities for product verification and validation.

We sought to understand how quality managers and other software product stakeholders can come to an agreement about QA activities, when they make decisions with a focus on software functionality or features. In particular, we wanted to improve the acceptance of the software quality by the customer at release time by using a structured method [2] employing selected QA activities. At this point, the transparencies of the conducted QA activities, and their results, have to give confidence to all parties in order to release the software. To give confidence in the results, our method makes transparent the suitability of choices between mitigation of the quality risks, and the constraint of limited resources for QA activities for delivering the demanded quality.
Recently, research on selected phases of the software development process has received much attention as it has attempted to understand and improve software quality management (SQM) choices during the software life cycle. For example, Bach proposes risk-based testing [3] for the end of the development phase, while Boehm and Huang suggest value-based software engineering [4] as a holistic approach for software development that includes QA. However, most previous research considers a selected viewpoint, rather than a SQM viewpoint, within one software development phase.

We used a combined value-engineering (VE) [5] and risk-management (RM) [6] approach in order to get acceptance of adequate product quality assurance activities by a minimum set of stakeholders: the customer, the developing party, and the software operating party. QA activities are based on constructive and analytic QA activities, which are selected for the specific software product context. This is motivated by the observation of Gough, for whom VE and RM are twin powers [7]. We applied our QA method, Effective Quality Management (EQM), to projects in different domains from enterprise to embedded software to prove our generic and holistic quality management method in practice.

**Box: EQM - Effective Quality Management**

The EQM method is based on Value Engineering & Management (VE) and Risk Management (RM). VE is an effective method to identify the value of a product’s functions. The basic method was developed at General Electric (GE) after the Second World War to use restricted resources to implement the most valuable functions. VE is now an integral part in many engineering disciplines focusing on cost optimization. RM is an effective method to identify product risks and serves as a part of many industrial sectors’ processes.

EQM assumes that variation exists in every software product’s life cycle. The unexpected variation causes SQM adjustments to handle new situations with suitable QA. EQM is based on the Deming cycle also known as the Plan-Do-Check-Act (PDCA) cycle [8], an established quality process for continuous improvement. EQM extends the Deming cycle with the Identification phase, which is used for the systematic analysis of the product functions to identify their value and risks. The extension results in the Identification-Plan-Do-Check-Act (IPDCA) cycle (see Figure A). The result of the Identification phase is that the functions of a product are identified and their value and risks are evaluated in the quality risk aspects. The Plan phase results in a set of validation measures. The validation measures are selected for suitable function validation to mitigate the identified quality risks of the functions. The Do phase executes the defined validation measures for the functions. The Check phase looks for the effectiveness of validation measures by checking product quality indicators. The Act phase sets up corrective actions to improve the product quality if needed. Corrective actions or improvements also include adaptation of quality risk aspects (for improvement of the quality focus value), validation measures (if new methods are available) or product quality indicators (to reflect product usage better). One matrix row of figure A
contains a product function like \textit{login to system} and its quality risk aspects evaluation like \textit{workaround options and efforts or complexity of the function}. The evaluation result is the quality focus value which is mitigated by the selection of validation measures like \textit{requirements review} or \textit{unit tests}. The effectiveness of the function validation is monitored by the values of the product quality indicators like \textit{amount of defects}. The evaluation and monitoring can use predefined value groups like \textit{middle} for easier evaluation or checks.

**Figure A.** The IPDCA cycle is mapped to a matrix. The EQM matrix implements all activities of the IPDCA cycle for application in software projects.

Although many industrial sectors have adopted VE and RM and many publications have reported success in their application (see [4] and IEEE Software - Special issue on Risk, May/June 1997), their use together in engineering is still a matter of debate [7]. One major issue is how risks and values fit together. Despite this issue, we designed EQM to handle value as an aspect that can be identified from the user’s viewpoint. Risks should be identified from the user’s and product developer’s viewpoint. From the point of view of a product developer, this entails the risk that a user’s expected value will not be fulfilled. This leads us to express value as a quality risk aspect in EQM, because EQM supports a product view of SQM in the product development and maintenance phases.
**Method description**

We designed EQM to help SQM to negotiate acceptable quality targets with the stakeholders of all parties, and to adjust them over the product life cycle if needed. Main stakeholder parties are users or customers, the development department, and the operational management. Often in projects some stakeholders, like users or customers, do not personally participate in the QA planning process, and make only a review of the QA strategy and plan. In this case, in the first step, the SQM has to substitute for the missing stakeholders in the QA planning meetings. In the second step, the SQM has to legitimate the plan for the stakeholders to accept. The same happens if changes with the planned QA activities are required to react to unexpected occurrences which cause adjustments to the planning.

