Software Quality Development and Assurance in RUP, MSF and XP - A Comparative Study

Wolfgang Zuser
Vienna University of Technology
wolfgang.zuser@inso.tuwien.ac.at

Stefan Heil
Capgemini Consulting Austria
stefan.heil@capgemini.com

Thomas Grechenig
Vienna University of Technology
thomas.grechenig@inso.tuwien.ac.at

ABSTRACT
The support of software quality in a software development process may be regarded under two aspects: first, by providing techniques, which support the development of high quality software and second, by providing techniques, which assure the required quality attributes in existing artifacts. Both approaches have to be combined to achieve effective and successful software engineering.

In this study, we compare three of the most industrially relevant software development process models (Rational Unified Process (RUP), Microsoft Solution Framework (MSF) and Extreme Programming (XP)) regarding their software quality support in terms of software quality development and software quality assurance. Based on the results we propose a de-facto standard for quality support in software development process models.

Categories and Subject Descriptors
D.2.9 [**Management**]: *Software process models (e.g., CMM, ISO, PSP), Software quality assurance (SQA).*

General Terms
Management.

Keywords
Software Quality Development, Software Quality Assurance, Software Process Models.

1. INTRODUCTION
Successful software engineering strongly depends on the delivery of high quality software. High quality is typically defined by quality attributes like customer satisfaction (which is mainly determined by being on budget and time), adherence to functional requirements, usability, security, stability and many more.

Besides skilled people, an appropriate and well-working software development process is a key success factor for achieving the demanded high quality software. Traditional software development process models four core process steps: analysis, design, implementation and test. These processes guarantee the correct analysis of requirements and their valid implementation. They do not necessarily support quality attributes like usability or security.

Some process models include deployment and maintenance activities after test. These processes support the stability of the running system. Many software process models define accompanying software project management activities, which mainly support the quality attributes customer satisfaction by keeping the project on time and budget.

However, most of them do not explicitly define software quality development and software quality assurance as a part of the software development model itself. Yet, these quality support processes are vital for achieving high quality software including all quality attributes listed at the beginning of this section.

Typical software quality assurance models are designed as "add-on" for software development processes, yet with some major restrictions:

- "Add-on" software quality assurance should be performed by an independent quality assurance department. Due to budget and personnel restrictions, the organizational structure is not easily adapted to meet this requirement in many small and medium software companies.
- "Add-on" software quality assurance uses methods for finding defects, which may already exist in artifacts. This is an expensive method to ensure quality support compared to the prevention of defects in the development process.

Software quality development produces the same degree of software quality avoiding these pitfalls. Additionally, software quality assurance must assure the absence of defects at the end of a project milestone. The usage of software assurance techniques can be minimized due the extensive usage of software quality development techniques. Thus, the total cost of software quality may be reduced.

In this paper, we examine three of the most known software development processes regarding practices for supporting software quality development and software quality assurance in the software development process itself. The findings can be used as basis for selecting a suitable process for high quality software development or for adopting running processes to well-working practices found in this study.

Section 2 gives an overview of the related work on software quality development and software quality assurance in software development process models. Section 3 gives a short introduction in the Rational Unified Process (RUP), Microsoft Solution Framework (MSF) and Extreme Programming (XP). Section 4 describes the principles and techniques, which we have derived

3-WoSQ '05, May 17, 2005, St Louis, Missouri, USA.

form analyzing the three models compared here. In Section 5, the three process models are evaluated based on the principles and techniques presented in Section 4. Finally, we summarize our results and suggest a de-facto standard for software quality support. This standard may be applied to the definition, adoption, or selection of software development models.

2. RELATED WORK

The term "software quality engineering" focuses on the processes involved in the development and establishment of software quality [10]. Software quality engineering includes software quality development and software quality assurance. Software quality development consists of requirements engineering, system and software design and implementation. Software quality assurance consists of software quality assurance, quality management and verification & validation. This work focuses on the establishment of software quality assurance by proposing a bottom-up approach, rather than on the aspects of quality development.

