Best Practices for the Organizational Implementation of Software Testing

Tim A. Majchrzak
Department of Information Systems
University of Münster
Email: tima@wi.uni-muenster.de

Abstract

Software testing as a main part of the development process is essential for the successful production of highest quality software. We have cooperated with regional companies in order to capture their problems with software development and to learn their distinct best practices in software testing. Based on our observations we created a framework to categorize recommendations for testing. In this paper we present the research methodology used and the framework built. We further illustrate four major recommendations considering the organizational implementation and strategic alignment of testing. They are derived from the analysis of the companies’ status quo and best practices. Each recommendation is classified using the framework to show the conditions its application depends on.

1. Introduction

With the advent of the software crisis in the 1960s [9], software quality finally gained a wide recognition. Spectacular disasters like the crash of Ariane 5 [2], [21] were noticed by the general public. Reports on extensive projects that are not only delayed, but exceed the allowed budget by far due to software problems can be found in the media regularly. In Germany e.g. the road charge-project “Toll Collect” or the currently introduced electronic health insurance card (“Elektronische Gesundheitskarte”) have been subject to lively public and political discussion. Unfortunately, evidence of failing software projects can be found widely [8].

However, it is much worse to see that not only some large projects fail but that delays, budget-exceedance and even failure is common when developing software. Improving software quality [18] is thus a task of greatest importance. Admittedly, achieving quality is an extraordinary hard task. Even after several decades of research, many questions are remaining. As software systems grow in size and complexity, maintaining or even rising the level of quality becomes more problematic. In the style of Wirth’s Law [36] it can be argued that the complexity of software grows faster than methods to control it are developed.

There are many projects that include software development as a major or the only part. Of course, not all of them fail. We hence propose to have a closer look at successful projects. Learning best practices in software development in combination with progress based on theoretical research will be the foundation for better software. With this idea in mind, we worked with local companies to analyze their strength and weaknesses considering the way they develop software. This approach led to a large number of observations. Thoroughly analyzing them enabled us to deduce general recommendations along with a framework to describe the conditions needed to apply them.

In contradiction to common estimation, it is not always obvious what makes testing successful. There is no set of best practices which is generally known and implemented by any company. In fact, individual strengths could be found for almost any participant. Interestingly, some of the deduced recommendations can be found in academic or even practical literature already. Their adoption appears to be more challenging than described, though. Other practices we found are not described for the field of software development but are known in other contexts. A main task and an important contribution of this paper is hence to present recommendations in the light of their applicability. The recommendations focus on the organizational implementation of testing. We also emphasize the strategic impact of organizational change as it has to be driven by the business strategy [12].

This paper is structured as follows. Section 2 introduces the project’s background. Our research methodology is sketched in Section 3. Section 4 explains the framework we designed. Actual recommendations are described in Section 5. Finally, Section 6 draws a conclusion. Please note that we give an overview of related work with each recommendation rather than discussing it in a section of its own.

2. Project Background

A number of companies that either develop software as their main business or that have a software-driven business are based in Münster, located in North Rhine-Westphalia, Germany, and its surrounding region. The software developing companies are mostly medium-sized. Münster also is home for a couple of larger financial service providers, i.e. banks and insurers. All companies are members of the local chamber of commerce. It supports the Institut für Angewandte Informatik (IAI – Institute of Applied Informatics) which is hosted by the University of Münster. The IAI is
based on the interdisciplinary work of the Münster School of Business Administration and Economics and the Department of Mathematics and Computer Science. While the IAI is run by researchers, its projects are accomplished in cooperation with local enterprises. Projects shall lead to new insights beneficial for companies [13] as well as to research progress.

It was known from regular communication between IAI researchers and companies that most of the companies had identified a general room for improvement of their software testing processes. Unfortunately, achieving improvements was not possible in many cases. Besides, many companies were not sure which actions to take. Another problem is that market-pressure forces them to concentrate on operative work rather than to spend time on test process optimization. This led to the idea for the IAI project on software testing (referred to as the project in the following sections).

