First Time Right in AXE using One track and Stream Line Development

Lovre Hribar, Stanko Sapunar, Ante Burilović
Ericsson Nikola Tesla d.d. (1,3), HEP d.d. (2)
Croatia, Split
E-mail: lovre.hribar@ericsson.com

Abstract: **Software development project and the different phases within the project are constantly improved over the decade with the various methods, approaches and the best practice in the software design development. Stream line development and the One track are method aiming at raising the quality of software product releases by minimising number of faults at the time of delivery. Component software quality has a major influence in development project lead-time and cost. The resulting design delivery to verification phase will be more predictable quality software with shorter lead-time and Time-To-Market (TTM).**

1. INTRODUCTION

It is always about quality and cost. Or even better, it is always about ratio between quality and cost. As the quality level of the final product is set at the beginning of the project, a large number of faults can result in project delays and cost overruns. The number of faults in a large software project has a significant impact on project performances. To keep project performances development teams require fast and accurate feedback on the quality at the early stages of the design cycle.

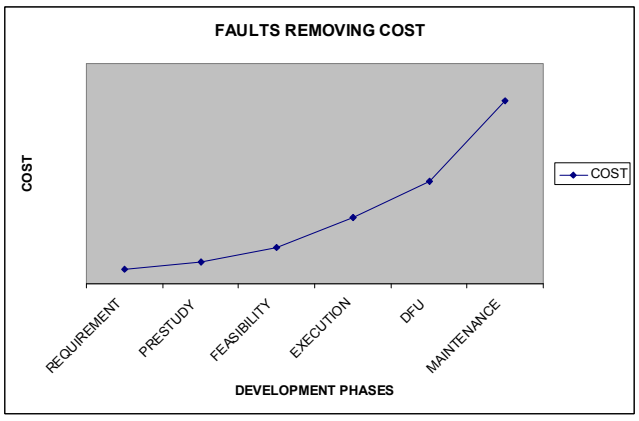


Figure 1. Faults removing cost

IEEE Standard 12207 defines QA this way: "The quality assurance process is a process for providing adequate assurance that the software products and processes in the product life cycle conform to their specific requirements and adhere to their established plans" [14].

The ANSI/IEEE definitions for quality control is measuring the quality of a product and the Quality Assurance measures the quality of processes used to create a quality product. The ISO/IEC 9126 product quality standard [15] is used to categorize the metrics.

ISO 9126 (ISO 1991) is called software product evaluation: quality characteristics and guidelines for their use. In this standard, software quality is defined to be "the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs". Software quality is decomposed into six factors: functionality, reliability, efficiency, usability, maintainability, and portability.

Feedback on software component quality is something which we need very early in the project. It is very hard to predict and determine the quality of the software components at the beginning of the project. In addition, the software quality prediction has a significant role in easing the maintenance of software. In short, the prediction is helpful in software development, testing and maintenance activities.

Predictions can either be made on the basis of historical data collected during implementation of same or similar projects, or it can be made using the design metrics collected during design phase [16].

If the software component is inherited from the previous project, it is very important to have information about software component quality from the previous project (number of faults, type of faults, maintainability…) and information from the field (customer sites, exchanges in live operation…) [9]. All this information together with preventive and corrective actions will help project to keep all commitments and to keep time to market with the first time right quality.

Also, if the same software component is used in similar applications, impact on quality is visible and can be predicted better with the more accuracy. During the different development stages the quality of the software component must be tracked and visible. There is a number of the applied techniques but most important is the one which are tracking the number of faults for the software component [17].

But the quality is not for free. The quality does cost. So the key is to focus on evaluating the cost of quality and return on quality [10].

This article deals with implementation of the One track and Stream line development in the Ericsson Nikola Tesla R&D within TSS (Trunk Signaling Subsystem) part of the AXE. It describes actual application of the discussed principles for the current running projects in Wireline and Wireless application. Some of the applications are new, and some are further development of the existing applications. The project duration is also different, some of the projects were running for 2 years, but some of the projects are only 6 months last. The importance of cost of quality is stressed,

together with other factors that influence quality of a software product at the production side.

