Communication in Testing: Improvements for Testing Management

Tuula Pääkkönen[1] and Jorma Sajaniemi[2]

[1] Devices R&D, Nokia
tuula.paakkonen@iki.fi
[2] Department of Computer Science and Statistics, University of Joensuu
jorma.sajaniemi@joensuu.fi

Abstract. Testing in companies with highly competitive environments has many opportunities and challenges. Testing is diversified, both contextually and geographically: there are many testing goals, many ongoing parallel projects, many testing phases and all this needs to be managed so that the full view of testing is visible for management and reporting. Testing is also linked to project communication and psychology: communication needs to be constructive, and testers and test leaders should have good interpersonal skills. It really does matter how a failure or test report is formulated.
In order to localize testing problems in a large software-intensive company, we conducted a current state analysis via web-based questionnaire among experienced testing practitioners, most having at least 5 years experience in these tasks. The scope of the survey was decided to keep broad with the goal of finding future improvements within tool, process and method development.
In this paper, we will concentrate on those survey results that have psychological underlying. Based on the results, we suggest a set of improvements for testing and, especially, test reporting. Implementation of these improvements is still underway, but the findings and suggestions provide insight into psychology of testing.

1 Introduction

Testing in companies with highly competitive environments has many opportunities and challenges. Testing is led by business goals resulting in test strategies that are adjusted according to the needs of individual product program and its assessed risks, features and available resources. The context of testing is very versatile—testing is diversified, both contextually and geographically. There are many testing goals, many ongoing parallel projects, many testing phases and all this needs to be managed so that the full view of testing is visible for management and reporting.

Software systems develop and get more complex, as more features are added. For example, Dick et al. [6] claim that software systems are the most complex objects know to man, far surpassing the complexity of our own brains (having about 10^{10} neurons as opposed to 10^{20} distinct states in a large software system [9]). In addition, third party components, platforms and new technologies also add to the software engineering challenges. Development of new software is always risky; on the other hand legacy code has its risks, also, as changes can have unforeseen side-effects in other parts of a system. Complexity increases the number of internal links between components which increases the burden of testing. In a Norwegian survey [24] among 60 companies with 166 respondents, the highest scores of having a "major problem"" in software development were in testing (30%), documentation (31%) and quality assurance (28%). Interestingly, areas receiving a "no problem" mark were programming (39%), installation (29%) and system design (17%).

Testing does not only involve the functionality of software, but non-functional aspects like performance and reliability, or quality characteristics of the software. In addition, depending on the system under test (SUT), testing complexity is increased by the amount of test data and the number of different system configurations. Potential test data can be infinite, so testing all combinations cannot be done and testing needs to focus on the most critical aspects, most used features and things which end-users most often face.

Already in 2000, Tomayko [33] asked "why do we have 30-year old software crisis." His answer was that in software projects challenging project management conditions are present and this seems to be true even if there has been software process improvement activities [5, 14, 19, 29]. The importance of communication, knowledge and information is emphasized by Aaen [1] who warns about "not confusing information with knowledge": processes cannot be all-covering and there should be room for improvisation. Also the role and importance of tacit knowledge is often forgot, as it is intangible and can be hidden behind several layers of organization. Regardless of improvement methods in use, continuous improvement should be kept flowing [30].

Testing is linked to project communication and psychology. The certification material of International Software Testing Qualification Board (ISTQB) [16] has a section (albeit short) titled "the psychology of testing". The main focus there is on communication: communication needs to be constructive, and testers and test leaders should have good interpersonal skills. It really does matter *how* a failure or test report is formulated. As organizations consist of various kinds of individuals, there is a real need for accurate diplomacy with willingness to listen and to be careful with choice of words used. Defect analysis path goes through development, fix, and verification phases and will often go through many groups of workers. Communication problems may occur if people in the testing path are seen as messengers of unwanted news. Te'eni et al. [32] highlight three communication strategies—contextualization, affectivity and involvements—in order to achieve action, understanding, or agreement or to influence the receiving party. In contextualization, the interpretation of a message is added to it with previously gathered information. Affectivity describes feelings of the sender (e.g., a test engineer), and involvement is mainly empathy trying to look into topic from the receiver's point of view.

