Industrial Experiences of Developing Quality Gates for Software Development Process

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.

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

1 Introduction

There are several different aspects for controlling software quality [3]. One widely discussed aspect is modeling. Typically, in 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.” [2]

The purpose of this paper is to model a typical software quality gating process. According to Humphreys & Feiler (1992) and Osterweil (1987), 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.

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 [1,8,9]. 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 [1,9].
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.

2 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 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.

3 Mastercodeline Gate
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 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.

The developed criteria for mastercodeline gate were:

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

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.

4 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 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.

5 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.

6 Discussion of the Quality Gate Implementation in Company A

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 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 (2008) 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.

7 Conclusion
The purpose of this paper was to discuss about the most common software development quality gates. Based on the discussion in Company A it was seen that quality gates need to be inline with the existing software development process. They need to be constantly developed further by gate owners and the software content coming to the gate needs to be checked by a gatekeeper.

In Company A software engineers defined 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.

However, 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. 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 when bad quality was accepted and justified. One major reason justifying bad quality was the general business reason and decided business schedules.

References: