Industrial Experiences of Developing a Model for Software Development Quality Gates

PASI OJALA
School of Business and Information Management
Oulu University of Applied Sciences
Hintanmutka 17, 90650 Oulu
FINLAND
pasiojala10@yahoo.com

Abstract: - The purposes to use quality gates in software development are many. Quite often companies see that the usage of quality gates improves their overall efficiency, effectiveness and output quality of software delivery chain. They also see that the usage of quality gates helps them to make things right at once by not skipping quality assurance actions.

This paper defines quality gate model for a software company. As well it shows that even quality gates might be in place they are not always followed because of a business reason. Companies can forget their well structured quality gate systems when business reason justifies it. The results of neglecting quality gates might lead to a situation where software asset output is not trusted anymore and quality is not known. In a longer run quality gate system seems to be as good as human being who is keeping it.

This paper discusses about the most typical software development quality gates in an industrial context. As well it gives reasoning why these gates are usable and defines general criteria for each of them. Paper notifies that even quality gates are in place, they are not useful if not followed.

The theoretical discussion in this paper is constructive and follows the constructive research method. Industrial experiments are explained using a case study method.

Key-Words: - Software quality, quality gates, software development

1 Introduction

There are many different activities in software product development process [20] . According to Humphrey [10], the paradigm of the software process proponents is that the quality of the software development process is closely related to the quality of the resulting software. In this paper this relationship between development process and resulting software has been used as a one starting point for building the proposed and discussed quality gating system. Therefore, the proposed and discussed gating system tries to influence not only to the quality of software development process but also to the quality of resulting software (see for example [16,17]

This paper also shares the idea of Krasner [15] who points out that "in a mature software organization, the following holds:

- Quality is defined and therefore predictable
- Costs and schedules are predictable and normally met
- Processes are defined and under statistical control".

Krasner's idea of defined and predictable quality is one main point for building the proposed quality criteria. This can be seen for example in the nature of gate keeping system, where each gate is having a gate keeper who is having the responsibility to predict the quality of software coming to his gate. It can also be seen that Krasner's idea of the need to define processes and control them is build inside the discussed gating system. This is because the quality gating system has been defined as a process and it is also controlled using different kinds of gating metrics.

Having the basic assumptions on the background this paper has also a long discussion on the background. Year by year the amount of software code seems to increase in several software products. The inevitable result of this seems to be also that software is including increasingly more errors as well. As competition is becoming increasingly more demanding and customers need to have more value in their software products, new methods for controlling the software quality are clearly needed.

This paper outlines that to be effective in the quality control a company needs to see the multidimensional nature of software development. Software development has many different kinds of activities and these activities differ from each other. As well as software development activities differ from each other, differ also the people working in a software company. They have different kind of ambitions and targets in their life and often they also achieve bonuses in a different way. Taking into account the differences in people and software development activities it seems justified to tell that also quality system needs to base on understanding the differences in software development environment.

Therefore first of all it seems justified that quality system sets criteria for individual software developers. Theoretically these criteria should be flexible and easy to follow for different kind of developers working with different kind of targets. Secondly in almost every software company there are several teams who deliver their code to a common codeline which combines all contributions together. Codeline gate criteria should be should be able to control the quality of different teams working together.

Thirdly every software company publishes software releases for internal testing use. These releases are extremely important for all technical teams. Using these releases all technical teams are able to get a common picture of how electronic product is working.

Fourthly every software company publishes releases for customer use. The quality of customer releases is extremely important as their quality is part of customer experience. If in a longer term customer sees this quality bad, it will have an effect to his buying decisions as well.

2 Model-Based Software Quality Process Improvement

There are several different aspects for controlling software quality [16]. One of the most discussed aspects is modeling. Typically, in modeling, “a model is an abstract representation of reality that excludes much of the world’s infinite detail. Generally, the purpose of a model is to reduce the complexity of understanding or interacting with a phenomenon by eliminating the detail that does not influence its relevant behavior. Therefore, a model reveals what its creator believes is important in understanding or predicting the phenomena modeled.” [19]

The purpose of this paper is to model a typical software quality gating process which has been seen important in improving software quality. According to Humprey & Feiler [11] and Osterweil [21], a process model is an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model and can be enacted by a human or machine.