Software quality is achieved by three approaches: testing, static analysis and development approaches [11]. The integration of all three approaches is the most desirable approach. However, there is no consensus about the details of such an integrated framework.

A different categorization of approaches towards software quality regards four ways to establish software quality: software quality via better quality evaluation, better measurement, better processes, and better tools [12]. On the process level, the authors propose the usage of standards like Total Quality Management (TQM), ISO 9000, Capability Maturity Model (CMM) or Personal Software Process (PSP).

Large-scale quality models like Capability Maturity Model (CMM) or ISO 9001 tend to form a SQA in terms of a "process police" [1]. SQA takes care only that the process requirements are met but does not consider the quality of the process itself. Instead of SQA in terms of CMM or ISO 9001 a better solution is to embed quality evaluation in the development process.

RUP meets almost all requirements to reach CMM levels 2 and 3 [5]. XP is compatible to CMM as well [8]. Both development models require adaptations in order to completely fulfill CMM requirements. Specialized maturity models for XP are introduced combining CMM(I) with PSP [7].

Agile methods (like XP) have some practices that have both development functionality and QA ability [3]: create unit tests, continuous integration and acceptance testing.

3. THREE MAJOR SOFTWARE PROCESS MODELS

The software development processes discussed in this paper represent the most common and widely used process models throughout the industry. Although most companies use specific models, many of them are derived from RUP, MSF or XP.

RUP was developed based on findings of hundreds of different software development projects and the ideas of some of the most influencing people in the software engineering discipline. XP represents agile process models. MSF was developed based on the large experience Microsoft has gathered over the decades in engineering standard software products.

RUP, as the only process available as a product, has become state-of-the-practice in the industry over the last years and has been extended to support elements of XP.

Microsoft is pushing its Solutions Framework by including it in the upcoming release of the Visual Studio Team System and follows Rational's example by providing not only a process description but also a full product to support it.

3.1 Rational Unified Process (RUP)

The Rational Unified Process [4] is the most commonly used software development process in the industry. The main advantages are the support through Rational Software, which is constantly improving the process, the tightly coupled tool support and tool documentation, as well as Rational's support and mentoring in implementing the process. RUP delivers a well-structured framework, divided into phases and workflows, allowing easy navigation within the framework. Additional concepts of artifacts and workers are easy to understand and facilitate resource planning and work structuring. The process includes comprehensive quality assurance measures. Minimal standards as requirements engineering and iterative software development are included as well as testing, configuration management and collaboration with the customer during the overall process. Additional reviews form an important part of the process, continuously monitoring quality and progress.

3.2 Microsoft Solution Framework (MSF)

The Microsoft Solution Framework [6] provides deep insight into the software development practised at Microsoft. MSF represents Microsoft's attempt to spread its software development knowledge. The MSF is one of two complementary frameworks (besides Microsoft Operations Framework (MOF)), forming an overall solution for large software companies occupied with medium sized and large projects. However, the MSF may be applied alone. It does not depend on MOF, which allows an easy implementation in medium-sized companies. Focussing on delivering high quality software, the major principles of the framework include iteratively adding functionality and the "team-of-peers" approach without a dominating project leader. Both principles improve not only the quality of software but also the way it is built. Microsoft supplies a couple of templates for artefacts recommended in the MSF. Since the MSF does not include a huge set of recommended tools (which will change with the next release of Visual Studio) it can be easily integrated into existing development environments.

3.3 Extreme Programming (XP)

Extreme Programming [2] represents a completely new approach of developing software. For smaller projects, XP offers a good approach to achieve high software quality. The tight involvement of the customer, the main focus on testing and the approach to reduce design efforts support this goal. However, XP may lead to organizational problems when applied in large projects. For most projects, the tight integration of the customer during the development is hard to achieve.