1.1 First Time Right

First Time Right (FTR) is an *initiative* or *procedure* mainly established for Quality Assurance. In simple words Quality Assurance is the planned and systematic activities implemented within the quality system and demonstrated as needed to provide adequate confidence that an entity will fulfill requirements for quality [18]. It is about balancing methodology, leadership, and technology. It is about taking into account human factors as well as technological ones. FTR is not international standard like ISO [3] so any particular company defines its own scope of FTR activities. The aim is to deliver features with the expected functionalities, within the expected times and with a quality level which minimizes (ideally nullifies-fault free product) and makes predictable the rework caused by faults found after function test (i.e. the faults slipped through from previous verification phases and system test faults). Defects are found as early as possible which cuts the costs, increases customers and sponsors satisfaction and increases the organization's capability. The initiative/procedure is based on quantitative criteria (mandatory documents, best practices, specific measurements, fault slip through measurement) applied in a controlled and structured way, enabling continuous improvements by propagating learned lessons among projects [11]. FTR focuses on designing for quality (preventive action) instead of solving problems (corrective actions).

1.2 Measurement and how it is defined

According to Fenton and Pfleeger [12], software is a physical entity which can be measured by its quality, since physical objects are easily measurable. Also according to these authors, the software quality measurement operation should be easy. However, software quality measurement presents many difficulties, and this is because the concepts of effort, functionality and complexity related to software measurements have neither agreed upon boundaries nor precise definitions. Software Measurement is still emerging as a field of knowledge and, most often, traditional quality criteria of measurement methods, such as repeatability, reproducibility, accuracy and convertibility, have not even been investigated by software measurement method [13].

FTR initiative applies to the selected project features and project phases. For each selected feature and phase, a Quality Index (QI) is selected, associated to specific criteria taking into account the project context and the specific feature characteristics. This means a quality strategy customized per project/feature. The overall initiative is subject to a continuous improvement, based on the feedback from its application to avoid the big bang [1].

2. ONE TRACK

One Track (OT) is mainly a development strategy that needs a set of system changes as input. The system changes are transformed from a set of high level features and system requirements. One Track describes a software factory that frequently delivers a new component tested system version. The system is developed in small steps [4].

The One Track development pattern describes a way of developing software with a high degree of control and feedback by using continuous planning and small integration steps. To find an optimal way of implementing the new system changes a project anatomy is used to understand the dependencies between different changes. Intention is to integrate the system frequently, typically every week, to avoid the big bang and obtain early feedback.

The system is tested or verified (depending on the stage of the development work) by the next levels of test (e.g. Function Test, System Test and Network Test).

The objectives for effective testing are to locate and repair faults in the software, to identify error prone software, and to complete testing on schedule. The purpose of System and Network testing is to find errors which normally result from unanticipated interactions between subsystems and components. It is also concerned with validating that the overall system provides functions specified in the requirements [8].

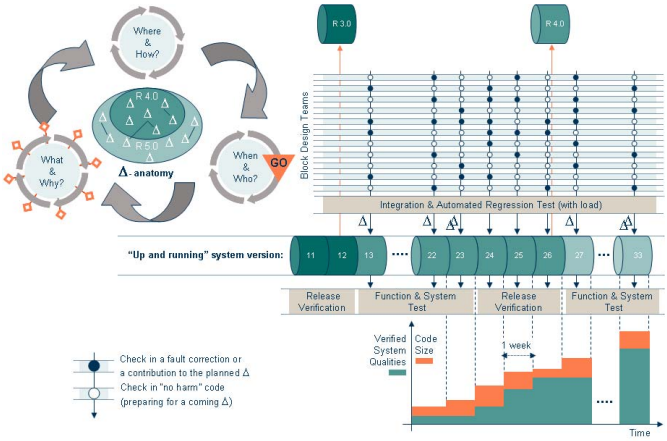


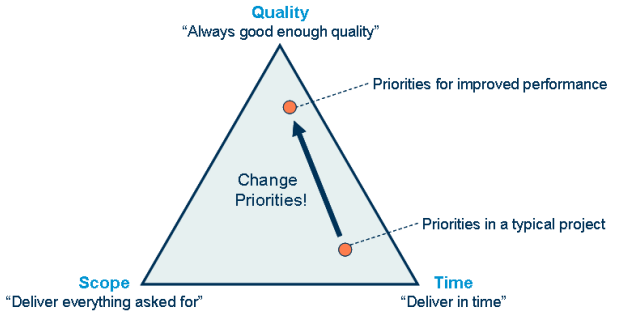
Figure 2. One Track - the whole picture

The six main principles [5] in One Track are:

- Prioritize quality higher than new functionality
- Develop the system in small steps
- Continuously plan system integration
- Use only one track of versions
- Do extensive regression test before verification
- Parallelize test and verification activities

2.1 Prioritize quality higher than new functionality

The enabler for One Track is high software quality. To facilitate high software quality the correction of faults must (in most cases) have higher priority than new functionality.