The main parts of this paper are constructive. Constructive research constructs new reality by using research results which have in part been presented before. [13] The used research results in this paper cover for example earlier discussions of modeling software processes and using modeling as a basis for improving software quality.

This paper presents also experimental industrial experiences of developing quality gates model. In nature this part of the paper can be seen as a case study. Typically, a case study is an empirical inquiry that meets the following criteria: [25]

- It investigates a contemporary phenomenon within its real-life context, especially when
- The boundaries between phenomenon and context are not clearly evident.

In this study, all industrial experiences are explained with real-life data. During the data collection phase, the researcher read several documents and guaranteed the findings using several interviews with different people. Typically, in data collection, the researcher also organized teamwork sessions to find consensus for discussed topics. Since the criteria for each gate were not in every detail evident, the presented criteria are reported as they were defined in the company. This perhaps also helps in keeping to the real-life context and in drawing conclusions for each defined gate and criteria separately.

Methodologically, the applied research method can be understood also as case, because the inquiry: [25]

- Copes with the technically distinctive situation in which there will be many more variables of interest than data points, and as a result
- Relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result
- Benefits from the prior development of theoretical propositions to guide data collection and analysis.

According to Curtis et al. [6], “traditionally, the modeling of information systems has focused on analyzing data flows and transformations. This modeling accounted only for the organization’s data and that portion of its processes that interacted with data. Newer uses of information technology extend computer use beyond transaction processing into communication and coordination. Successfully integrating these systems into the enterprise often requires modeling even the manual organizational processes into which these systems intervene.” Furthermore, Curtis et al. [6] list the following three such applications:

- Business process re-engineering – the redesign of an organization’s business processes to make them more efficient.
- Coordination technology – an aid to managing dependencies among the agents within a business process; it also provides automated support for the most routinized component processes.
- Process-driven software development environments – an automated system for integrating the work of software-related management and staff; it provides embedded support for an orderly and defined software development process.
- In practice, research on software process modeling supports a wide range of objectives [14, 22] . Curtis et al. [6] list five basic uses for process models, ranging from understanding aids to automated execution support:
- Facilitate human understanding and communication – requires that a group be able to share a common representational format
- Support process improvement – requires a basis for defining and analyzing processes
- Support process management – requires a defined process against which actual project behaviors can be compared
- Automate process guidance – requires automated tools for manipulating process descriptions
- Automate execution support – requires a computational basis for controlling behavior within an automated environment.

Typically, in process modeling, “a model is an abstract representation of reality that excludes much of the world’s infinite detail. The purpose of a model is to reduce the complexity of understanding or interacting with a phenomenon by eliminating the detail that does not influence its relevant behavior. Therefore, a model reveals what its creator believes is important in understanding or predicting the phenomena modeled.” [6]

Humphrey & Feiler [11] have presented a foundational lexicon on which to build a conceptual framework for software process modeling and definition. They define a process as “set of partially ordered steps intended to reach a goal.” Any component of a process is a “process element”. A process step is “an atomic action of a process that has externally visible substructure.” Determining that a process element is a process step depends in part on whether any further decomposition of the element’s structure is needed to support the objectives of the process model.

According to Humprey & Feiler [11] and Osterweil [21], a process model is an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model and can be enacted by a human or machine. The levels of abstraction within the domain of software development range from the detailed process steps executed entirely on a machine, to the larger-grained human processes involved in executing a lifecycle phase, to the abstract stages of the lifecycle chosen for the product. Defined or not, the collection of all the process steps executed to develop a software system constitutes a software development process. These processes, however, do not constitute a software process model until they are represented in some medium. A process model to be performed by a human will be called a process script, while one to be enacted by a machine will be called a process program.