4. FINDING COMMON SOFTWARE QUALITY DEVELOPMENT AND ASSURANCE PRACTICES

The best practices presented in this section, which support software quality development and software quality assurance, have been extracted applying an analytical bottom-up approach. This approach is based on the analysis of the chosen process models (RUP, MSF and XP) adding all principles and techniques defined in the processes to the criteria catalogue, which apply to the basic goal: support of software quality development and software quality assurance.

The criteria descriptions are based on the definitions of RUP, MSF and XP and on expertise derived from the local software engineering community. The bottom-up approach ensures that no criteria are selected which are not implemented in at least one industrial process model. Therefore the criteria catalogue will be rather industrially practicable than scientifically optimized.

This approach prevails empirical approaches (which are preferred more often). In order to be able to compare process models upon experiences of developers, it is necessary to find a sufficient number of developers with experiences with all three process models, which is practically impossible. Another possibility is to conduct controlled experiments, where the same group of developers (otherwise side effects resulting from different skills of the developers can not be prevented) conducts very similar projects with the different process models. Given a project with realistic size and complexity, this approach once again is practically impossible due the long duration of such an experiment.

4.1 Iterative software development

To establish higher software quality, a software development process has to use an iterative and incremental development approach. Iterative software development is characterized by refining and improving artifacts in several iteration cycles. Incremental software development means to start with a draft version of an artifact in the initial iteration and to extend and refine it through the following iterations. Iteration cycles include all development activities analysis, design, implementation, testing and finally deployment. Quality assurance can be applied more effectively during the overall process. By using an iterative approach, a process gains more flexibility in dealing with changing requirements or scope. The product releases of the product force early feedback from the customer and the stakeholders, which is vital for improving the overall quality of the software. However, iterative development must be supported by risk management and early involvement of the end users to achieve its full potential.

XP builds on a very strict iterative approach, demanding a daily build of all components. This limits the time needed to encounter errors and forces developers to fix a problem as soon as possible. Of course, incomplete components or single methods are excluded from the daily build. The work breakdown structure must consider these issues to allow an integration of smaller components every day. Using this approach requires a lot planning, but definitely enables high software quality.

4.2 Quality as an objective

A software development process needs to define quality as a major objective to improve overall software quality. Quality targets have to be defined and documented by involving the project team and the customer. This ensures that the quality goals become achievable and measurable. The MSF defines the qualitative targets of the project at the beginning and stresses the fulfilment of these goals as a major part of the project.

4.3 Continuously verification of quality

A set of procedures that document every change during the project is required to finally ensure quality. Not only project status reports, but also assessments of the current activities and possible changes are needed to identify problems as soon as possible. To support these procedures, every project needs a defined process to managed changes. All these actions can be implemented as meetings or as supporting workflows.

Continuously verifying quality includes extensive testing. Besides internal testing, external acceptance tests with the customer are needed too in order to verify that the product fulfils the needs and requirements of the customer. A software development process must therefore include a testing workflow throughout the complete process, including external tests with the end users to ensure high software quality.

4.4 Customer requirements

A software development process builds on clear structure and methodology to elicit and document customer requirements. It also has to integrate these requirements into the complete process. The elicitation of requirements is one of the most complex software engineering disciplines. The needs and wishes of the customer, who normally does not have a deep technical knowledge, have to be documented so that developers are able to build an application based on that information. Thus, it is necessary that the project team understands the customer and his business. Otherwise it is not possible to correctly implement the customer needs. A software development process has to focus on the requirements throughout the entire project. Eliciting and documenting the requirements at the beginning is not sufficient. The implementation of the requirements has to be traced during development.

Furthermore, the software development process has to ensure that not only the customer, but also the end users are involved in the requirements process. Success of the project strongly depends on these two groups: the first buying the product and the second using it. A software development process has to define procedures to train the end users to use the final product.