In order to localize testing problems in a large software-intensive company, we conducted a current state analysis via web-based questionnaire among experienced testing practitioners, most having at least 5 years experience in these tasks. The scope of the survey was decided to keep broad with the goal of finding future improvements within tool, process and method development. Even though the survey concerned a single company, many of its findings seem to apply also to other organizations developing complex software.

In this paper, we will avoid technical aspects of testing and concentrate on survey results that have psychological underlying. The rest of the paper is organized as follows. In the next section, some background on current testing practices is given. Then the survey and its central results are presented in Section 3. Main findings are then discussed in the next section, followed by some suggestion to overcome problems in Section 5. Finally, Section 6 contains the conclusions.

2 Contemporary Testing in Industry

This section looks at some aspects of contemporary testing practices. We will start with process management aspects, continue with the idea of incorporating testability into the whole software construction process, and finally we will look at failures from the end-users' and test engineers' perspectives.

2.1 Processes and Practices

The ISO 9126-1 standard [15] lists three quality approaches within software life-cycle: process quality, product quality and quality in use. These can be mapped to the software engineer perspective, testing perspective and end-user perspective, respectively. Software engineers use either documented or undocumented processes in software development. Testing engineers work with product quality inside the organization, and work as independent analyzers through testing activities. When software is mature enough, it is released to market. Finally, the last observer of the quality is the end-user. For a software company, process quality and product quality are the easiest to measure. However, quality in use is most important for any product. Even the

price of a product is not such decisive factor in purchasing the product as is its quality. Good user experiences will spread through various social networks and act as good marketing pull.

Agile methods are spreading in software industry; in a recent survey [35], 34% of respondents reported having used agile methods for over two years. Agile methods focus on working software, collaboration, change response and individuals and interactions [7]. From the testing point of view, this is welcomed, as communication is what testing organizations require. The survey also lists as one benefit of agile methods their ability to enhance software quality.

There are few methods suggested for preventing software failures. These include the Personal Software Process (PSP) [28] and process improvement frameworks like CMMI [34], SPICE [17], and Test Process Improvement (TPI) [22]. As stated by name, these focus on the process quality approach as defined in the ISO 9126-1 standard. "Small-scope" failure prevention methods—like PSP and many agile methodologies—focus on the daily work activities of engineers with emphasis on the most important issues, i.e., on actual development and doing the right things. For example, more effort is focused on module testing when components are manageable and less effort is focused on documentation in the early stages of software development when things change rapidly.

2.2 Quality View All the Time

Software failures are unfortunate side-effects of the specification, design and programming phases of software development. Dick et al. [6] go as far as saying that

> "Quite simply, software is at once the most complex and least reliable technology mankind has yet developed."

They also state that failures appear randomly, which makes formal methods an inappropriate approach for software reliability engineering.

Harter and Slaughter [13] instruct that quality is incorporated to a product throughout the life cycle and it cannot be tested into the product. Testing does not create quality, but its role is mostly to enforce and show it, because testing occurs at a late phase of product development when a lot of the software development work has already been completed. Software testing reporting aims at finding failures and to gain insight on product maturity. Test engineers can participate in specification reviews and they can help in module testing, but implementation is done by software engineers and test engineers cannot help in that work.

A classical definition of the purpose of testing is to find errors [25], but Hamlet and Voas [12] suggest that testing should be considered in the earlier phases, also:

> "Design for high testability, then test to uncover failures. When no more are found, there are less likely to be hidden ones that have been missed."

This is based on the notion that the more failures are found, the more failures are still hidden. Good testability makes it easier to find failures, analyze them and basically just find more failures [21]. Good testability also eases considerably the development of test automation. Currently, regardless of the platform and tools used, it seems that test automation is an add-on task, on top of software engineering effort.

Good testability is not a hard thing to achieve. For example, Pettichord [26] suggests that simple logging of events can be sufficient in providing test automation engineers with a deeper view of the internals of the SUT. This is an easy start, which can then be extended by providing means to observe and control the SUT.

2.3 Failures from Different Perspectives

End-users and Failures End-users are getting more aware of computer software failures, which are even getting publicity. For example, there has been a lawsuit to a large company over

selling software which had security flaws [4] and class-action suites showing the joint consumer power. An end-user typically deals with mystical error texts, blue screens or "hangs", and these occur so often that users take them granted as expressed by Bach [3]:

> "We have collectively—and rationally—ceased to expect that software normally works well, even under normal conditions."