Model-based software process improvement involves the use of a model to guide the improvement of an organization’s processes. Historically, process improvement grew out of the quality management work of Deming [7], Crosby [4] and Juran [12], and it is still aimed at increasing the capability of work processes. Essentially, process capability is the inherent ability of a process to produce planned results. As the capability of the process increases, it becomes predictable and measurable, and the most significant causes of poor quality and productivity are controlled and eliminated. By steadily improving its process capability, the organization matures. Maturity improvement requires strong management support and a consistent long-term focus. In addition, it necessitates fundamental changes in the way managers do their jobs.[1]

One means of achieving this focus has been the use of capability-maturity models, such as CMM, CMMI or BOOTSTRAP. All these models provide a common set of process requirements that capture best practices and practical knowledge in a format that can be used to guide priorities. By using a model, organizations can modify or create processes using practices that have been proven to increase process capability. Organizations may also employ models to assess process capability for two purposes: to establish a baseline for improvement and to measure progress as improvement activities proceed [1].

Generally, model-based improvement begins with management commitment and assessment. The findings of this assessment, in turn, feed action plans. When these plans have been completed, further assessments are performed and the cycle continues. The goal is for the organization to mature so that it continuously monitors and improves its processes, consistently produces high-quality products, is agile within its marketplace, and adjusts quickly to customer needs [1].

Process modeling work is young, and the span of the research agenda is still undergoing formulation. According to Nielsen & Pries-Heje [18] “the use of a CMM or similar models as a basis for your improvement is an expression of faith in the maturity model paradigm rather than a strictly rational act.” Therefore, when using or combining the aforementioned models to other models it is important to understand the models’ underlying assumptions.

Typically, process maturity models focus solely on one factor in a company – processes – and neglect other aspects. They assume that good processes alone will lead to the goal of better software development. Furthermore, they assume that what is best for one kind of company is also good for another kind, by assuming that you can generalize best practices across the software industry and that the more best practices in your company, the better your maturity [18]. Typically, these models are also based on two underlying philosophies, which influence their assumptions as well. The first of these philosophies is Total Quality Management (TQM), which is a “management philosophy embracing all activities through which the needs and expectations are satisfied in the most efficient and cost effective way by maximizing the potential of all employees in a continuing drive for improvement” [2] . The second philosophy is step-by-step, one-small-step-at-a-time learning, such as that practiced in the Japanese Kaizen strategy [5].

3 Software Development Gates in Company A

In the background of this paper there has been a long discussion of which kind of gates are needed for controlling quality and which are not [16,18,19,24]. Answering for the question which kind of gates are needed in software development and which kind of gates are not is not easy. It depends highly on the developed software, development environment and software development process in question [16, 18, 22, 23].

In this paper the software development process in Company A bases on the idea that software developer designs and writes code. Every day he delivers his code through developer gate to daily build developed by his team together. On weekly basis or when software teams have implemented a full subsystem they deliver their contribution to the mastercodeline through mastercodeline gate which combines all builds together. Bi-weekly the code is gathered together from mastercodeline as releases using release gate. All releases passing release gate are used for development purposes in product programs. Final releases which are going to customer products go through also a branch gate which measures the final maturity and quality of software.

The defined quality gate system in Company A is managed by a program manager. Each gate has also a special gate owner who develops his gate further and gives support for gate users if needed. In practice, all software packages coming to any gate need to pass gate keepers check which is done using a criteria list defined for the gate in question.

This paper defines quality gate as a special milestone in a software project. It sees that quality gates are located between different software development phases. Furthermore this paper sees that each quality gate includes a check of the documents of the previous phase and includes special requirements on these documents.

Based on their software development process definition quality gate development program in Company A defined following four quality gates for their software development usage:

- Developer Gate
- Mastercodeline Gate
- Release Gate
- Branch Gate

These gates were seen important because they are all control points for delivering code to new participants of company’s software development process. Company A saw that by using these gates they would increase visibility to the most problematic areas of their software development.

Following chapters present how the quality gate system is working in Company A. The purpose has been to explain who is responsible of each gate and who are delivering code to the each gate in question. Some emphasis has been used also for illustrating what kind of criteria each gate includes and how each criteria should be interpreted. As all gates are clearly different to each other their implementation has also differed from each other. Due to these reasons some initiative has also been taken for illustrating how each criteria was reviewed and approved.