4.5 Architecture driven

In modern software development, the architecture of a system has a major impact on the overall quality of the product. One reason for this is the integration into existing systems and environments as a major part of today's software development. Re-use has become increasingly important due to increasing time and cost pressure. Using a well-designed architecture allows easy integration and re-use, so a software development process has to be architecture driven.

4.6 Focus on teams

A team has to be seen as a set of equal persons, who together are responsible for the quality and success of a project. When responsibility for failure can be assigned to a single person, the project success is not guaranteed anymore. Focussing on teamwork also improves motivation of the project members, as everyone is seen as an equally important part of the project. This finally leads to a high identification of team members with the product. It is obvious that motivated team members contribute to high quality, as they work more concentrated and conscientious. A software development process has to include a well-defined team structure, including an efficient task assignment and clear communication guidelines.

The MSF "team of peers" assigns equal value on each role and team member: the complete team has six goals to achieve and can be seen as a powerful implementation of the team focus principle. In most projects some goals have a higher importance than others. Managers decide alone or assign responsibility for failure to their team members. The MSF "team of peers" concept avoids these problems and enables a project to profit from the full capability of team work.

4.7 Pair programming

Pair programming is closely linked to the focus on teams, but was picked as another assessment criterion since it has been underrated for its contribution to high quality in the past.

XP demonstrates how two developers can complement each other rather than inhibiting each other. One developer implements the current method while the other is working on integration issues. This approach saves time, and minimizes the number of errors. Better solutions are more likely since two persons most likely have different perspectives of the same problem and therefore, complement each other in solving it.

4.8 Tailoring with restrictions

A software development process has to be defined and formalized in way regarding its application to a wide set of projects. A process that concentrates on a small or specialized set of projects ensures high quality in a particular environment. The major objective of a process is the application to many projects within a company without reducing its strengths. A process should be adaptable to a variety of projects based on its core elements.

To cover the complexity of typical projects a specified set of core elements has to be maintained. A high number of core elements does not necessarily improve the quality of the final product.

A software development process should therefore rely on core elements. Building on these core elements, the process should define practices for tailoring the process to the project type and project size.

4.9 Configuration and Change Management

A well-functioning Configuration and Change Management (CCM) is a major part of software quality assurance. The existence or non-existence of CCM has its greatest impact during software maintenance and evolution, yet basics for CCM (proper documentation, source code archives etc.) have to be set during the development process.

4.10 Risk management

A well-defined risk awareness and mitigation management form together an effective risk management and is a key factor in achieving high product quality. Risk management enables early risk mitigation and the possibility to act instead of to react to problems and risks. The MSF introduces its own model for risk management. It consequently emphasizes on discipline as an essential for keeping a project on track improving overall delivery quality. Risk management as an own discipline is a criterion for any software development process focusing on improvement of software product quality. Risk management as a part of daily meetings and other disciplines implies that identifying and managing risks becomes less important since the project team is occupied with other issues. The project becomes susceptible to risk factors.

5. EVALUATION OF THE QUALITY SUPPORT IN RUP, MSF AND XP

5.1 Software Quality Support in MSF

As shown in table 5.1, the MSF provides most practices for software quality development and software quality assurance. Quality is formally defined as an objective, a really unique feature that differences the MSF from the other processes. This quality objective is continuously verified to keep the project on track and contributes to a very effective quality assurance within the MSF. CCM and pair programming are not provided by the MSF, but can be easily integrated into the framework to accomplish a maximum of quality assurance. The MSF provides methods regarding requirements and architecture as both are key elements used during the overall process. Similar to RUP MSF starts at the beginning of a project, in the Envisioning phase, with requirements definition and management as well as drafting the first candidate architecture and prototypes. The focus on teams, defined as a team model ("team of peers") is another key strength of the framework. The approach is unique as it is the only teamwork approach that defines goals for each role. An internal client to the customer relationship enforces quality assurance.