Since end-users use many applications in their tasks, errors encountered during a day accumulate from different applications. For example, a photographer faces errors in camera software, graphics editing software, printers etc. Thus, end-users will often encounter many failures within the duration of a day.

It seems that most end-users are quite kind by nature and they use support channels and help desks only minimally [11]. Testing engineers can utilize help desks by collecting reports on failures and anomalies, thus, the use of these channels should be encouraged. Some users actively report failures and take an interest in fix schedules. It depends on the individual and the criticalness of the failure how often certain kind of failures are reported. In open source software, end-users are encouraged to report failures of daily use.

Failures from Test Engineering Perspective Failures may be hard to find. For example, Jeske et al. [20, p. 109] report of a beta-test whose total duration was 528 hours and the last failure in a sample of seven failures took 488 hours (i.e., 14 weeks) to be found. It is thus clear that the most crucial defects should be attempted to be found first, because it makes a difference on the project schedule whether high severity defects are found at week 1 or at week 10.

In mature software the mean time between failure can be several days whereas in immature software new failures can be found by the minute. With immature software, defect numbers can raise high, so that management of failures becomes a challenge. Test engineers need to keep a wary eye on duplicate defects, so that the same failure can be assigned as existing defect to several test cases instead of reporting a new failure instance each time. Uniqueness of failures is important as it eases both development and test engineers work. In agile practices, one of the goals is to reduce waste, i.e., unnecessary steps, which can just add to complexity, but not to value [7].

An important question in test management is the ownership of software failures, i.e., the responsibility of getting a failure fixed. Basically, test engineers are representatives of the end-user, and needed to take up that task. Only by clear ownership, the joint effort of the testing organization can be focused to the most critical software failures. Indeed, software testing organization could take more active role in ownership of software failures than what is currently done.

Finally, individuals as such do not usually disagree about the need of fixing a certain software fault. Engineers do not leave failures in software on purpose, either. However, due to limited resources, everything just cannot be taken care of. It should be noted that correctly working parts of the SUT can only be discovered by testing, which confirms that under the specific conditions used in the tests, the SUT works.

3 The Survey

In order to localize testing problems in a large software-intensive company, we conducted a current state analysis via web-based questionnaire in fall 2007. This section gives an overview of the survey and presents results that are important from the point of view of the current paper. The results are discussed in more detail in the next section.

3.1 Method

The survey was targeted at experienced testing practitioners across the organization, with at least 5 years experience in various testing tasks. The respondent population was test managers and (senior) test engineers in the organization, from many sites and many different projects. Invitation to participate in the survey was sent to 60 people, and 23 answered, yielding 38% answering rate. The targeted sample aimed to find experienced people, and this was successful as four respondents had over 10 years experience in testing issues, ten had 5-10 years, and six had less than 5 years experience; three respondents chose not to tell experience level. Basic testing process and practices used by the respondents were the same with the same infrastructure, but there were also major differences based on different project needs, and people were involved in different testing phases.

Table 1. Survey Parts with Example Questions.

Questionnaire Content	**Nbr of Questions**	**Example Questions**
Background Info	3 open, 3 closed	Q2: Work Profile
A. Test Case Creation	4 open, 2 closed	Q6 What are the problems with test planning? (if any) Q7: What is challenging in test case creation?
B. Information Availability	4 open, 2 closed	Q16: What information of each system under test content is known before testing? (System Under Test (SUT) here = build/changed product/variant or what is applicable in your case)
C. Test Case Designing	9 open, 2 closed	Q19: How are the test cases shared across projects?
D. Testing Management	4 open, 1 closed	Q27: What is the most interesting/inspiring aspect or activity in your testing activities currently? Q28: What is the most challenging aspect or activity in your testing currently?
E. Testing Tasks	4 open, 1 closed	Q32: What kind of testing/activites are considerable part of your work? Q35: I am satisfied with current information of system under test's testing in different test setups?(test setup=test environment, configurations, test data)?
F. Software Development Practices	3 open, 1 closed	Q39: How does the using of agile methods impact testing?
Future Development	3 open	Q41: What would you want to get/achieve with testing management as a process (in future or now)?