As industrial experience paper the purpose of this paper is to open up discussion for developing quality gate system for industry use. Therefore clearly more studies are needed for finding out better criteria for each gate than presented here. In the future studies it is also clearly more necessary to find more empirical support for proposed criteria.

4 Software Developer Gate

In Company A there are several software development teams. All development teams have several software developers who design, write and module test individually their software code. Company A sees that their designers have specialized to build different kinds of software. Some designers are specialists in building user interfaces and some designers have specialized in building embedded software with more direct need to understand the nature of hardware as well.

Company A has confirmed that it uses a lot of effort to give to its designers a possibility to build that kind of software for which they have a strong interest. Company A organizes development discussions twice in a year with each designer and tries to agree together with each designer the content of personal development plan. This is done because Company A sees that when it is having motivated designers for designing software code it can also publish a more mature gating system for controlling the quality of each designer's software code. The basic underlying idea for building developer gate has been to understand the need for personal development and give more space for each designer to design their code in question. In all together Company A sees that it has creative and independent software designers who are able to take bigger responsibility in code designing than what normally seems to happen in average software companies.

Company A saw that the most natural point to control developer quality is to control the contribution of software developer who delivers his code to the daily software build. For this purpose Company A defined a developer gate which needed to be passed daily by each software developer. In practice all contributions of each software developer were checked daily using a developer gate criteria list. The persons responsible of checking were the subsystem owners of the code and each developer in question.

From software process point of view developer gate in Company A includes all those steps that developer needs to do when committing a task to the version control tool (for example Synergy). The developed criteria for the developer gate included following criteria:

1. Check that developer uses a copy of the latest available version of the target environment.
2. Check that developer runs a private build and that his /her code must compile without errors and warnings.
3. Changes or error fixes done by developer should not cause any additional errors or warnings.
4. Make sure that code dependencies are known, taken into the development environment and changes are communicated to all relevant persons.
5. Check that new code and changes are unit tested and configurability tested according to guidelines.
6. Test coverage is measured and new code and changes are tested in reference hardware
7. Peer code review is done for the major changes
8. Static analysis has been run before review and high warnings should be analysed and removed
9. Code is committed to the version control tool.
10. There are zero memory leaks with memory allocation failures on in hardware.
11. Complexity analysis is executed and results are analyzed.

All developer gate criteria were reviewed in Company A. The participants of the review included several developers, subsystem owners and architects. Generally the review was easy. The longest discussion was held around the use of latest available target environment. There were opinions which saw that it is not possible to follow these criteria literally. Even the latest available target environment always exists it is not available always for example for subcontractors as they are not working in the same premises. This comment was written down and software development manager took an action point to start discussions how latest available target environments could have been offered also for subcontractors.

Company A organized several trainings of developer gate criteria interpretation for all software developers, architects and subsystem owners. Software developers saw defined criteria usable and they were satisfied that they had finally agreed practices for daily builds. They considered it to be a relief that they now had common principles for everyday work. Subsystem owners and architects also saw that developer gate system helps them to control the general quality of code developed in different teams.

Software development project managers saw that the biggest challenge for them is to try to make schedule planning for which their teams would always be committed. They saw this difficult because Company A was not used to situation where software development commitment was driving software contribution in every software project. Software development project managers were also skeptical that even plans would be realistic, is top management letting them to follow them even they had been approved beforehand.

5 Mastercodeline Gate

In Company A there are several software development teams. All development teams have several software developers who design, write and module test individually their software code.

Historically, software integration had faced a lot of challenges in Company A. One reason for these challenges had been the way how developers had been implementing their code. Rather often the code had been implemented in isolation from each other which had caused a lot of incompatibility and visibility problems between software modules. From Company A point of view the result of isolation had often been long integration times and huge amount of integration errors.