The MSF can be tailored to fit specific project needs, but the core elements of the framework are stable. This ensures that every project using the framework as a unified set of procedures achieves a common quality standard. Finally risk management is represented within a own model in the framework, introducing the most comprehensive risk management in a software development process.

5.2 Software Quality Support in RUP

RUP performs nearly equally. Quality is not defined as an objective as in the MSF. RUP strongly verifies quality throughout the process. A well-defined review procedure is triggered at the end of iterations. This review verifies the achievement of all targets of the iteration and, discovers the reasons for failure, as they are an important input for the next iteration planning activities. Customer requirements are very well-handled within RUP, as the requirements discipline is heavily involved during Inception and Elaboration phases. To establish requirements traceability is one of the most important goals in RUP. The process is also very architecture driven. The first candidate architecture is already created during Inception phase.

Table 5.1: Quality Support Practices for RUP, XP and MSF

Criterion	MSF	RUP	XP
Iterative software development	✓	✓	✓
Quality as an objective	✓		
Continuously verification of quality	✓	✓	✓
Customer requirements	✓	✓	✓
Architecture driven	✓	✓	✓
Focus on teams	✓	✓	✓
Pair programming			✓
Tailoring with restrictions	✓	✓	
Configuratrion&Change Mgmt.		✓	
Risk management	✓		

RUP strongly focuses on teamwork, defining different roles and responsibilities. However the concept of "team of peers" extends RUPs teamwork approach. The tailoring approach of the Rational Unified Process is highly sophisticated. A specific tool is available within the process suite to support the tailoring of the process. The tailoring approach is defined within the process itself, leading to so-called "development case" artifact. The Rational developer network offers a lot of tailored RUP implementations for different technologies and project types. Nevertheless, the tailoring does not build on predefined core elements. Tailoring is not restricted. RUP defines many artefacts and causes a lot of overhead even if the process is tailored. Risk management is more stressed out than in XP, but cannot compare to the complete risk management process defined by the MSF. RUP completely fulfils the CCM criterion, as this is one of the biggest strengths of the process using Rational's own Unified Change Management approach. CCM is a supporting workflow that tracks changes and their impact throughout the project. RUP involves all major stakeholders of a project in the CCM workflow and Rational developed a large variety of supporting tools. Finally, the pair programming concept is not included, but can be easily integrated in to the process.

5.3 Software Quality Support in XP

Extreme programming performs poorly regarding quality assurance methods only. The reason for this is that XP is not as formalized as the other two processes assessed. Additionally, XP focuses on small development projects that are not necessarily demanding CCM or risk management streams. Therefore, XPs strengths apply only to smaller projects where the approach enables simple and fast development, but nevertheless on delivering high quality as it focuses on software quality development. For large development projects, it may be a reasonable approach to integrate some of the key elements of XP into one of the other two processes. XP continuously verifies quality equally as the other two processes assessed. The approach that the customer is on site and involved in the iteration planning process strengthens quality control from the customer site. The short iterations force the project team to develop functional releases at the end of each iteration to pass acceptance testing. Requirements, however, are not as important as in RUP. XP uses so-called "user stories" to capture the requirements but defines no ongoing process for requirements management. XP cannot be tailored, as it already consists of a minimal framework. The focus on teams together with the pair programming principle forms a very effective team approach, another key strength of Extreme programming.

6. CONCLUSION

The comparison of the software process models RUP, MSF and XP shows that all three models define practices, which support software quality development as well as software quality assurance. These practices allow developing quality software with no need for further software quality assurance (in most cases performed by independent QA departments).

MSF and RUP are the most elaborated software process model regarding software quality support. XP defines less software quality support practices than MSF and RUP. However, much software quality support is implicitly in XP principles, which affect the mindset of developers. XP is specialized for small and medium software projects, where its software quality support is as good as in MSF or RUP.