The full questionnaire consisted of 12 close-ended and 34 open-ended questions (including demographic questions). Table 1 shows a full listing of all survey parts and the division of questions across various topics that were based on Test Management Approach (TMAP) [27]. TMAP covers the full life cycle of testing with a ready-made framework and was therefore considered a viable way to organize questions in this context. The questions topics were based on the following TMAP areas: (test) control, planning, preparation, specification, execution, check for completion, and infrastructure. As shown in Table 2 that maps TMAP areas to the associated survey topics, the order of questionnaire parts differs from that of the TMAP life cycle: to make the full testing life cycle visible to respondents, the questionnaire starts

with questions concerning test case creation, execution and reporting. Testing management and control topics go along the process all the time, and they were asked at the end of the questionnaire together with future topics.

Table 2. TMAP areas vs. Survey Parts.

TMAP Area	**Survey Area**
Control and Planning	E. Testing Tasks and F. Software Development Practices
Preparation	B. Information Availability
Specification	A. Test Case Creation and C. Test Case Designing
Execution and Check for Completion	D. Testing Management (partly)
Infrastructure	D. Testing Management (partly) and Future Development.

3.2 Results

Overall, questions that most brought up improvement suggestions and problems were in the areas of execution (mainly test case designing), completion and infrastructure. The most important issues named by respondents were reporting, visibility and infrastructure to enable evolving working practices. In the following some questions and most revealing answers are shown. These are discussed in more detail in the next section.

Question 18 "Do you feel that there are any problems with test cases design? (creation, sharing, communication, structure or hierarchy)?" was answered by 20 respondents from organizations where test cases were shared between test projects. For this, five respondents gave an affirmative answer with the following answers reflecting the feelings of many:

- Sharing is a problem. The same work is done in many organizations.
- Test cases are spread across organization, and getting e.g. info on all test cases of one product is a challenge.

Question 28 asked "What is the most challenging aspect or activity in your testing currently?" Answers included the following:

- Test result reporting improvement
- There are many stakeholders and everyone has different expectations or requirement processes.

Question 33 "What is taking too much time?" received also interesting comments with reporting, test case content and updating perceived as problems. In addition, integration with development activities was also seen as a problem:

- Straightforward testing of normal functionality. This is NOT a testing issue, but a development & integration issue. If SW is not ready then the late implementation does break the SW. And the more complex the SW gets, the more effort we have to spend on this.

Question 34 "Into which task(s) would you like to spend more time?" revealed that respondents would like to use more time on testing management than on practical testing activities:

- Test strategy
- Reporting

– More complex test cases matching feature interactions that cause errors not easy to detect. Stability problems (resets, freezes, non reproducible problems.) A lot more into analyzing!!!

So it seems that test engineers and managers suffer from basic things: how to handle various kinds of information, where to get accurate information and on the other hand, how to tell about their own testing so that the data (e.g. testware) is usable for the next steps and management. In essence, communication seems to be a central problem.

4 Discussion

If defect prevention methods are excluded, testing is the phase where defects or failures should be found. Based on the survey, testing engineers feel the pressure put on them. They attempt to communicate with all stakeholders, which means focusing on many different information sources that needs to be reported. A central question is how processes and tools support this information sharing and gathering. This section discusses the findings of the survey from the communication viewpoint.

4.1 Test Case Sharing and Test Reporting

One issue raised in the survey was reporting. This issue was raised in many questions as a sub-tone. Respondent expressed mostly a desire for *overview visibility* across organizational levels, testing phases, and multi-site teams. Testing activities are divided to several teams, and it seems that there is a need for overviews—for both reporting and planning—that would help in grasping the full testing status of a product family or one particular product. Especially, respondents expressed a need for *level-to-level reporting*: what has been done on earlier levels and how it can be made to affect activities in the following levels.