The purpose of mastercodeline gate in Company A was to make sure that software code passes needed criteria before new implementation can be brought to the mastercodeline. Furthermore it was seen to ensure that software modules have been made ready enough so that they do not cause a lot of feedback (in the form of errors) from the later phases of software development process. In practice mastercodeline gate was assuring that software modules have been build up using best practices and efficient communication before integration.

Company A defined that the gate keeper of mastercodeline gate is defined by the software development team. The possible gate keepers in their organization were therefore for example subsystem owners, chief engineers or architects. This was because these people were understood to have wide enough understanding of the developed software to understand integration related problems as well.

The discussion of developing mastercodeline gate was rather long. This discussion revealed all typical problems of Company A's software development. It also showed that typically many integration related problems are caused because people see that their responsibility has ended when they have delivered their code. Many designers also notified that as their personal compensation systems are based on their individual contribution, historically there has not been strong enough interest for looking integration related problems with other designers.

Keeping in mind the development discussions related to the mastercodeline gate Company A saw that following gate criteria would hel them to control and improve the code coming to the mastercodeline for integration purposes:

1. Software asset is compliant with build tool and build is done
2. Codeline policy rules are followed
3. Configuration policy rules are followed
4. Intellectual patent rights issues are closed and documented
5. Unit and module testing coverage are measured
6. Code complexity is measured
7. All features are done
8. Functionality, performance and regression are measured
9. There are 0 critical errors

Especially, software subsystem owners, chief engineers and architects saw the gate criteria usable. Some of them highlighted that if there are continuous errors in code they do not know how long they should send back this kind of code as the danger is that it starts to influence other developer team's code too. The solution for the discussed problem was that common sense needs to be followed in this kind of situations. However, criteria were seen to increase the visibility to the most problematic areas which was seen to give better possibility for correcting problems as well.

Even in practice the mastercodeline gate criteria was developed for controlling the software integration quality it was often neglected to the business reason. Business reason was developed for being an exception for situation where the entire product would be in danger not coming to the markets in time. However, in practice several teams did not dedicate themselves for their deadlines and their code was not good enough in quality when coming to the mastercodeline. The result of this was that mastercodeline jammed of different kinds of code contributions which were not good enough in quality.

Mastercodeline would have worked better in Company A if there would have been more visibility to the each contribution coming to the gate. Later on Company A saw that it needs to develop more preventive actions for controlling the software quality before software arrives to the gate as it is too late to influence to the code when it already has arrived there.

6 Release Gate

The purpose of releasing software in Company A was to ensure that product programs will get integrated software for their product development purposes. In Company A release gate was considered to ensure the quality of software releases and provide high quality development environment for developer teams.

According to release gate Company A saw that software release is not ready unless gate is passed. Company A defined release gate as biweekly implemented cyclic gate. It saw that if release gate is not passed in planned time then the release in Company A is cancelled. Company A defined following release gate criteria:

1. No build breaks
2. Smoke tests passed
3. Maturity criteria for main and other configurations available
4. BAT (basic acceptance test) test results available
5. All needed language variants created
6. R&D environments shared to relevant places
7. Release done according to approved release template
8. Release note done according to approved template.

The responsible release gate keeper in Company A was an integration manager. He highlighted in gate criteria review that developer gate and mastercodeline gate are more important when assuring good code quality than release gate. However, he saw that the release gate is also important because it gives visibility of software quality to the product programs. Other software developers saw this important because there had been problems in communicating software maturity and quality related issues to other product development areas. Generally, approved release templates and notes were considered to be good initiatives for handling these problems.

The approval procedure of release gate criteria was rather easy. The main reason for this was perhaps that there were not so many people who worked as integration managers. For a smaller group of people it is easier to come to a final conclusion than for a bigger amount of people.

7 Branch Gate

The purpose of branch gate in Company A was to give visibility to the software maturity for every branch off from the mastercodeline. The Branch gate was a set of criteria that was used to guard and understand the quality and maturity of the software content prior to any software branch was given for final use (customer releases) to a product program.

