Implementation of Fault Slip Through in Design Phase of the Project

Lovre Hribar
Ericsson Nikola Tesla, d.d.
Poljicka 39, Split, Croatia
Phone: +38521435914, E-mail: Lovre.Hribar@ericsson.com

Abstract: **The Fault Slip Through (FST) process is a way to secure early, correct and cost effective fault detection through analysis of issued trouble reports. A way to secure that faults are found in the right phase; Right phase means the most cost-efficient phase. It is a (fault based) method for identifying improvement opportunities. A method of measuring how many percent of faults inserted in one phase of development are detected and corrected in a later phase of testing and/or operation. The process brings many advantages such as early correct and cost effective fault detection, enables fault slippage measurements, avoiding doing the same mistakes over and over again, less redundant testing and closer test coordination, improved quality and less stopping faults, shorter lead times and improved delivery precision and finally less resource consumption in the latest phases of the projects (cost efficiency). This article has shown that the FST from the process/method point of view should be very efficient in very large projects with a high complex solution.**

I. INTRODUCTION

The number of faults in a large software project has a significant impact on project performances and hence is an input to project planning. 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. For project planning purposes and for quality management, an important measure is the trend of *TR inflow* in the project – i.e. how many trouble reports (faults) are reported in a particular time [1].

The *TR inflow* is a measure which is eminent on the project level and depends on the sub-projects (or work package) testing phase. *TR inflow* is one of the most important variables to monitor in large scale software projects. It provides the management with a possibility of identifying whether a given project is not going to meet the set goals and to adjust the project plan, if needed. It allows also the organization to optimize resource allocation for projects – e.g. when there is a large *TR inflow*, the organization needs to provide additional person-hours to keep the project on track (e.g. by ordering overtime).

Large software projects have very different dynamics to small projects; the number of factors that affect the project is much larger than for small and medium software projects. Large software projects also tend to develop complex software-hardware systems. The current practices for large software projects at Ericsson Nikola Tesla rely heavily on expert estimations, which are rather time consuming; in particular the experts use Case Based Reasoning (CBR, (Maiden and Sutcliffe, 1993) while constructing the predictions for fault inflow – by identifying similarities and differences between projects the experts construct the predictions.

FST approach results in a method which is simple and which has high-cost efficiency (e.g. the costs of miss-predictions are smaller to the costs of building and maintaining more accurate models), which could be seen as a trade-off between prediction accuracy and costs of predicting.

II. QUALITY

Everyone agrees that quality is important, but few agree on what quality is. Kitchenham (1989) notes that "quality is hard to define, impossible to measure, easy to recognize". Gilles states that quality is "transparent when presented, but easily recognized in its absence"[3].

For instance, when someone tells me that “good enough is not good enough,” I remember the stakeholder and critical purpose perspectives and translate that apparently paradoxical statement into something I can question, such as “good enough for you is not good enough for me” or “good enough to survive is not good enough to succeed.” Then the dialogue becomes one of examining whose values matter or what purpose we are really trying to achieve. Confusion about Good Enough Software is understandable and forgivable, since no one has published an actual detailed description of what Good Enough means. Jones seems to define it as the practice of deliberately leaving bugs in the code so as to shorten the schedule. I’ve heard other people define it as providing the minimum quality that you can get away with. [2].

Garvin concluded that "quality is a complex and multifaceted concept." Garvin described quality from five different perspectives: the transcendental view, that sees quality as something that can be recognized, but not defined; the user view, which sees quality as fitness for the user's purpose; the manufacturers view, which sees quality as conformance to specification; the product view, which sees quality as tied to inherent characteristics of the product;

and the value-based view, which sees quality as dependent on what a customer is willing to pay for it [4].

After a decade of performing process improvement, rework for organization's software development projects was dramatically reduced all over the world. Economically, the concept arises of right sizing the Quality Assurance (QA) function with respect to the needs of the customer(s) or the quality goals of the producer organization. There is a cost for quality; it is not free.

At a minimum, QA functions should be sized sufficiently to satisfy the customer's requirement for product quality. The customer wants the product at a low price with no flaws. The producer wants to *make money*, be competitive, and increase business — QA is a cost to be trimmed. Clearly, it is impossible to simultaneously satisfy these parties. In competitive areas (multiple producers of the same product), the marketplace decides the product price. From the producer's perspective, QA needs to be efficient and rework minimized. Minimizing the cost of QA and rework makes the product more competitively priced and maximizes profit.