The need for visibility was mentioned both when planning test cases and when reporting test results. In test case sharing, there were two cases where the technical means of sharing a huge test case asset was not seen sufficient. First, expert users were not able to drill down deep enough into the test case hierarchy and select correct test cases satisfying their individual testing needs. Second, novice users found even the current test case sharing methods confusing and unhelpful. Thus, the *technical solution is not able to serve human communication needs.*

With test reporting, respondents' needs were also two-fold: people in test engineer roles desired to have a *more targeted analysis* of data of their own project. On the other hand, managers wanted to get *top-level view* of the testing status with less details.

4.2 Product Complexities

Developing embedded system with hardware and software components creates a complex environment [2]. When product portfolio is large and there are many variants, testing management becomes challenging. Exhaustive testing is impossible, so product development needs to make strategic decisions to select those areas that are most important for testing and focus on them.

In the survey, one of the issues raised by test engineers was the quality of products when they are entering testing. There are many potential causes for low quality, e.g., differences in test environment configuration or development organizations' hurry to get testing results as soon as possible, so that they can start fixing failures. However, bringing software too early to testing will lead to a flood of failures. There is an extra challenge for both sides, as test engineers need to verify fixes when they have been done, and development needs to understand which defects are the most important to focus on. A large number of defect reports leads thus to increased communication needs and, presumably, to *increased communication quality requirements.*

4.3 Importance of Infrastructure

As a sub-tone in the survey, there were scattered hints of problems with existing tool infrastructure and its capabilities. Since then, these issues have been investigated further, and some of the identified change needs have been implemented with some other underway. However, the same problems have been noticed also locally and local solutions have been developed to take care of some aspects of the problems. Thus, *communication problems between organizations* have resulted in overlapping work.

In a different context, Schwarz and Ben-Ari [31] found that students will use tools that they find somehow attractive. The same seems to happen within organizational context—the laws of attraction are just different. In a multi-site organization with lots of ongoing, parallel and sequential projects, the amount of data to manage is huge. Therefore, lots of enterprise-level infrastructure is needed to hold that information and to provide searching capabilities within it. In smaller contexts, one can do test management with spreadsheets, and that can be just a fine tool, but in larger contexts such a simple solution will not work. Therefore, individual sites build and use various test, document and project management tools to plan, report and forecast future. Sites are thus attracted by different tools and *problems in articulating* the importance of a single tool makes it hard to achieve homogeneous infrastructure.

4.4 Processes in Different Contexts

In a large organization with multiple parties, the ultimate goal of processes is to abstract complexities and provide activities and work routines that are easy to follow. In practice, however, different practices and goals are hard to abstract away and to unify, as stated by one respondent:

> "The whole organization should be aligned to work towards the same targets in a big picture by having similar processes and clearly defined and communicated responsibilities."

Harmonizing processes can be very time-consuming and more difficult than anticipated. Agreeing on common processes takes lots of time, as there are lots of different stakeholders, different vocabularies and different prevailing practices. It is also challenging to negotiate about process content, as normal daily tasks must be supported in best possible (or at least practical) way. However, a process is not the goal—the final product is. How then can one manage a complex environment, and yet keep the process itself lean and agile? Furthermore, one part of an organization may want detailed tasks and descriptions, while other parties may want the freedom to use their own solutions. Pinpoint solutions are tempting but they should be geared towards common processes. There seems to be a need for better *means to communicate* in process improvement activities.

5 Solution Concepts

In this section, we will suggest solutions to many of the problems described above. As of writing this, not all of these suggestions have yet been tried in practice.

5.1 Status Reports

Software testing aims to find errors [25], so failure reports form one of its major results. Moreover, testing reveals the overall maturity of the SUT and this is described by cumulative reporting.

Failure reports contain lots of information, e.g., for each defect the project name, unique id, description, test level, severity, logs, screenshots, identification of test object (SUT), link back to existing test case, contact info of the tester, etc. [8, 23]. The main purpose of failure reports

is to communicate detailed information of found defects to the development organization, but failure reports have other uses, also. Currently software process improvement activities utilize this information actively, whereas testing organizations themselves do less so. We suggest that testing organization should use failure reports, e.g., when assessing the validity of existing test cases, so that test cases do not leave gaps where failures might slip away. The number of existing test cases is so large that pruning test sets for unnecessary test cases may clarify the true purpose of individual test cases and sets.