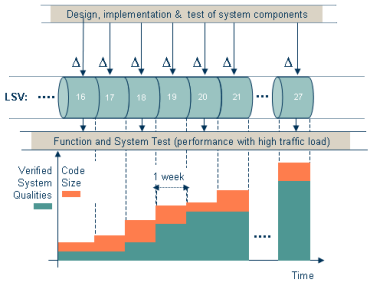
The quality level of the design base determines development speed, just "working faster" is wishful thinking

Figure 3. "More features faster" starts with high quality

To be able to run One Track the quality of the design base must be good enough. If there are any faults in the system, we have to find them early, correct them quickly and make sure that we do not have to correct the fault several times. But, in the end we really must stop making the faults in the first place.

2.2 Develop the system in small steps

In One Track the main idea is to split the solution into scenarios and further down to components. Regression test is performed to make assure that the small component changes are working.



Each new step …

- .. results in a new "up and running" component tested system version, ready for function and system test
- .. is made within a week

Figure 4. Develop the system in small steps

We do not want the "big bang" function delivery in the end.

Why small steps? Shorten the feedback loop "frequent and relevant feedback improves quality".

Small steps provide frequent and indisputable milestone since true progress is visible at the end of each step.

The components that are affected by the changes are tested and verified on a low level by the designer using basic test or multi basic test. The components must not break what was working before and they must not break the integration.

During the night the components are integrated and a light regression test are run. In the morning there will be feedback from the light regression test. Any faults found should be corrected immediately before the planned changes and corrections are being implemented.

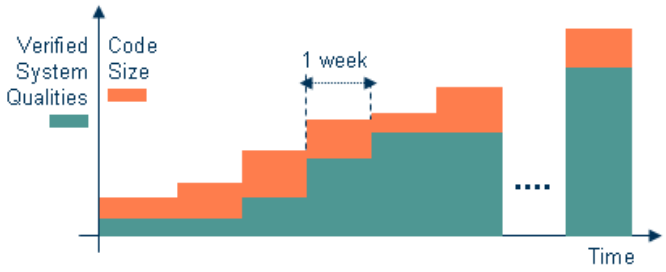


Figure 5. Stepwise system growth including non-verified system qualities

As a part of the design team, you have to deal with the changing design base on a daily basis. Traditional quality assurance tools (code reviews, component/basic test, quality doors, SQR, utilization of trace and debug tools etc.) should be used by the component teams. Teams can be organized as a cross-functional team or a component design team depending on the design environment, product complexity and the product organization. The important thing is to have synchronization points at the weekly build for all teams.

2.3 Continuously plan system integration

The integration plan is a living document that need daily care and be communicated to all involved. System designers and testers need to be involved in this work.

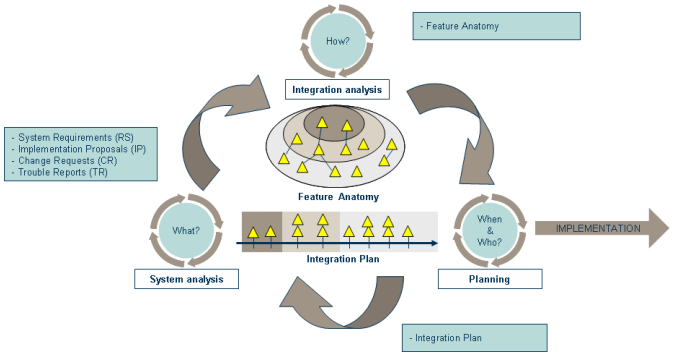


Figure 6. The planning cycle

The center of the "planning cycle" and a very important tool for understanding the relationships between the changes

is the project anatomy (also called delta anatomy and feature anatomy). The dependencies constrain the order in which the changes can be done. Changes without dependencies, for example, can be developed in parallel with each other.

2.4 Use only one track of versions

The important thing is that we have a straight sequence of versions that builds on each other and these are the only versions that are released and delivered to a customer. Customers do need to accept frequent upgrades and new versions with new functionalities and new system qualities often. One advantage by having only one sequence of versions in maintenance is that we do not need to maintain several different versions of the system and we do not need to correct faults more than one time in the code. One Track allows implementation of coming features earlier as long as specified content to integration every week is fulfilled.