The IPDCA-cycle of EQM guides the SQM during the product life cycle:

- **Identification**: The common terminology of all stakeholders is the set of the functions and features of the software. Our method uses the function/feature list to break down the software product into manageable units which are available over the product life cycle. Furthermore, the stakeholders identify the value and quality risk aspects (QRA) of the product. Value can be applied by the ranking of functions in a more VE style or in QRAs like mission relevance, percentage of function usage per day or do nothing costs. The QRAs are categorized in groups like high impact, middle impact, and low impact. In the next step, the functions are evaluated against the QRAs. If possible, the evaluation is made with all relevant stakeholders ending up with a common understanding of the function value and risks. The quality focus value (QFV) is calculated based on the results of the evaluation of the functions.

\[
QFV = \sum_{QRA=1}^{n} Value_{QRA}
\]

In a second step the weighting factor (WF) for QRA is useful. The WF helps to prioritize on the level of the QRAs. Furthermore, the WF can be used to emphasize QRAs in different phases of the product life cycle.

\[
QFV = \sum_{QRA=1}^{n} (Value_{QRA} \times WF_{QRA})
\]

- **Plan**: The stakeholders plan the validation measures (VM) for the product functions. In the first step the set of VMs is defined. The VMs handle verification activities, too. VMs contain analytic and constructive QA like reviews, static code analysis, unit-tests and if necessary the coverage targets etc. If an established set of VMs is used, the costs and the effects of the VMs in the product domain are known. This knowledge is useful to optimize the value and effectiveness of
QA by selection of VMs. The targets of the VMs are defined. Keep in mind when defining targets DeMarco’s [9] observation that qualitative targets can be more useful in practice than quantitative targets.

Depending on the demanded quality, different best-practice strategies can be applied for the assignment of VMs to functions. The first strategy that we suggest for a cost-limited product development is to start with the function with the highest QFV; and then derive explicitly only the minimum set of VMs to mitigate the risks as much as necessary by minimal costs. This is done up to the point that the entire budget is spent or all functions have a minimum set of VMs (minimal costs for product). If the budget has been spent before all functions are assigned with VMs, the stakeholders have to decide whether to increase the budget or to live with the transparent validation gap of the product. If the entire budget has not been spent, we would then start with a next iteration of the list to derive additional VMs for improving the product quality and iterate until the entire quality budget is spent (product quality fits cost target). For the second strategy, we suggest deriving VMs motivated by the value of the QFV. By this strategy the set of suitable VMs defines the quality budget. For each function a suitable set of VMs is derived to mitigate the quality risks to an acceptable level. Functions with a higher QFV receive more resources for validation in both strategies. Furthermore the stakeholders discuss and decide together about the spending of resources for VMs.

In case of unexpected occurrences and unplanned changes to the project with impact to the QA resources, the SQM sets up corrective action mostly by reallocation of VMs to handle the impacts.

- **Do**: Execution of the planned VMs.

- **Check**: Controls the execution of the planned VMs and updates the product-quality indicators (PQIs). PQIs are metrics to measure the product quality. As best practice, we suggest using some customer or user PQIs for final product quality information and some development phase internal PQIs for the early set-up of corrective actions. The customer or user PQIs, for example bug reports, are important because the internal PQIs can have good results in the case of good quality. In the case of unsuitable or insufficient VMs, no quality issues might be detected. During the set-up of EQM, the PQIs can be defined. Examples are bugs from testing, customer or user feedback and review findings, which are assigned to functions. To show the effectiveness of early error detection of the VMs, a mapping of review findings, bugs, etc., to the development phase assigned PQIs is useful.

- **Act**: Sets up a continuous improvement, based on Kaizen [10]. If the PQIs are not as expected, a root-cause analysis is started. This analysis typically leads to more knowledge about the
effectiveness of the selected VMs. This knowledge will be used to define the corrective actions. A second focus of this phase is the monitoring and optimizing of the effectiveness of the VMs.

Method application

We designed EQM to be applicable in different product development contexts and we demonstrate this on some EQM application examples. In all product development contexts we had positive experiences with the acceptance of the EQM derived VMs, and their resource allocation for QA activities and adjustments during development. Because we were able to motivate the decisions and made the resulting quality gaps transparent.

V-Model

The V-model example is based on the electric/electronic development of an engineering company. The functions or features are evaluated on QRAs which are mostly motivated by the V-Model 97 (Development Standard for IT Systems of the Federal Republic of Germany – General Directive No. 250/3) like criticality or complexity but also extended with specific QRAs. One of the specific QRAs is mission relevance, which expresses the value aspect of a function in the customer or user value view of the product (Figure 2). The functions and their features like, for example, the change to neutral gear, are evaluated and ranked for each QRA. The QFV is calculated based on the quality-risk evaluation. The WF depends on the focus of the quality-risk aspect in the project or product life cycle phase - represented by the numbers in the first line above of the QRAs. For some selected QRAs we defined validation measure sets to fit V-Model 97 requirements, which associate results of the QRA evaluation to VMs. Validation measure sets define quality measures for the values of a quality-risk aspect. EQM defines validation measure sets for packaging a group of the independent VMs to a logical monolithic VM for the mitigation of a specific quality risk. A V-Model example in which we applied validation measure sets was the quality risk aspect criticality. Criticality should be independent of management choices because the associated VMs are constraints with predefined target values of the validation measure. These constraints are defined in the validation measure set of criticality like code coverage targets for unit tests of critical functions.

We derived VMs for not predefined VMs by validation measure sets based on the QFV and the specific quality-risk profile of a function. In the example this is visible by the selection of more VMs for the feature forward in comparison to neutral to mitigate its higher QFV. Furthermore, the quality target of the VMs is defined like the MISRA (automotive coding conventions) conformance analysis or the explicit unit test coverage value targets.

In a next step, we integrated EQM into the development process. This will lead to a function or feature list in the requirements specification tool which can be used for EQM. The project schedule and resource
allocation (the responsible person for the validation measure) are defined in the project management tool, which handles all project tasks including QA tasks. Bugs and change requests are associated with the function list, which has a working backward chain to provide current quality information. The information of the backward chain is used to validate the effectiveness of the functions’ selected VMs. The direct mapping makes it easier to classify bugs and change requests, because the QRAs can be used as classification information. Furthermore, the association is useful for quick validation effort estimation because function changes have to be validated according to the EQM.

Figure 1. EQM as a matrix with orientation on the V-Model 97. This leads to a project-independent set of quality risk aspects (see criticality and complexity) and validation measures.

**SCRUM**

The SCRUM (scrumalliance.org) example is based on the software for an airline’s customer benefit program. The software involves developing two separate sub-systems. One system is being developed in India, the other in Germany. Both systems are interacting with more than 50 other systems or interfaces. Due to the different business cases or functions that are handled with the system, different teams for systems integration and user acceptance tests were set up. Three functional teams are located in different locations. A team of nine members is responsible for the spend stage of the benefit program, and work according to SCRUM.

The first important step is to define the right QRAs for the software product context. An important aspect of SCRUM, or rather agile project management, is that it is value and priority driven. One way to map the value paradigm of agile development to the QRAs is to define risk values in terms of money. As an alternative to the quantitative, money driven approach, we selected a qualitative driven approach. This requires the definition of a set of comparable units (like in Figure 1 for each quality-risk aspect: the low, middle and high impact). The project defined five different types of value and risk. In the presented project, the context of the system’s function is the spend environment in call centers, which are dealing with different spend use cases and their activities. Functions (‘stories’ in SCRUM terminology) are activities of use cases. The first QRA is value from the user’s viewpoint. In order to make the...
contribution of its activities for the daily business transparent, value measures how often the function is used during the daily business (0-20%, 21-40%, 41-60%, 61-80% and 81-100%). We split the other risks of functions into two QRAs. One was based on the customer/user viewpoint of the probability of loss occurring impact (not really an impact, impact with an inexpensive workaround that is hidden from the customer, impact with an expensive workaround that is hidden from the customer, impact with a workaround that is visible to the customer and show stopper). A last risk aspect makes it transparent that some functions are important for a very small customer group, which generates very high revenue – but is a high risk if these customers could not be served.

With this set of QRAs we covered all relevant value and risk drivers in the QFV. This approach brings together both the risk-based quality management and the value-driven agile development paradigm. We transformed values in a risk view, via an accepted table of classification units for values and risks. The point is: losing high-value functions due to defects provokes high costs – this leads to high quality risks. Vice versa: quality issues due to these risks reduce the value of a function.