Despite the problems sketched above, the majority of companies are successful. Thus, they are effectively developing software. This leads to two conclusions forming the foundation of the project:

1) The companies do not have perfect testing processes. Problems of various severity along with a strive for cost effectiveness or improved software quality can be found for any company.

2) Each company inherently has a set of procedures and distinguishing knowledge that enable successful software development.

Due to the diversity of companies and the ways they develop software, strengths and weaknesses can be expected to be comparable to a certain degree only. In fact, strengths are likely to be complementary. There are several paradigms for the style of software development. Several distinct software development processes like the Rational Unified Process [20] and methodologies like Extreme Programming [4] are used. Diverse philosophies towards achieving software quality exist. And, of course, employees with individual backgrounds influence the way software is developed and quality achieved. Besides, it also is important whether software is developed to be sold or if it will be used in the company to support business processes. Each way of developing software and each approach of achieving quality in software leads to other best practices. For practical reasons, companies cannot shift the way software is developed completely. Hence, they stay limited to experiencing some best practices only.

The diversity of participants and the implications for analysis can be best described giving an example. One of the participants provides software for financial service providers, especially banks. As a full-service provider the company develops products needed to represent banking processes including a back-end, calculations software for employees and a customer front-end, e.g. for online banking. This kind of software constitutes a highly sophisticated system. Due to national and international laws and to ensure the customers' trust in it, the system has to comply with highest security standards. The company does not only test it in a traditional way but has an in-house installation simulating a banking landscape. This installation is built from several mainframes and server systems, powerful enough to serve actual banks. It also consists of a triple-digit number of terminals to simulate intra- and inter-banking transactions as well as the access of banking applications by customers. Building up such a testing installation takes years of planning and a considerable effort, yet enables a very realistic testing. Some defects only appear due to complex interaction and the workload of real applications and not by running simple test cases or mocking functionality. Such defects can be found before software is deployed to the banks. Building up an infrastructure like this forms a recommendation with specific conditions under which it should be considered for realization. It however is only applicable to a very limited number of other companies and thus not further discussed in this paper.

The project's ambition was to determine best practices identified by participating companies and to collect those that should be helpful for the majority of companies. The methodology used to learn about the companies' best practices and to derive recommendations from them is described in the next section. A comprehensive book of recommendations will be shared among participating companies.

3. Research Methodology

The project has a special character as it is conducted by information systems researchers, supported by the chamber of commerce and targeted at local companies. Research methodology reflects this as it had to adopt the requirements of practitioners. This includes a limited time for meetings with executives from companies along with the companies' interest in results that can easily be implemented. Recommendations were not only to represent theoretical thoughts on possible improvements but state actions that are advisable. At the same time an academic flavor should be kept to make sure project results mark research progress.

Qualitative research methods were used to compile best practices. They are perfect for "understanding a phenomenon from the point of view of the participants" [23]. Semi-structured interviews with experts from the participants led to data suitable for analysis. At the same time, the flexibility of interviewing employees from heterogeneous companies could be kept. Even though we do not publish them as such, each set of interviews can be seen as a kind of case study [38]. "Research in IT must address the design tasks faced by practitioners" [22]. Our research methodology therefore is related to design science. It has a focus on solving business problems and seeks to lead to both technology-oriented as well as to organizational-focused findings [13].

Ideally, the return-on-investment (ROI) of improving software quality should be measured [29]. But this is only possible with a quantitative analysis such as described by
The authors formulate hypotheses and test them against empirical data. Despite the focus on code inspections, the approach could be transferred to the topic of our paper. However, we focus on qualitative analysis. A subsequent project could check (by utilizing a variety of metrics [18]) if implementing recommendations helped companies to increase annual revenues due to higher software quality [29].

The basic course of action can be sketched as follows. Firstly, companies supporting the IAI were contacted. Employees responsible for testing were chosen, e.g. senior test managers, managers of departments responsible for testing, or leading software development employees.