2.5 Do extensive regression test before verification

The first quality check is done by the design teams doing basic test on component level. The One Track method uses a second level of tests, done by the integration and regression test team. They ensure that we do not break what worked in the previous LSV and that legacy functionality is kept.

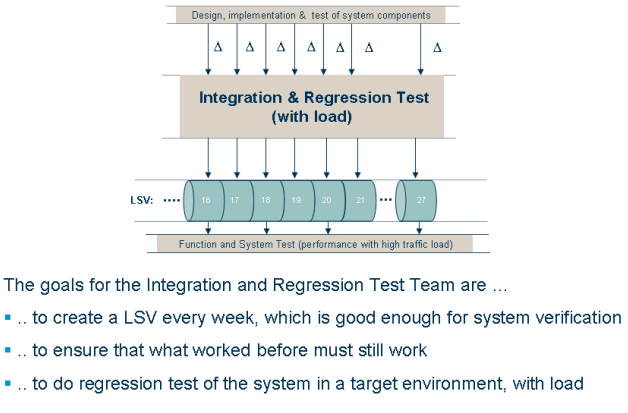


Figure 7. The integration and regression test team

After delivery to integration and regression test team the design teams continue implementing new changes. If faults are found, they are corrected immediately by the design team. A new system version with corrections is available typically the next day.

The advantages of having a centralized integration and regression test team are that we only need a few target test environments and that the test team members can be specialized in this area. The disadvantage is that we get another hand-over from design to test.

By getting the regression test team to work in cooperation with design we can minimize this hand-over.

2.6 Parallelize test and verification activities

One very important enabler for high product quality is to get feedback from test as early as possible (parallelize the verification activities). The second reason is to decrease the verification lead time.

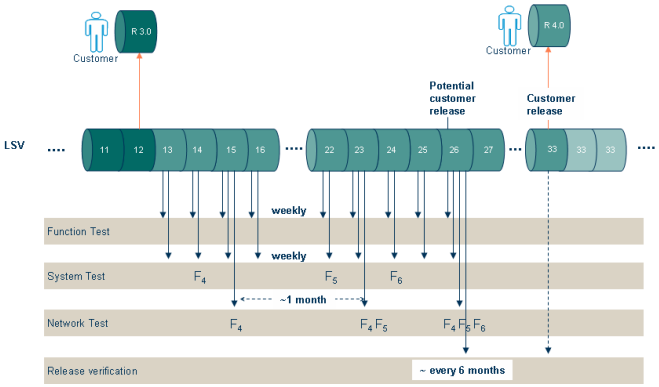


Figure 8. Parallel test and verification activities

A risk with starting system verification very early is that the system might not be mature enough for verification. In this case, let testers test instead of verify to start with, to get rid of as many bugs as possible. It is much easier and cheaper to verify a system without the bugs.

3. STREAM LINE DEVELOPMENT

Streamline Development is a model for co-operation between development units and product management to achieve development efficiency with customer focus. Streamline Development is built on three principles:

- Scope for started development is not changed.
- decoupling of development decisions and release decisions
- frequent and continuous system growth with verified features or capabilities

Streamline development defines the interface and certain operations between release strategy and development strategy but does not in itself say anything about which release strategy is preferred or what development strategy to use. Streamline development represents development of features in a never-ending stream of iterations.

3.1 Scope for started development is not changed

The pre-study scope is larger than the feasibility scope and what is really developed in execution is in the end even less. This is called the scope-out strategy. Apart from waste of resources in earlier phases another drawback with scope-out is that features that should be highly prioritized are managed the same way as lower priority features.

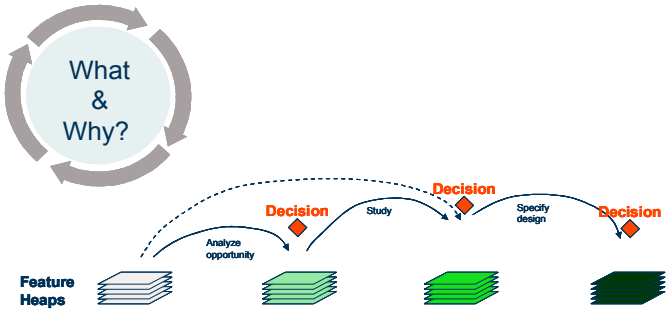


Figure 9. Illustration of impact of decisions on feature level taken according to highest priority from a business perspective

Instead of scope-out Streamline Development talks about scope-in. Decision is taken when there is a business case for it and activities for that feature are started independent from other features.