Company A defined following criteria for branch gate:

1. Fully understand the maturity of each feature
2. Fully understand the system maturity
3. Localization testing run rates
4. SW Maturity Regression
5. Full understanding of all problems with fix plans are in place
6. Reliability results available
7. Software application certification status is available
8. First round of pre-certification testing is done
9. First round of pre-certification testing is done
10. Product requirement lists are checked
11. Plans for mandatory features are approved for productization activities

The responsible gate keeper in branch gate was a branching manager. He saw in criteria review that even branch gate might not be the most important from software quality point of view, it surely gives a good visibility to the final maturity of code given to product programs. Therefore, he saw that defined criteria and gate are in place as they help to communicate about the situation with product lines.

In addition it was mentioned that branched quality is the quality which goes also to the customer. Therefore direct customer feedback should be discussed in contrast with branched software quality. If customers are happy for the quality branch criteria works but if customer is not happy it should have an influence to new branch gate criteria.

8 Experiences and Collected Opinions of Quality Gate Development and Implementation

During the quality gate definition and implementation program all software development personnel in Company A were highly motivated and happy for the purpose of the quality gate program. They saw that gate system is very logical and helps to tackle software development problems in early phase of the development. It was also a common expectation that the implemented gate system will increase efficiency and visibility of the software development.

After Company A had been running quality gate system for six months all gate owners, gate keepers and several developers were interviewed. Based on the interview results it was possible to make general conclusions. Firstly, all interviewees told that gating system is not working as effectively as it could. During six months they had started to develop exception handling policies for each gate for letting bad quality to pass the gates. Especially mastercodeline gate seemed to be in chaos because it was receiving several builds which were far away from planned quality. The reason for this was that software development teams were not making realistic schedules for their code development. They were too often too optimistic regarding to the schedules and the result was that that the delivered code was not fulfilling the mastercodeline gate criteria.

Based on the interviews software development teams told that even they have responsibility of making realistic schedules they did try to do so in the beginning. However, due to business reasons they did not ever have planned and approved time for making their code. Product programs and their management seemed to be stronger in Company A and for these reasons they were constantly pushing software deadlines tighter. Finally the result was that software development teams did not have enough time to make sure that delivered code is good in quality.

Integration manager and branching manager told that in their opinion their gates worked very well. They saw that their gates give a good visibility to the code and even it should be send back to the developers they were forced to approve bad quality because of a business reason coming from product programs and top management. In their opinion Company A was back in a situation where it was before starting quality gate development program. They had huge amount of errors in their code and when something was fixed for the next release another new problem popped up which had not been known earlier. Finally, also unstable software development environments were causing more and more errors.

The personnel had presented constructive criticism of the quality gate implementation program to the top management of Company A. Top management saw that the product schedules (time to market) are more important than software quality.

The results of this implementation projects support the findings of Hammers & Schmitt [17] when say that adapting quality gates effectively is challenging. The top management of Company A had prioritized time to market for products so important that it was possible to neglect software quality for that reason. In the end the personnel of Company A started to create exception handling policies for criteria which was earlier seen very important for software quality.

The motivation of software development personnel dropped significantly. Software development personnel saw that they are not getting enough support for their work. They notified that it is amazing that even software development takes almost 80% of the development resources in the company it is still not possible to make reasonable schedules for software development. In their opinion other development areas were always planned more realistically and their planning also started clearly earlier. So the conclusion was that the planning of software development always started too late and it was done separately from other product planning with the result that software was always waited to be ready as the last thing.

The results of this study also support the statement that top management's support is crucial for software quality and process improvement initiatives. However, in product business it is not always inevitable that this support would be present. There are many other development areas which are competing inside the company of the management support.

Finally, it was easy to see that in Company A quality related problems started already in developer and mastercodeline gates. Even Company A tried to encourage designers for having more agile working methods too much agility just seemed to lead to a chaos. For an observer it looked that schedules were not made to be followed and the lack of preventive quality controlling actions just increased the chaos in each gate.

9 Discussion and Conclusion

The purpose of this paper was to model a typical software quality gating process. This was done because software quality improvement was seen important and industrial experiences seemed to require a solid model for building quality gates in an industrial context.