In a second step, researchers visited the participants for expert interviews. For smaller companies the manager responsible for testing was interviewed directly. For medium-sized to larger companies meetings with various employees were arranged. We sought for a deeper insight into all aspects of testing in the respective companies. The interview was opened with questions on how testing is done by the company. This included to learn who is responsible for testing, when testing is done, what is actually tested (e.g. user interface, system interfaces, main algorithms), which techniques are used and how testing is done. Further questions considered the used testing tools [15].

After conceiving the status quo, company specific problems and excellences were discussed. We asked our interview partners to argue about any troubles encountered when testing software. Besides, we tried to find out what changes to testing were sought for, even if the current situation was not considered problematic. Finally, we discussed strategies for successful testing. This part of the interview was the most creative one; rather than asking fixed questions, we encouraged our interview partners to capture and reflect on the way they test software. This led to fruitful discussions which gave us the chance to discover best practices.

The third step was to arrange and accumulate the results and to find interdependencies. An overview of the status quo has been drawn which is a particularly important information for the participating companies. Then best practices described by our interview partners were analyzed and recommendations extracted. Similar strategies were aligned and interrelations discovered. This included taking the companies’ situation and the kind of software development into consideration. Especially, we had to check carefully the prerequisites that applied when best practices were developed. Even though a strategy might be suitable in terms of enabling the development of high quality software, it might not be applicable in general.

The above consideration led to the construction of a framework (see Section 4). It is meant to help deciding which circumstances have to be met and which prerequisites have to be made before implementing a recommendation. The actual implementation varies with particularities met in companies and often can be done in several steps.

4. Developing a Framework

Five determinants were taken into account when we designed the framework. They mark the most notable conditions under which recommendations apply. We directly derived them from what companies told us about their software testing best practices.

The level of demand characterizes how much effort and how fundamental strategic adjustments are needed to implement the recommendation. While some recommendations are mere hints how to perform better, others cannot be implemented easily. Especially, some recommendations rather require the initiation of a continuous process of optimization than to represent a single action. In general, effort needed to adopt a recommendation correlates with the impact on testing and the longevity of the actions to undertake.

We consider basic recommendations to either be applicable without difficulty or to be very fundamental. Due to the basic nature of the recommendation, the processes of many companies might be in alignment with them already. If they are not, we strongly advise adoption.

Many recommendations are more than basic hints. If they can be implemented with a reasonable effort or we consider their effectiveness very high, they are categorized as advanced recommendations.

Some recommendations mark target states. These ideals cannot be reached easily. We consider them as general guidance. Whether companies should invest in achieving the target state, or not, is dependent on their situation. Essentially, actions that are categorized as target states represent continuous approaches to optimize testing processes.

The second determinant is the project size. In small-sized projects typically the development team is responsible for testing directly. Medium-sized projects usually have individual teams for development and testing. They also have some kind of manager to coordinate both teams. Large projects in our definition can only be found in companies with at least a couple of hundred employees. There are not only project specific teams for development and testing but departments with general functionality. An example is a department for testing which would be involved in any project.

We do not mean the differentiation to be measured by actual numbers of employees involved but by the characteristics projects meet. Large companies use to have the roles of developer and tester separated even in small projects. Smaller companies with less than 100 employees hardly have general departments even if some employees have special skills and participate in several projects simultaneously.

As the third determinant we differentiate between the kind of software produced. There is individual software development on the one hand and standard, i.e. mass market software on the other. Regression testing e.g. is not as important if developing software individually, especially if the project is a one-time contract. For a mass market application that will
be supported and revised for years, it however is extremely important. Individual software development often premised a close customer relation; standard software is built for the (mostly) anonymous market. Many recommendations are suitable for both kinds of software development. Some of them only focus one of it, though.

It is mentioned if recommendations only bear a meaning for a given number of releases of a software developed. Some actions can only reasonably be taken if there are at least a couple of releases. We differentiate between one, several and regular releases. Regular describes that the software will be released and supported at least for some time, probably without initial temporal limitation, and that there will be releases of updated versions.