The early decisions, from a release date perspective, open up for later decisions as well and a possibility to deliver features on short time notice. Each decision represents an investment and hence, should be based upon a feature's priority according to a business case.

The way to break down large features is to define ambition levels to be able to sell a new feature as soon as possible. This is one of the main principles in Streamline Development.

3.2 Decoupling release decisions from development decision

The traditional way of deciding upon releases and release contents is to tightly couple those decisions. Decoupling release decisions from development decisions means that one series of decisions handles what should be part of the product, from a strategic or customer perspective.

The advantage with decoupled decisions is that features can be developed continuously from best knowledge of business cases at any time. Also the follow up of release costs and feature costs is simplified as assignments are connected to each decision.

The consequences of not adhering to this principle is a great risk that a majority of the scope-in decisions are taken very close to the release date and we get a big bang situation in development and release verification. As the development is more a continuous operation than connected to a single project release. It could therefore be managed from a multi release view.

That is, the only existing projects are release projects coupled to taking the product from to the market.

3.3 Frequent and continuous system growth with verified features or capabilities, LSV concept

In order to be able to release on short time notice with confidence knowing that we release a system with good quality. Using development strategies that end up in Big Bang integrations every four or six months will destroy the benefits from Streamline Development.

The release scope has to be settled well in advance to manage the release verification in time for general availability. The final call for new features is then moved back in time reducing flexibility for product management again. The solution is frequent and continuous system growth of verified features or capabilities. New functionality is tested on component or system level and that it is secured that legacy still works.

3.3 Benefits

In Streamline Development the development organization is continuous. The resources often don't have to be prioritized between rivaling projects and costly administrations are reduced.

As development can start as soon as there is an approved business case for a feature the development resources are always used and for highly prioritized tasks.

Product Management does not have to order features two years ahead but rather take the decision six to nine months before the market window occurs. Combined with a sales model that offers smaller but more frequent releases and even single features it makes Ericsson more flexible to customer needs.

All this together, the release aspect and consequences of higher speed to market demands higher capability of responsiveness of the delivery and deployment organization.

4. HUMAN FACTOR

When moving to a new technology, we want to understand the new issues and factors that influence our projects. We want to understand people as a main contributors to the project changes. If we are changing project routine this will have influence on the people who are accustom on the project stages and a traditional way of work. If the project was bad planned or if the quality of the software component is not good enough, originally planned project activities and project commitments will not be kept. To overcome this situation project managers offen requires from organization management more people to be added into the project. Adding more people to a late project or to the project with the software component quality problems makes it only later with questionable quality [7]. By adding more people to a running project, a project manager influences a 'human factor' of the project – productivity. One new person on a project will keep about two experienced project members off their work for the next 2 to 3 weeks. Not because the person is not smart, but just because he/she has to learn and

understand. Moreover, new person if it is not experienced enough will also introduce some quality issues to the software component.

Unfortunately, the project has to be finished in time anyway, so project managers often reach for their toolkit – fire fight equipment. Fire fighting is a human reaction to a 'bad' process and 'bad' planning. This reaction causes stress, frustration, defects that have to be repaired, so less time - more fire. Time pressures induced by development schedule constraints are an often-cited source of quality problems in technological systems. Problems arise when developers, feeling that they are under pressure to meet task deadlines, take shortcuts in dealing with unanticipated complications [19].

4.1 Stress

A certain amount of stress is necessary for us in order to get things done. During stressful projects the time for recovery is often not available. This in turn increases stress. When under high stress, one starts to make mistakes. These mistakes cause more work, more stress, and more mistakes. Quality will suffer, and often quality control is abandoned in order to 'save' the project. Although a person under high stress works faster initially, he/she will be less efficient in thinking, finding solutions, etc. More stuff is produced, but it often is not the right stuff. It looks that more is done, but suddenly it becomes clear that the implementation of the real functionality is still behind.

How can stress be made predictable? Next to indicating the complexity, and re-use level for specific parts of a system, you can also give an indication of expected stress and how well the team can cope with the stress. Stress is a risk factor.

When stress hits the project, and it will of course be in a way that we not notice, project control is often more difficult [2]. The quality is jeopardize and the "First time right" can be turned on "Last time right".