Based on this set of QRAs, a prioritization in the values of the functions was easy, because adequate risk mitigation delivers more value per function. Furthermore, we defined a set of VMs for the handling of the quality risks. This led, in our case, to a mandatory and optional (and selectively used measures like pair working versus reviews or early regression-testing) set of quality assurance activities. One PQI we used is bugs per SCRUM story or function.

SPICE

The SPICE (ISO/IEC 15504) example is based on the electric/electronic product development organization of an automotive supplier. For the implementation in software product specific domains of an organisation, tailoring of the EQM parameters is needed. The challenge is to keep the balance between a very specific definition of the parameters in a specific domain context, and its reusability for similar or other domain contexts. EQM solves this parameter issue by using a flexible level of abstraction of the product by the functionality, which is applicable in almost all domains and the flexible set of QRAs and PQIs. Furthermore, EQM fulfils normative requirements by its openness towards the VMs that are being used. SPICE requires that the VMs are associated to processes. In a development process environment conforming to SPICE the applicable set of VMs has to fit this requirement by VMs, which conform to the defined and applied processes. We mapped the VMs to processes of the automotive SPICE, like the MISRA analysis, as a part of the unit verification strategy of the ENG.6 (software construction) process. ENG.6 is controlled by the SUP.1 (quality assurance) process, which includes EQM as part of its project quality assurance strategy. Equally important for an implementation in an organisation is the possibility of the refinement of the EQM method parameters which allows a step by step introduction of EQM. EQM provides this with an initial set-up of functions, the quality risk aspect...
evaluation and the VMs. In a next step these could be refined or enhanced, once more information becomes available on the product, the organisational constraints or other details are provided. For example, after a few projects in dynamic multinational development teams, we added as an explicit quality-risk aspect: know-how/experience level of developer.

A major constraint that has to be managed by an organisation is the training of those employees who need to be acquainted with the usage of EQM. Further constraints are the management of change and the controlling of the implementation progress during a method rollout. After EQM rollout to the electric/electronic product development organization, we verified the establishment of EQM in independent SPICE assessments. The assessments were conducted by customer assessors in different international development locations.

Summary

We have designed, applied, and improved EQM over the last six years in different domains, in small and large projects, in product development programs, and software systems development organizations, and got confirmations in SPICE assessments. By the transparency of EQM, all projects improve the acceptance for their QA activities by the relevant stakeholders. We believe that EQM can help QA teams make the right choices and make the product quality transparent, by monitoring quality risks and product evolution over its life cycle. Figure 2 shows the life cycle of function-development in a SPICE development context during two iterations (x-axis) of the IPDCA-cycle as relations of the QFV and PQI per function (y-axis) and iteration. Functions with only one QFV are only included in one release. Figure 2 shows that during the iterations the QFV becomes smaller and more precise. A smaller QFV leads to less (cheaper) and more precise VMs, which lead to a better and more precise PQI values.
Figure 2. Two iterations of the IPDCA-cycle with the evaluated QFVs of product functions and the associated PQI “defects” after the application of the derived mitigation VM.

The EQM is useful for prioritizing QA efforts in practice with a preventive focus. EQM is useful to understand the quality risks of functions and their changes, and thus to prioritize QA efforts effectively. For example, the QA team could prioritize efforts for new functions by selecting specific risk mitigation activities.

EQM should be carefully used in contexts where the quality risks are unclear and the effectiveness of risk mitigation by VMs is unknown. For example, if only a default set of quality risks are used, then the derived VMs won’t always lead to good product quality. We cannot recommend applying the EQM method without reflecting critically upon the suitability of the identified quality risks and defined VMs.

To implement EQM in a running program or organization, we recommend starting with the preparation of PQIs by root-causing value and quality issues like defects back from operation. These results are used to motivate the change to EQM for SQM. The next step is to check if all relevant VMs for a potential mitigation of the analyzed defects are available for the product developers. If not, start the training of VMs to the people. Parallel to the VM analysis, it’s possible to define QRAs and evaluate the functions. The last step brings all together in the IPDCA-cycle. EQM was successful applied in project acquisition,
redesign, and development projects. As of now, two companies integrate EQM in their standard processes.

References


About the authors

Alexander Poth manages the corporate IT-QA for a major automotive OEM and was a senior consultant at SQS Software Quality Systems AG. Contact him at poth.de@gmail.com

Ali Sunyaev is an assistant professor at the Department of Information Systems, University of Cologne, Germany. Dr. Sunyaev has (co)-authored several international journal articles (including articles in leading journals such as Communications of the ACM, ACM Journal of Data and Information Quality and IEEE Computer). Contact him at sunyaev@wiso.uni-koeln.de