The main theoretical parts of this paper were constructive. Constructive research, in this paper, constructed new reality by using research results which had in part been presented before. These most important starting results of the research included discussions of modeling, software process development and improvement and quality gating.

This paper saw that the purpose of a quality gate model was to reduce the complexity of understanding or interacting with a phenomenon by eliminating the detail that does not influence its relevant behavior. Therefore, a quality gate model was seen to reveal what its creator believed to be important in understanding or predicting the phenomena modeled."

Furthermore this paper saw that a quality gate process is a "set of partially ordered steps intended to reach a goal." Any component of a quality gate process included therefore a "process element". In addition a quality gate process step was seen to be "an atomic action of a quality gate process that had externally visible substructure." Determining that a quality gate process element had a process step depended in part on whether any further decomposition of the element's structure was needed to support the objectives of the quality gate process model.

The experimental industrial experiences of this paper were discussed using a case research. This was seen important so that it would be possible to have a rich and thorough understanding of examined phenomena in an industrial context.

Based on the experimental discussion in Company A it was seen that the quality gates needed to be synchronized with the existing software development process. They needed also to be constantly developed further by gate owners and the software content coming to the gate needed to be checked by a gatekeeper.

The gate implementation project in Company A was run by a project manager who had the overall responsibility of approving each gate criteria. Each separate gate was developed further in development teams lead by gate owners who facilitated development discussions.

After theoretical discussions in Company A software engineers defined and implemented four different quality gates. These gates were considered to ensure the efficiency, effectiveness and output quality of the software. The defined quality gates included developer gate, mastercodeline gate, release gate and branch gate.

The purpose of developer gate was to ensure the daily quality of the developer's code. This gate was defined because it was seen important that many quality problems are corrected in the earliest possible phase. Mastercodeline gate was dedicated for ensuring that the code made by several developer teams is good in quality. It was justified because Company A saw that there are several integration problems caused by bad visibility and inefficient communication and efficient use of mastercodeline gate would minimize them.

As developer gate and mastercodeline gate were dedicated more for the purposes of software developer and teams, Company A saw that they have a need to provide visibility of their software quality also for the product lines. For these purposes Company A defined the release gate for controlling the quality of biweekly releases given to product lines for development purposes. As Company A saw that final customers and the releases going to final products are extremely important, they decided to define a branch gate for controlling the software quality in final products as well.

Even quality gate system was seen usable in Company A it faced several problems. Biggest problem was that Company A did not seem to believe on it after taking it into use. Company A seemed to think that the extra time used for controlling software quality is not paying back as more predictable and efficient software development. Therefore Company A started to develop several exception handling policies so that bad quality was finally accepted to go through the gating process. One major reason justifying bad quality was the general business reason and business schedules.

However, as software products are becoming more and more complex and more and more people are involved in making them it would be a good idea to continue researching quality gating as one tool for improving software quality. More experimental research on implementing different kinds of gate criteria in different kinds of companies and software development processes is clearly needed for having more complete research results.

It should also be notified that process maturity models to which this gating system also based on focus solely on one factor in a company – processes – and neglect other aspects. They assume that good processes alone will lead to the goal of better software development with better quality. In real life this assumption might necessarily not come true what also happened in Company A. Furthermore, we should also understand that what is best for one kind of company is not necessarily best for all kinds of companies and therefore modeling a quality gate process for any company bases on understanding its business needs.

References:

[1] Ahern DM, Clouse A & Turner R (2001) CMMI Distilled. A Practical Introduction to Integrated Process Improvement. New Jersey, Addison-Wesley.
[2] British Standard (1991) British Standard 4778: Part 2.
[3] Brown, N. (1996): Industrial strength management strategies. IEEE Software, volume 13, issue 4, July 19, pp. 94-103.
[4] Crosby PB (1979) Quality is Free. New York, McRaw-Hill.
[5] Colenso M (2000) Kaizen Strategies for Successful Organizational Change: Enabling Evolution and Revolution within the Organization. New York, Prentice Hall.
[6] Curtis B, Kellner M & Over J (1992) Process Modeling. Communication of the ACM Sept 35(9): 75-90.
[7] Deming WE (1986) Out of Crisis. Cambridge, MA, MIT: Center for Advanced Engineering Study.
[8] Gill, N.S. (2005): Factors affecting effective software quality management revisited. ACM SIGSOFT Software Engineering, volume 30, issue 2, March, pp. 1-4.
[9] Hammers, C, Schmitt, R (2008): Governing the process chain of product development with an enhanced Quality Gate approach. CIRP Journal of