Testing finds failures that cover different failure angles of product characteristics. For example, the same test case can produce one failure that is related to usability, and another that is related to functionality. We suggest that test engineers should view all, say, usability related failures together, in order to help software designers to proactively develop better products. When all failures for certain quality characteristics are easily available and not scattered throughout a report, the identification of failure patterns becomes possible.

Besides individual failures, it is equally important to report the qualities of the SUT as a whole. We suggest that communication of quality should include a descriptive analysis of the SUT, in order to give a better view on the SUT than just pure statistics of passed test cases. Furthermore, it is important to include non-functional aspects of the SUT in reporting, also.

Cumulative reporting describes how testing progresses during time with regard to test cases and failure amounts [23]. As testing is divided to many groups doing testing of different areas (hardware, software) or categories (security, robustness), we suggest combining these into one report that reveals the full status.

As listed above, there are many things to be reported and the message needs to be formulated in different ways for different audiences. We suggest that all data should be made available for everyone via tools. Moreover, these tools should be flexible enough so that they would serve both novices with shallow information needs and experts with their needs for detailed information and complex search mechanisms.

However, a common tool brings forth the problem of data interpretation. There are many stakeholders and henceforth a risk for miscommunication. Especially, test results may turn into "cold" statistical numbers if various stakeholders look at the raw data only, i.e., number of cases passed and failed, number of failures found, etc. There is a need for human interpretation to be communicated between various parties. For example, a test manager could take a step back and give top five current worries; those would be much easier for the development organization to react on than traditional defect lists.

5.2 Terminology Agreement

In one follow-up of the survey, different parts of the organization agreed upon failure terminology. Common terms helped in getting common understanding and improvement, even at the process level. Steps in this direction came from agreeing a common glossary and then harmonizing data model of an existing test management system. The common test management system contains better quality data of various projects and enables better reporting mechanisms and more advanced reporting.

We thus suggest the use of common terminology across cooperating organizations. Even though this seems to be trivial, it is not the standard way of communicating in industry. A common, documented glossary does help, as there are always new people and new subcontractors and other parties joining development.

In addition, software engineers can be accused of avoiding responsibility and trivializing software failures as bugs or "glitches" [10], which make failures sound less harmful. A common terminology with clear definitions for the terms increases honesty in expression.

5.3 Test Engineers' Motivation

Test engineers' task is to find failures, which are then reported to development to be fixed. Instead of having testing metrics to show how many failures are found, we suggest that testing metrics should be changed to measure the number of failures that are fixed. This would highlight the communication aspect of testing: it is not sufficient to identify failures, but failures must also be documented in a way that makes fixing them possible. Moreover, with this change the focus of testing would move from failure identification to product quality improvement.

It can be argued, that if someone owns the copyright to software code, doesn't he then also own the failures? Of course, failures shouldn't exist, so they are an extra add-on to the software. We suggest that ownership of failures should be given to test engineers so that they become responsible for getting rid of the failures. Again, this would improve communication between test and software engineers and improve product quality.

Currently, test engineers' responsibility is to execute piles of existing test cases in order to obtain a certain testing coverage level. We suggest that test engineers should be allowed to take also an open attitude towards the SUT and use their existing competences to do more exploratory testing, i.e., to explore the SUT based on their past experiences. Such a shift from formal testing approach to a more experience-based testing could help in pinpointing some failures earlier.

5.4 End-User Error Reports

End-user error reports are a typical example of communication problems. End-users describe failures briefly, probably assuming repeatability by just writing down few steps, even though the error may be due to some specific environment, configuration issue, or a long series of seemingly unrelated actions. As a result, the developer side has problems in finding the failure cause; the software works for them, but not for the user at, e.g., other side of the world. Furthermore, a typical question in a failure report concerns identifying the component where the failure occurred. In many cases, this can be very hard for an end-user to find out.

We suggest that there should be detailed instructions of how the end-user is expected to report a failure in consistent manner. Existing test management and defect management tools give frameworks of fields, which are available in defect reports, but still most information is in plain text. Development and testing engineers should agree, what each data description field should contain so that it is useful for both parties.

In addition, we suggest that software testing should more actively utilize user stories and use cases [18] also in negative cases, i.e., in order to illustrate the full effect of a failure, engineers should consider what kind of use case does arise if "everything fails". Of course, such "negative use-cases" cannot be anticipated and written beforehand, but in critical failures this could be a good way to communicate the effects of failures.