A good production process will satisfy nearly all of the customer's requirements without QA, i.e., quality is built in, not inspected in. The customer, reasonably, cannot expect a perfect product. However, customers can mitigate their risk of purchasing poor products by testing performance and inspecting physical details during the production process and prior to accepting delivery. His investment in product testing and inspection is an expense, and a portion of the product price is attributable to the customer- generated rework. Minimizing the expenditure for QA yet meeting the customer's quality requirement is not a simple matter [5].

In the software development process at Ericsson Nikola Tesla, while quality is seen from all of the above views, the most important view is that of the project managers. The situation in which the software is developed and used heavily influences the project manager's view.

To accomplish the task, project managers must have indicators for improving the processes and achieving the needed level of quality. Thus the project manager's view of software quality is pragmatic and relatively simple - high quality software is software that "works well enough" to serve its intended function and is software that is "available when needed" to perform that function.

The criterion of "Works Well Enough" includes satisfaction of functional, performance, and interface requirements as well as the satisfaction of typical "ility" requirements such as reliability, maintainability, reusability and correctness.

The criterion of "Available When Needed" is dependent upon the software's role in the system. Thus the project manager is interested in a "pragmatic" quality model and metrics program, one that will help in the successful development and operation of a specific system. Any model and associated metrics program that is to be funded by a project manager must be aimed at satisfaction of the two criteria and at the identification of risks that they will not be met.

III. FAULT SLIP THROUGH DEFINITION

FST is the basic measure used by the method suggested in this paper. FST is similar to phase containment measurements where faults should be found in the same phase as they were introduced [8]. The essence of such approaches is to analyze when faults are inserted and found and from that determine which faults are in-phase and out-of-phase [8]. The time between when the fault was inserted and found is commonly referred to as 'fault latency'.

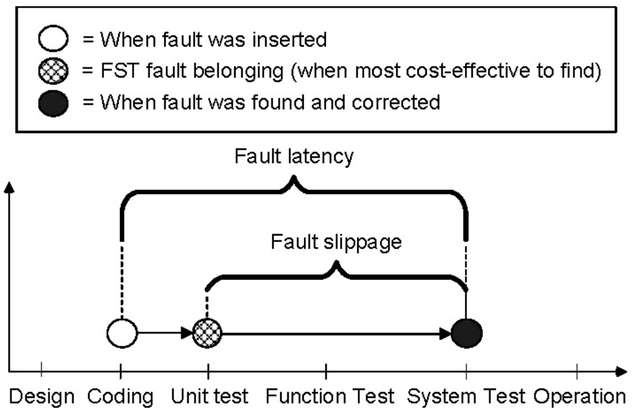

○ = When fault was inserted
▨ = FST fault belonging (when most cost-effective to find)
● = When fault was found and corrected
Fault latency
Fault slippage
Design
Coding
Unit test
Function Test
System Test
Operation

Fig. 1. Fault latency versus FST

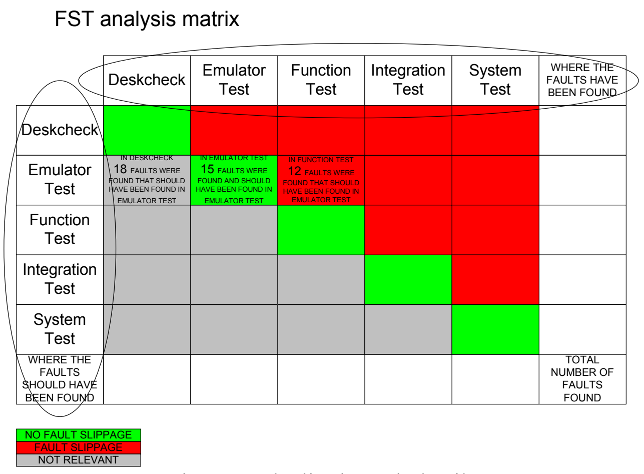
Since most faults are inserted during early development phases, e.g. requirements elicitation, design, and implementation, the fault latency measure is not good at evaluating the quality of the verification process. Further, if for example a test activity is improved, it is not possible to use measurements based on when faults were inserted to evaluate the result since only later phases are affected by the improvement. Instead, the FST concept considered more appropriate because the primary purpose of measuring FST is to make sure that the test process finds the right faults in the right phase, i.e. in most cases early.