Manufacturing Science and Technology, volume 1, Issue 3, pp. 206-211.

[10] Humprey WS (1989) Managing the Software Process, Reading (MA), Addison-Wesley.

[11] Humprey WS & Feiler PH (1992) Software Process Development and Enactment: Concepts and definitions. Tech. Rep. SEI-92-TR-4. Software Engineering Institute. Pittsburgh.

[12] Juran JM (1988) Juran on Planning for Quality. New York, MacMillan.

[13] Järvinen, P. (2001) On research Methods. Opinpajan kirja, Tampere, Finland.

[14] Kellner M (1989) Software Process Modeling: Value and Experience. SEI techn. Rev. Software Engineering Institute. Pittsburgh, Carnegie Mellon University: 22-54.

[15] Krasner H (1999) The Payoff for Software Process Improvement: What it is and How to get it. In: E1 Emam K & Madhavji NH (eds) Elements of Software process assessment and improvement. Los Alamitos (CA), IEEE Computer Society.

[16] Mahnic, V (2008) Assessing Scrum-based Software Development Process Measurement from COBIT Perspective. 2nd WSEAS Int. Conf on COMPUTER ENGINEERING and APPLICATIONS (CEA'08) Acapulco, Mexico, January 25-27, 2008, pp. 23-28, pp. 589-594.

[17] Mahnic, V & Zabkar, N (2008) Measurement repository for Scrum-based software development process. 2nd WSEAS Int. Conf on COMPUTER ENGINEERING and APPLICATIONS (CEA'08) Acapulco, Mexico, January 25-27, 2008, pp. 23-28.

[18] Nielsen PA & Pries-Heje J (2002) A framework for Selecting an Assessment Strategy. On Improving Software Organizations. From Principles to Practice. New Jersey, Addison-Wesley: 185-198.

[18] Ojala, P., Implementing a Value-Based Approach to Software assessment and Improvement. Doctoral dissertation. University of Oulu, 2006.

[19] Ojala, P., (2008) Experiences of Implementing a Value-Based Approach. WSEAS Transactions on Information Science and applications, issue 1, vol 5, January 2008.pp. 385-395.

[20] Ojala, P (2008) Experiences of Implementing a Value-Based Approach to Software Process and Product Assessment. 2nd WSEAS Int. Conf on COMPUTER ENGINEERING and APPLICATIONS (CEA'08) Acapulco, Mexico, January 25-27, 2008.

[21] Osterweil L (1997) Software Processes are Software too. Proceedings of the Ninth International Conference on Software Engineering. Washington D.C. IEEE Computer Society May 17-23: 540-548.

[22] Riddle W (1989) Session Summary: Opening Session. Proceedings of the Fourth International Software Process Workshop. Brighton, UK. IEEE Computer Society: 5-10.

[22] Salger, F, Engels, G, Hofmann, A. (2009): Inspection effectiveness for different quality attributes of software requirement specifications: An Industrial Case Study. WoSQ'09, Vancouver, Canada, May 16, pp. 15-21.

[23] Salger, F, Sauer, S, Engels, G. (2009): Integrated specification and quality assurance for large business information systems. Proceedings of the 2nd annual conference on India software engineering, Pune, India, February 23–26, pp. 129-130.

[24] Wilson, P. (2006): Quality gates in use-case driven development. Proceedings of the international workshop on software quality, Shanghai, China, pp. 33-38.

[25] Yin RK (1994) Case Study Research. Design and Methods. Second Edition. Thousand Oaks (CA), Sage Publications.