6 Conclusions

In this paper, we have presented results of a survey originally made to localize testing problems in a large software-intensive company. However, we have avoided technical aspects of testing and concentrated on survey results that have psychological underlying. Based on the survey, we have suggested a set of improvements for testing and, especially, test reporting. Implementation of these improvements is still underway, but the findings and suggestions provide insight into psychology of testing.

Our central findings are related to communication. To start with, software failure and test result reporting needs to be done in a consistent and thorough manner. Of course, a good software process does not automatically end to good quality in the end product, but the development process covers one third of the quality approaches stated in the ISO 9126-1 standard [15], the

other quality approaches being product quality and quality of the use. These can be mapped to the different perspectives of the failures, so that quality of the use relates to the end-user view, product quality is what the test engineers see, and process quality is a matter of software engineering.

A testing organization can typically report product quality via executing test cases and monitoring the product. It may also be able to report some qualities of use after the product has been some time on market, but a testing engineer is not, however, an actual end-user. We suggest that testing organizations should take more active "ownership" of failures found and actively promote the needs for fixes and failure prevention in future.

In order to prevent future errors and to ensure efficient fixing in current products, the error effect as perceived by the end-user should be visualized, so that the true effect is told instead of plain numerical symptom and impact codes. There is also a need to describe product maturity verbally instead of numerical statistics of test cases executed and passed.

Communication problems are worsened by complex development environments. With a simple development organization and simple products, it is possible to know the expert of any area, and to get clear answers to specific questions. Currently, this is not the case anymore. Communication is also often performed by pushing information from one to many. Information is shared actively by different parties by pushing it via various channels like e-mail, www-pages, Wiki-sites, meetings and instant messaging. As different stakeholders have different backgrounds, it becomes hard to follow and understand all this information. As one solution, we have suggested the use of agreed, common terminology that is accurately defined.

In future, we would like to see alternative ways to communicate testing results, since failure data is an important asset for learning. Better communication between the two functions of software engineering—development and testing—will bridge the gap between the two sides and increase understanding between them. It is not just a question of adding something to the communication mechanisms, but also about how to make the communication more accurate, targeted and on-time.

7 Acknowledgements

We wish to thank colleagues from many companies, whose work and feedback has motivated the writing of this paper, and all respondents of the survey who were able to share their insight.

References

1. I. Aaen. Software process improvement: Blueprints versus recipes. *IEEE Software*, 20(5):86–93, Sept.-Oct. 2003.
2. Pekka Abrahamsson, Antti Hanhineva, Hanna Hulkko, Tuomas Ihme, Juho Jäälinoja, Mikko Korkala, Juha Koskela, Pekka Kyllönen, and Outi Salo. Mobile-D: an agile approach for mobile application development. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 174–175, New York, NY, USA, 2004. ACM Press.
3. James Bach. Quality is dead #1: The hypothesis. http://www.satisfice.com/blog/archives/224, 2009.
4. Michael A. Cusumano. Who is liable for bugs and security flaws in software? *Communications of the ACM*, 47(3):25–27, 2004.
5. M. Diaz and J. Sligo. How software process improvement helped Motorola. *IEEE Software*, 14(5):75–81, Sep/Oct 1997.
6. Scott Dick, Cindy Bethel, and Abraham Kandel. Are software failures chaotic? In *Proceedings of the 2002 Annual Meeting of the North American Fuzzy Information Processing Society*, pages 316–321, 2002.
7. Martin Fowler and Jim Highsmith. The agile manifesto. At http://www.ddj.com/184414755.
8. B. Freimut, C. Denger, and M. Ketterer. An industrial case study of implementing and validating defect classification for process improvement and quality management. In *11th IEEE International Symposium on Software Metrics*. Fraunhofer IESE, Kaiserslautern, Germany, Sep 2005.
9. M.A. Friedman and J.M. Voas. *Software Assessment: Reliability, Safety, Testability.* John Wiley & Sons, Inc., 1995.