Figure 1 visualizes the difference between fault latency and FST. When measuring FST, the norm for what is considered 'right' is the test strategy of the organization. That is, if the test strategy states that certain types of tests are to be performed at certain levels, the FST measure determines to what extent the test process adheres to this test strategy. This means that all faults that are found later than when supposed are considered slips [6].

Faults not identified (slips) during the different steps of the verification phase are subject to detection by the customer during his product testing and inspection. The customer's perception of product quality is created largely from the faults he identifies.

To gain repeat business or good references for new business, Ericsson strives to minimize the faults that propagate, or leak, through his production and quality processes.

One of the most common ways that Ericsson uses FST data is for evaluating the degree of FST to a verification phase.


FST analysis matrix

	Deskcheck	Emulator Test	Function Test	Integration Test	System Test	WHERE THE FAULTS HAVE BEEN FOUND
Deskcheck	🟩	🟥	🟥	🟥	🟥	
Emulator Test	▨ IN DESKCHECK 18 FAULTS WERE FOUND THAT SHOULD HAVE BEEN FOUND IN EMULATOR TEST	🟩 IN EMULATOR TEST 15 FAULTS WERE FOUND AND SHOULD HAVE BEEN FOUND IN EMULATOR TEST	🟥 IN FUNCTION TEST 12 FAULTS WERE FOUND THAT SHOULD HAVE BEEN FOUND IN EMULATOR TEST	🟥	🟥	
Function Test	▨	▨	🟩	🟥	🟥	
Integration Test	▨	▨	▨	🟩	🟥	
System Test	▨	▨	▨	▨	🟩	
WHERE THE FAULTS SHOULD HAVE BEEN FOUND						TOTAL NUMBER OF FAULTS FOUND

🟩 NO FAULT SLIPPAGE
🟥 FAULT SLIPPAGE
▨ NOT RELEVANT

Fig. 2. Fault slip through details

To create trend lines from the FST data, the most easily alternative is to use a TR web based solution which makes it possible to get real-time updated results, i.e. assuming that it is able to fetch the measurement data from the fault database. The appropriate frequency of measurement points, e.g. weekly or monthly, depends on the length of the monitored projects.

The result from such measurement points can be seen in Figure 3. There are two necessary prerequisites to apply this method:

1. The fault reports should include a field stating which phase each fault should have been found in, so that the statistics can be fetched from the fault database regularly. If this for any reason is not possible, an alternative is to have regular follow-up meetings, e.g. weekly, where the FST value of each reported fault is determined.

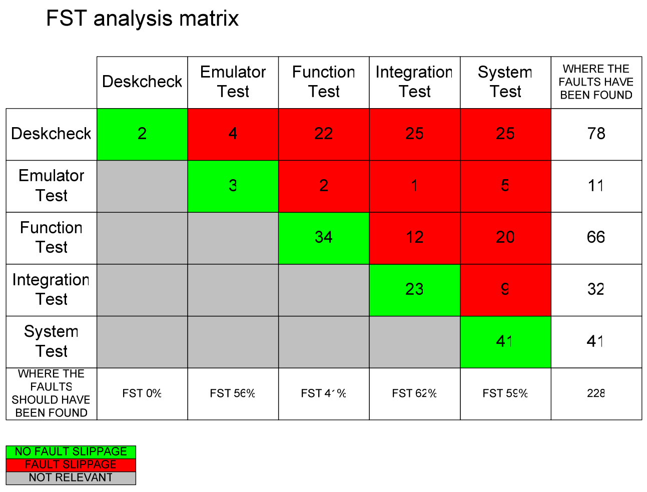
2. If regular follow-up of the FST trends are going to be useful, goals that the trends can be compared against should be set. In our experience, the preferred input for such a goal is a baseline value obtained from a previously finished project. Based on this value, the goal should be set after what differences are expected in the project to study, e.g. if a planned process improvement aims at reducing the FST to a certain degree, the goal value should be specified accordingly.

It is also possible to set a goal without the baseline value. However, it will then be hard to know what an appropriate goal is. With these overall prerequisites met, it is then possible to monitor the process quality during the verification stages of projects.