Some recommendations are applicable for the complete process of testing. However, many of them are mainly meant for some or only one phase. This concept [35] is also referred to as staging by practitioners. Stages are:

- **Component test** Developers test the code just written. This is done on the level of a module.
- **Integration test** Several modules of the software to develop are connected and their interaction is tested. Stubs and mock objects [33] are used.
- **System test** Major modules and components are connected and their interaction is tested. External services are still simulated by stubs, however, most internal services are available. Remaining stubs are replaced by the actual services during this stage.
- **Acceptance test** Prior to releasing the software for the customers, there is a final test. It is no more technically driven. The basically finished system is tested against its requirements.

All determinants are depicted in Fig. 1. The level of demand and the phase of development build a matrix. Ticks imply that a recommendation is meant to be used in a specific phase under assumption of the specific level. The three other determinants are represented as bars. A gray shade indicates that a recommendation is meant to be used under the given condition. The fading means that the recommendation might be used for a condition but this decision has to be pondered upon (see the below example for clarification). Analogously, a tick in the matrix can be in brackets to indicate that the recommendation applies to the specific phase but is optional or less important.

Please consider the example in Fig. 1:

- The recommendation can be applied to the phases of integration, system and acceptance testing. It both has a basic and an advanced part indicating that the general recommendation should be followed in any case. It does not need to be implemented in a single step. Positive effects beyond a basic benefit can be achieved.
- If implemented completely, component testing will be affected positively as well. As this effect is secondary, the tick is displayed in brackets.
- A major impact is expected for small-sized projects. The recommendation should only be applied to software that has subsequent releases. The fading indicates that there should be a larger number of them. In addition, it fits to software released in regular circles. Naturally, the framework is a basic guideline and has to be augmented with additional comments. It should not be seen on a pure project level. At least larger companies and extensive software development departments should judge the applicability by using the framework on typical projects.

Whereas the framework was introduced to aid the accessibility of the recommendations, it can be used generally. It should be helpful whenever best practices or recommendations considering software engineering are to be applied to companies of heterogeneous nature.

5. Organizational Recommendations

5.1. A Simple Idea First

To have separate roles for software developers and testers seems obvious. Even older practitioner’s literature such as [19] leaves no doubt that there are not even distinct roles but developers and testers are not the same employees. We however found that – especially in small companies – there is no such distinction. By reasoning from the literature and by observing how a definition of roles helps those companies which have it, we recommend to define roles for developers...
and testers and to adjust testing to them. This is a basic recommendation which applies to any project (Fig. 2).

From our findings we conclude that a shortage of staff, small budgets and approaching deadlines do not justify to have blurred roles. It is important to understand that roles do not enforce that a single employee cannot both be developer and tester. Having defined roles and making sure that each employee only works according to the demands of one role at a time is very important to keep complexity under control and keep projects structured. Only with an understanding of steps taken in a project, tasks assigned to employees, the amount of time spent on tasks and an idea, how successfully the tasks were fulfilled, software quality can be managed.

Especially when working test-case driven, writing test cases is part of the development [6]. It is perfectly acceptable that developers write test cases first and then work on the productive code. Tests from the integration phase on should however be seen as independent steps. Employees doing these tests should be testers when executing them. Testing must not be seen as an incidental, negligible task. To counter distraction, developers taking the role of a tester should not test their own code from the integration phase on.

The tester’s role has to be defined and supported strategically. It should for example be reflected in the assignment of tasks and time budgets. Testing has to be motivated as an important task in the development process. If it is seen as a cumbersome and exhausting task that consumes time which is needed for development, testing will not be successful. It should be done in a structured and documented way. We strongly disapprove to have testers directly try to fix defects they found, or to only report them to a developer. Documenting them is a very important task. It is done to prevent mistakes that led to the defect from being repeated. The defect can be fixed and the affected functionality tested again. Unstructured ad-hoc fixing usually leads to new bugs entering the productive code.