Each of the processes may be enhanced by including best practices defined in one of the other processes. Pair Programming, for example, can be easily included in the programming practice of MSF or RUP. The explicit definition of quality as a goal is easy to include in RUP and XP. There are already efforts of integrating the process models on a general level (e.g. [9]).

Looking at the comparison of the three selected processes, there are some practices for quality support, which can be found in all three processes. The practices, which are defined in all three processes, can be considered to form a de-facto standard.

Nielsen defines a significant de-facto standard for web sites if an element on a website can be found on 80% of all websites. Elements with 50%-<80% occurrence are not a significant de-facto standard, but strongly recommended.

According to wikipedia [13], a de-facto standard is a standard that is so dominant that everybody seems to follow it like an authorized standard.

Following these definitions, we define a de-facto standard for software process models as software process model elements, which are defined in the vast majority of public know software process models. Table 6.1 shows the practices, which have been considered to form a proposal for a de-facto standard for software quality support in software development process models.

Table 6.1: Outline of a de-facto Standard for Quality Support in Software Development Processes

Practice
Iterative software development
Continuously verification of quality
Customer requirements
Architecture driven
Focus on teams

We strongly recommend considering to include these five practices in every software development process used in a software company.

Furthermore we recommend considering following findings into the further development of international standards (e.g. ISO 12207):

- The inclusion of software quality development as a primary life cycle process.
- The inclusion of the five practices of the previously defined de-facto standard in the software quality development life cycle process.
- The further evaluation of the five practices form the criteria catalogue not included in the de-facto standard for inclusion into the software quality development life cycle process.
- The further evaluation of other software development processes (used in industry or described elsewhere) for best practices for software quality development for inclusion into the software quality development life cycle process.

7. REFERENCES

[1] Baker, E.B., *Which way, SQA?*. IEEE-Software, vol.18, no.1; Jan.-Feb. 2001; pp. 16-18.

[2] Beck, K.: *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

[3] Huo, M., Verner, J., Zhu, L., Babar, M. A.: *Software Quality and Agile Methods*. In *Proceedings of COMPSAC 04*, IEEE Computer Soc., 2004, pp. 520-25.

[4] Kruchten, Ph.: T*he Rational Unified Process: An Introduction*. Addison-Wesley, 2003.

[5] Manzoni, L.V.; Price, R.T.: *Identifying extensions required by RUP (rational unified process) to comply with CMM (capability maturity model) levels 2 and 3*. IEEE Transactions on Software Engineering, vol. 29 , no. 2 , IEEE, Feb. 2003, pp. 181-92.

[6] Microsoft Cooperation: *Microsoft Solutions Framework White Paper*. Microsoft Press, 1999.

[7] Nawrocki, J., Walter, B., and Wojciechowski, A.: *Toward maturity model for extreme programming*. In Proceedings *Euromicro Conference, 2001*. IEEE, 2001, pp. 233–9.

[8] Paulk, N.C: *Extreme programming from a CMM perspective*. IEEE Software, vol. 18, no. 6, IEEE, Nov.-Dec. 2001, pp. 19-26.

[9] Pollice, G.: *Using the Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming.* A Rational Software White Paper, Rational, 2001.

[10] Runeson, P., Isacsson, P.: *Software quality assurance-concepts and misconceptions*, In *Proceedings of the 24th EUROMICRO Conference*, IEEE Computer Soc, 1998, pp. 853-9.

[11] Osterweil, L.J.: *Improving the quality of software quality determination processes*, In *Proceedings of the IFIP TC2/WG2.5 Working Conference on Quality of Numerical Software. Assessment and Enhancement*, Chapman & Hall, London, 1997, pp. 90-105.

[12] Ward, W. A., and Venkataraman, B.: *Some Observsations on Software Quality*, In *Proceedings of the 37th annual Southeast regional conference* (CD-ROM), ACM, 1999, Article No. 2.

[13] Wikipedia: http://www.wikipedia.org.