IV. FST MEASUREMENT

The measurement was conducted at a software development department at Ericsson, which develops some software products on its main site. The projects develop software to be included in new releases of existing products that are in full operation as parts of operators' fixed networks. A typical project such as the one studied in this paper lasts about 1.5 year and has on more than 70 participants. The projects are performed according to an in-house developed incremental development process. Besides inspections of documents during design, the products are verified in five steps: Desk Check, Emulator Test, Function Test, Integration Test and System Test. According to the test strategy of the organization, the faults that belong to different phases are in this paper divided as follows:

Desk Check (DC): Faults found during code review.
Emulator Test (ET): Faults found during unit tests of a component.
Function Test (FT): Faults found when testing the features of the system, e.g. faults in user interfaces and protocols.
Integration Test (IT): Faults found during primary component integration, e.g. installation and component interaction faults.
System Test (ST): Faults found when integrating with external systems and when testing non-functional requirements.


FST analysis matrix

	Deskcheck	Emulator Test	Function Test	Integration Test	System Test	WHERE THE FAULTS HAVE BEEN FOUND
Deskcheck	🟩 2	🟥 4	🟥 22	🟥 25	🟥 25	78
Emulator Test	▨	🟩 3	🟥 2	🟥 1	🟥 5	11
Function Test	▨	▨	🟩 34	🟥 12	🟥 20	66
Integration Test	▨	▨	▨	🟩 23	🟥 9	32
System Test	▨	▨	▨	▨	🟩 41	41
WHERE THE FAULTS SHOULD HAVE BEEN FOUND	FST 0%	FST 56%	FST 41%	FST 62%	FST 59%	228

🟩 NO FAULT SLIPPAGE
🟥 FAULT SLIPPAGE
▨ NOT RELEVANT

Fig. 3. Fault slip through measurement

Figure 3 presents when the faults were found in the project in relation to the same measurement points as in Figure 2.
A fault trigger determines which type of test activity that should identify a fault (originates from IBM "Orthogonal Defect Classification"). For example, robustness faults should belong to the trigger 'robustness' (no matter which test activity that found them). The purpose with fault trigger classification is to determine which test activities in which phases that are fault prone, that is to evaluate the test process. Can for example be used to identify improvement areas that will reduce FST. If many of the faults found in ST belong to a trigger category that FT should cover, this provides feedback to where FT need to be improved. If a trigger activity contains few faults during ST but live tests finds many of that type, it indicates that the corresponding test activity is performed insufficiently in ST [7].

During the analysis, some of the reported faults were excluded because they turned out not to be real faults. Additionally, requirements faults were not reported in the fault reporting system. Instead, they were handled separately as change requests. Each test level verifies a varying number of deliveries from the design department depending on the number of feature increments and the number of required bug-fix deliveries. Further, to save lead-time, the

verification levels of the increments are performed partly in parallel, e.g. FT is not completed when ST starts. This introduces a risk for more fault slippages but if ST knows what in the delivery is tested and not in FT, the ability to start ST early on the parts of a feature that have been function tested saves more lead-time than what the additional cost of FST is worth. This is a major reason why the optimal FST goal rarely is zero. How to follow-up the measurement? Basic measurement formula:
FST (to phase X) = No. faults found in phase X that should have been found in an earlier phase / No. faults found in phase X.

V. ANALYSIS OF FST MEASUREMENT RESULTS

Applying the proposed method on the project in this paper, we identified some patterns on how the FST trend changes during a verification level. But why the FST levels are so different in the beginning and why do they even out when half of the time has passed? To answer these questions, the FST data was compared against when the faults where found to see if that affected the FST distribution. The fault data is due to confidentiality reasons only presented as percent of the total number of faults.

The result of the FST to each phase measurement is presented on Figure 4.