Testing ought to be recognized as a task that is as demanding as development. Developers with substandard skills or little experience are still likely to develop software that basically works. It is however highly questionable if they develop a software of high quality. Analogously, software can be tested by testers with substandard skills or little experience. They of course will find some, probably even most of the defects and help achieving a higher software quality. It is very questionable if the number and especially the kind of defects found and removed will lead to a software product that is considered to be of high quality. The most subtle defects that may have dramatic consequences if the software is used productively are likely to remain. Thus, developers and testers should have the same level of qualification. Employees with little experience should be supported by senior personnel.

Setting up the testing staff is a task that has to be done in accordance with the company’s situation. Small companies may not be able to employ dedicated testers. It therefore is very important for them to keep control of clear roles and tasks. We suggest to formally evaluate which employees are especially skilled for different types of testing. Ideally, teams of developers and testers that have complementary skills can be built. Some of our interview partners in the project also argued to have non-technically skilled staff among the testers. They even suggested that “nitpickers” would be the ideal candidates for this task. Employees with a critical attitude towards software are likely to find defects that testers with a deeper technical knowledge overlook. At the same time, it is highly discouraged to have executive staff participate in testing but in the very last phase. Executives usually cannot spend much time on a single task. They are hence testing superficially and do not find many new defects.

For later testing phases we suggest to involve sales personnel (also see Section 5.2), especially if the software is developed individually. After being briefed shortly, the sales employee tries to work with the product as if it was in productive use. Due to their customer-related experience, sales personnel know how the software product is supposed to work. Besides that, they can identify usability shortcomings. As a side effect, sales staff gains knowledge of the software. Only short coaching will be required when it is released. Sales staff is able to introduce the new software product earlier to the customer. It can be concluded that involvement of sales personnel leads to a knowledge transfer which both sales and development benefits from.

5.2. Requirements Engineering

Software quality is not limited to the demand for software with a small number of defects. In fact, the software has to be as useful as possible for its intended operation purpose. Furthermore, it should ease the workflow and support or even enable business processes it was implemented for.
It can thus be concluded that research and evaluation of requirements is very important. Defects caused by improper specification cannot be compensated by testing. Imprecise requirements and their consequences for software development lead to more complicated testing [18].

Requirements engineering is not a new approach and extensively covered by the literature [14], [30], [31]. Studies have found a correlation between proper requirements engineering and software development project success [34]. However, the “gap between research and practice in requirements engineering is immense” [25]. Both the positive impact of successful requirements engineering and the disaccord of theoretical finding and the practitioners’ reality were reconfirmed in our project. Further information on requirements engineering can be found in a survey on relevant technologies and their coverage [26].

We strongly suggest a strategic focus on requirements engineering when developing software. This basic recommendation applies to any circumstances under which software is developed (Fig. 3). Software is always developed to meet customers’ demands. It is of no regard whether it is developed for external customers or for an internal department. It also is of no importance whether customers have ordered the software or if it is developed initiative (e.g. for the mass market). In all cases the – probably yet unknown – demands and wishes of the customers have to be met. This can only be achieved by involving the customer in the development process or by researching on her demands.

For the contract-based development we found that discussing the requirements with the customer is a successful approach. This should be rather done than just to implement a product requirements document point by point. Ideally, the customer is not only in contact with sales personnel but with (senior) developers as well. These developers should be coached so they become skilled as requirements analysts. The more developers understand of the customers’ value creation, working routine and culture the higher will be the quality of the developed software.

An additional recommendation is to include staff for testing into the team that gathers the requirements. Requirements engineering is used to prevent software from having defects that can hardly or not at all be repaired by testing and debugging. Having testers involved in the specification process serves towards the aim of higher software quality. Existing standards such as [16] should be used to gain knowledge of requirements. Literature for practitioners such as [28] offers guidelines and checklists. We suggest to carefully adapt guidelines rather than to implement them without reflecting upon the individual way software is developed.