5. CONCLUSION

Software quality and improvement of software quality is the theme from the beginning of the software development history. And this is a very stressful process where the design team is under constantly pressure to keep the software quality within the settled goals from the beginning of the project.

Quality of software product and software components are constantly measured and interpreted on different ways, depends on the perspective (design, test, management, customer perspective…). In order to establish meaningful software quality measures, software quality measures methods and directions for particular environments/situations, the all involved persons within the different teams should have approximately the same empirical relational system of software quality. Measurement theory should be applied in order to know what we measure and how to interpret the results. This way we may be able to build better prediction systems for quality related attributes, such as customer satisfaction or reliability.

The results on the case study projects which are using traditional way of working without 3 weeks delivery cycles has shown that the projects will not benefit much by adding more people in the late project phases. As a result a number of the faults were received to late in the project and the cost for the faults removal was significant in compare to the planning one.

On the other hand, the best results were achieved in the projects which adopted 3 weeks delivery cycle and where the project activities are planned with incorporated time for quality improvement as a result of the FST analysis and SQR measurement. As a result, a negligible number of the faults are received on that software components and applications in general.

FTR as an initiative in this paper with One Track strategy and Stream Line Development model presented and evaluated in different projects are proven in gaining the quality as the customer expect, without unnecessary rework and maintenance cost. Measurement of software quality during every single stage in the project was very usefull in prediction of faults in the next stage of the project and diferent preventive and corrective actions performed within the projects was proven to keep project commitments and to keep time to market with increased quality of the software componets.

REFERENCES

[1] H.D. Mills: Top-Down Programming in Large Systems. In Debugging Techniques in Large Systems. Ed. R. Ruskin, Englewood Cliffs, NJ: Prentice Hall, 1971.

[2] Kris Oosting, "The human factor: predictable or unpredictable", ESCOM-SCOPE 2000, Combining the 11th ESCOM (European Software Control and Metrics Conference, with the 3th SCOPE (Software Certification Programme in Europe) Conference, 18-20 April 2000, Munich, Germany

[3] ISO, International Organization for Standardization, http://www.iso.org/iso/home.htm

[4] ***, "R-State Handling Cook-Book for Software Products", Internal Ericsson documentation, Sweden 2006

[5] ***, "The One Track method – main principles", Internal Ericsson documentation, Sweden 2006

[6] ***, "One Track Good Practice description", Internal Ericsson documentation, Sweden 2006

[7] Brooks, R. :"The Mythical Man-Month", 7th Printing, 1997, Addison Wesley

[8] Hyatt, L., Rosenberg, L., "A Software Quality Model and Metrics for Identifying project Risks and Assessing Software Quality", (th Annual Software technology Conference, UT, 4/96

[9] Lovre Hribar, Ante Burilović, Darko Huljenić, "Implementation of the Software Quality Ranks method in the legacy product development environment", 10.

International Conference on Telecommunications, ConTEL 2009

[10] Snadra A., Slaughter Donald E., Harter and Mayuram S. Krishnan, "Evaluating the cost of Software Quality", Communication of ACM, August 1998/Vol. 41, No.8.

[11] Kan, Stephen H., "Metrics and Models in Software Quality Engineering", 2nd edition, Addison Wesley Longman, Boston, MA, 2003

[12] Fenton, N.E., Pfleeger, S.L.,"Software Metrics: A Rigorous Approach. Chapman & Hall, 1996.

[13] Basili, V R, Selby, R W., "Calculation and use of an environment's characteristic software metric set." In: Proceedings of the 8th Int. Conf. on Software Engineering, London, 386–393, 1985.

[14] IEEE standard for a software quality metrics methodology, IEEE Std 1061-1998, Dec 1998.

[15] ISO/IEC 9126-1:2001 --- Software engineering -- Product quality -- Part 1: Quality model, 2001.

[16] Lovre Hribar, "Usage of Weibull and other models for software faults prediction in AXE", 2008. International Conference on Software, Telecommunications & Computer Networks, SoftCOM 2008

[17] Lovre Hribar, Sanja Bogovac, Zdenko Marinčić, "Implementation of Fault Slip Through in Design Phase of the Project", 31. International Conference, MIPRO 2008.

[18] Asad Ur Rehman, "Quality Cost Analysis"

[19] Robert D. Austin, "The Effects of Time Pressure on Quality in Software Development: An Agency Model", Information Systems Research, 2001 INFORMS Vol. 12, No. 2, June 2001, pp. 195–207