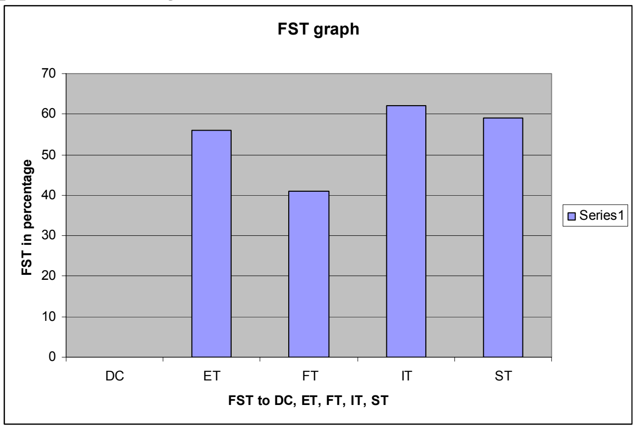


Fig. 4. Fault slip through to each phase

The result of the FST from each phase measurement is presented on Figure 5.

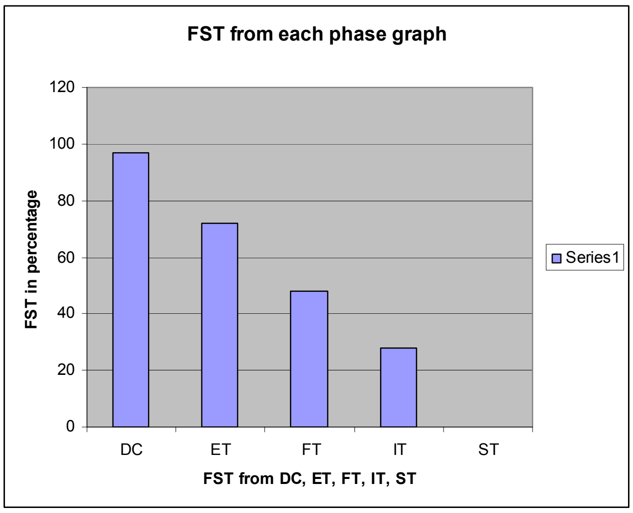


Fig. 5. Fault slip through from each phase

The first important observation from Figures 4 and 5 is that the faults were distributed rather evenly over time. However, the most apparent exception is FT of the project where the percent faults after a third of the project drop down on the lowest level of 41 percent. Relating this to the FST curve explained why the FST level could decrease from 100 to about 41 percent so fast, i.e. too few faults were found in the beginning to have a reliable FST value. On the other hand, when a significant part of the faults were found early, the curve stabilized early as well, e.g. IT of project. When cross-checking the project, one can see that the FST trend reaches a stable level when about 30 percent of the faults have been found.

The major implication of this is that as long as only a minor part of the faults have been found, the FST values might still change a lot. This implication however causes a problem since the total number of faults is not known until the monitored verification level is finished.

The FST results showed that it is possible to get good indications of the average input quality already in the first half of a verification stage. Such data makes it possible to implement process corrections early. Further, by relating the FST status to parts that were verified at certain measurement points, causes of FST can also be revealed. Nevertheless, relating a FST value to the percent faults found does not require an exact science. That is, today, managers make decisions about software quality using best guesses; it seems like this will always be the case and the best that researchers can do is to recognize this fact and do what they can to improve the guessing process [9].

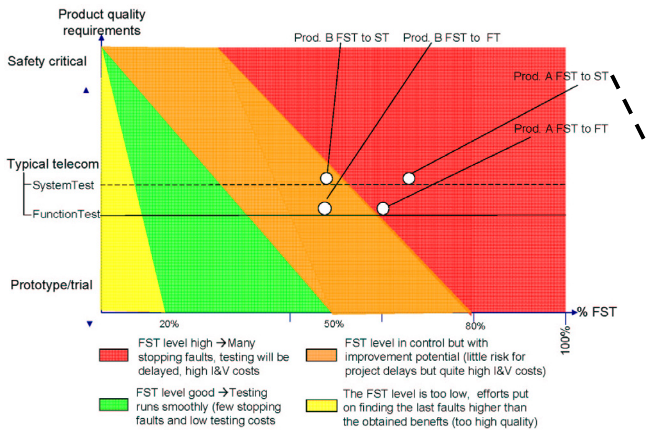

Product quality requirements
Safety critical
Typical telecom
SystemTest
FunctionTest
Prototype/trial
Prod. B FST to ST
Prod. B FST to FT
Prod. A FST to ST
Prod. A FST to FT
% FST
20%
50%
80%
100%
FST level high → Many stopping faults, testing will be delayed, high I&V costs
FST level in control but with improvement potential (little risk for project delays but quite high I&V costs)
FST level good → Testing runs smoothly (few stopping faults and low testing costs
The FST level is too low, efforts put on finding the last faults higher than the obtained benefts (too high quality)