10. Don Gotterbarn. The education and licensing of software professionals: the myth of "a perfected sciences" considered harmful. *SIGCSE Bulletin*, 32(2):8–9, 2000.
11. Chittibabu Govindarajulu. The status of helpdesk support. *Communications of the ACM*, 45(1), January 2002.
12. Dick Hamlet and Jeff Voas. Faults on its sleeve: amplifying software reliability testing. In *ISSTA '93: Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*, pages 89–98, New York, NY, USA, 1993. ACM Press.
13. Donald E. Harter and Sandra A. Slaughter. Process maturity and software quality: a field study. In *ICIS '00: Proceedings of the 21st International Conference on Information systems*, pages 407–411, Atlanta, GA, USA, 2000. Association for Information Systems.
14. Lena Holmberg, Agneta Nilsson, Helena Holmström Olsson, and Anna Börjesson Sandberg. Appreciative inquiry in software process improvement. *Software Process*, 14(2):107–125, 2009.
15. International Organization for Standarization. ISO/IEC Standard 9126: Software Engineering – Product Quality, part 1., 2001.
16. International Software Testing Qualifications Board. Certified tester - foundation level syllabus. At http://www.istqb.org/downloads/syllabi/SyllabusFoundation.pdf.
17. ISO.Spice. Spice project. http://www.isospice.com/categories/SPICE-Project/.
18. Ari Jaaksi. Assessing software projects: tools for business owners. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 15–18, New York, NY, USA, 2003. ACM Press.
19. P.A. Jansma and R.M. Jones. Advancing the practice of systems engineering at JPL. In *2006 IEEE Aerospace Conference*, pages 1–19, 2006.
20. Daniel R. Jeske, Xuemei Zhang, and Loan Pham. Adjusting software failure rates that are estimated from test data. *IEEE Transactions on Reliability*, 2005.
21. Ronny Kolb and Dirk Muthig. Making testing product lines more efficient by improving the testability of product line architectures. In *ROSATEA '06: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 22–27, New York, NY, USA, 2006. ACM Press.
22. Tim Koomen and Martin Pol. *Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing*. Addison-Wesley Professional, 1999.
23. Tim Koomen, Leo van der Aaist, Bart Broekman, and Michiel Vroon. *Tmap Next, for Result-Driven Testing*. UTN Publishers, 2006.
24. E.A. Larsen and K. Kautz. Quality assurance and software process improvement in Norway. In *Proceedings of the Fourth International Conference on the Software Process*, pages 131–148, Dec 1996.
25. Glenford J. Myers, Corey Sandler, Tom Badgett, and Todd M. Thomas. *The Art of Software Testing, Second Edition*. John Wiley & Sons, Inc., 1994.
26. Bret Pettichord. Design for testability. At http://www.io.com/~wazmo/papers/design_for_testability_PNSQC.pdf, 2002. Accessed 20-05-2007.
27. Martin Pol, Ruud Teunissen, and Erik Van Veenendaal. *Software Testing: A Guide to the Tmap(R) Approach*. Addison-Wesley Professional, 2001.
28. Marsha Pomeroy-Huff, Julia Mullaney, Robert Cannon, and Mark Sebern. The Personal Software Process (PSP) Body of Knowledge, Version 1.0. http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05sr003.pdf.
29. G. Rebovich. The evolution of systems engineering. In *2nd Annual IEEE Systems Conference*, pages 1–5, April 2008.
30. O. Salo and P. Abrahamsson. Integrating agile software development and software process improvement: a longitudinal case study. In *2005 International Symposium on Empirical Software Engineering*, pages 193–202, Nov. 2005.
31. Shmuel Schwarz and Mordechai Ben-Ari. Why don't they do what we want them to do? In *PPIG2006 - Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*.
32. D. Te'eni, A. Sagie, D.G. Schwartz, N. Zaidman, and Y. Amichai-Hamburger. The process of organizational communication: a model and field study. *IEEE Transactions on Professional Communication*, 44(1):6–20, March 2001.
33. J.E. Tomayko. A historian's view of software engineering. In *Proceedings of the 13th Conference on Software Engineering Education & Training*, pages 103–110, March 2000.
34. Carnegie Mellon University. Capability maturity model integration. http://www.sei.cmu.edu/cmmi/general/.
35. VersionONE. 3rd annual survey, "the state of agile development". At http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf, 2007.