Fig. 6. What is a good Fault slip through rate

Project managers commonly already estimate total number of faults, test cases, and test effort as a part of ordinary project planning (to be able to estimate delivery dates), so this should neither be perceived as hard or time-consuming to do. Regarding the basic FST measurement used, a basic assumption made is that the defined test strategy equals an efficient process. It is also important to be aware that a perfect test strategy is not the one that finds the most faults but rather one that reflects the most efficient way to assure the quality to a level that makes the customers satisfied. Therefore, the optimal FST goal is as earlier mentioned often not zero either Figure 6. Using FST for performance benchmarking of organizations is not recommendable because of:

Product differences: Product maturity, complexity/architectures affect fault slippage ratios.
Process differences: For example organizations using parallel testing (ST starts before FT is finished) gets a higher fault slippage.
Definition differences: Making a stricter fault slippage definition increases the fault slippage values, that is, the gap between the test strategy and current reality affects the degree of FST.
Cultural differences: Might affect how the measure is applied.

Therefore, comparisons should foremost be made against previous releases of the same product [7].

The most common validity critique of FST measurements is that it is subjective, i.e. since the phases belonging of the faults are determined by the people in the projects [6]. Finally, regarding the general ability of the results, the reported FST levels are only generalize within the organization, i.e. because they are dependent on test strategies and product complexity. Further, the commonality of the FST Trends should have some degree of generic pattern, e.g. that they tend to not change during the second half of projects. However, this is dependent on the similarity of the development process applied.

VI. CONCLUSION

Quality is something very hard to define, but it is a measure about how confident is user of the services in operator/vendor. It is always about quality and how the product In Service Performance (ISP) is. However, an ISP fault-free product most likely will not be affordable. Without some balance to the interests of the QA function, it can become too large. These are the influences of the classic market-share dilemma. There is no perfect quality; only good enough. The big new force that is propelling the good enough idea is the explosion of market-driven software. Companies are looking for the shortest path to better software, faster, and cheaper. They are willing to take risks, and they have little patience for the traditional moralistic arguments in favor of so-called good practices. It's time that we developed approaches and methodologies that apply to the whole craft, not just to space missions, medical devices, or academic experiments. Formalities, and the authority behind them, will be reexamined.

But, it must be stressed again, quality is very important especially today when we have huge competitions on the market. Also, two customers with the same application have two views on the quality of the same product. We are witnesses that some of the big vendors with the long history are not very successful today on the market. On the other hand, some new players are very eager to grab market cake.

Who will win in this competition? Definitely players with good enough quality products. And how can we reach the good enough quality products? FST method described in this paper will help in early detection of the quality of the products. On such way, project manager can take appropriate actions to gain quality as we want from our products.

REFERENCES

[1] Miroslaw Staron, Wilhelm Meding, “Predicting Short-Term Defect Inflow in Large Software Projects – An Initial Evaluation”, 11th International Conference on Evaluation and Assessment in Software Engineering, EASE 2007

[2] James Bach, “Good Enough Quality: Beyond the Buzzword”, Software Realities, 1997.

[3] Hyatt, L.E., Rosenberg, L.H., NASA Goddard Space Flight Centre, United States of America, “A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality”, ESA 1996

[4] Garvin, D., "What Does 'Product Quality' Really Mean?" Sloan Management Review, Fall 1984, pp 25-45.

[5] Walt Lipke, "Right Sizing Quality Assurance", CrossTalk, The Journal of Defence Software Engeneering, July 2004.

[6] Damm, L-O, Lundberg, L., Wohlin C., “Faults-slip-through – A Concept for Measuring the Efficiency of the T Test Process”, Wiley Software Process: Improvement and Practice, Special Issue, 2006.

[7] ***, “Faults-slip-through measurement”, Internal Ericsson documentation, Karlskrona, Sweden 2005.

[8] Hevner, A. R., Phase Containment for Software Quality Improvement, Information and Software Technology Vol. 39, 13 (1997), 867-877.

[9] Fenton, N., A Critique of Software Defect Prediction Models, IEEE Transactions on Software Eng., 25, 5 (1999)