Extending our recommendations we suggest keeping the customer involved throughout the development process. In this context customer again could be an internal department if not developing software for the market. Continuously involving the customer does not necessarily require the

![Figure 3. Classification of the second recommendation](Image)
customers’ requirements. Ideally, a long-term relationship to particular customers can be established with both sides profiting from the close cooperation in development.

When developing software for internal customers, close involvement of one of the departments that will use the software is recommended. In addition to the above benefits which also apply for internal development, acceptance problems which commonly occur on introducing new or changed (i.e. revised) software can be minimized. Internal staff not only gets to work earlier with new software and has more time to become accustomed to it but also can express wishes for changes before the software is finished.

5.3. Implementation of a Test Center

Companies with a higher double-digit number of development staff usually have dedicated testers among them. These employees should be systematically trained and their work should be focused on testing (see Section 5.1). As the number of testers grows, we recommend setting up a special department which we call test center. A test center is not merely meant to centralize testing which could even be contraproductive. In fact, we encourage companies to install a department that will coordinate testing companywide. It can further provide recommendations and guidelines and help to build a more transparent testing process. Additionally, it can pool resources and employees in order to adequately assign them to projects.

We recommend a test center for medium-sized and large companies that at least operate medium-sized projects (see Fig. 4). Due to the organizational change which is required, this is an advanced recommendation with potential for development. In the phases of component and integration testing a test center can be helpful e.g. by offering guidelines for successful testing or by assigning additional testers in case of a shortage. Its full value however is realized in the phases of system and acceptance testing. It can arrange teams of highly specialized testers, brief them and coordinate complex tests. This not only burdens the development staff of the affected projects but offers testing that meets the highest standards. Naturally, the larger projects become the higher these benefits will be. Setting up a test center has to be a strategic decision and it should be seen as a holistic instance for testing support. Thus, it is useful for software with at least a couple of releases. While it serves well for individual software development, it is especially useful for mass market development (see Fig. 4). The test center in our definition is a virtual entity. Black describes the concept of a test lab which is an actual physical location and especially used for hardware testing [7].

The main task of the test center is coordination. As a cross-divisional support-department it should be seen as a “helping hand” rather than as a regulator. As such it should provide guidelines for unified testing methods, recommendations for successful testing strategies and a corporatewide vocabulary. Guidelines should not be mandatory. Ideally, the test center becomes an entrusted department and faces general acceptance. It therefore can also act as a mediator between developers and non-technical corporate departments if these are involved in software development projects.

In cases where software is developed by departments or project teams that have no dedicated testers, the test center can support development by offering testers from a reserve of available employees. Later testing phases could be supported or carried out completely by test center staff. By doing so, a cross-project knowledge transfer is enabled. This is a competitive advantage as it offers the chance to increase productivity [1]. Projects that are near completion will also be taken a step towards perfection if testers with experience from a high number of projects are assigned to it. In the phase they get involved, the majority of defects should have been found. However, some of the most subtle defects may still remain. From our experience it is almost impossible to track down such defects without a deep understanding of software testing in general. Hence, senior testers are very helpful as they have acquired the skill and experience to look for those defects that otherwise would not be faced until software is deployed to the customers.

The test center is also able to bring a higher level of transparency to testing. For instance, it can support test documentation by offering document templates or by operating sustentative systems. It should also declare naming conventions. As a central department it is predestined for providing the test controlling (see Section 5.4).

Summing up, the test center should concentrate knowledge and experience and transfer it back into projects. It also is very useful to fulfill tasks that help to improve testing but are not economically feasible in single projects. This for example is the evaluation of tools for testing. An evaluation can greatly aid the tool usage but there is no sense in rating...
tools for every project. Rather, the most useful tools should be picked and offered to all development teams.

While we recommend introducing a test center, we cannot suggest the power it should initially get. No clear recommendation could be derived from what we were told by the project partners we interviewed. In order to avoid a resentment from developers and testers that could feel domineered, one idea is to start with a test center that has an advisory function only. At the same time, at least some corporate wide influence is required for a test center to operate efficiently. Thus, it could propose guidelines but take no direct actions in projects. If it gains trust over time, its level of authority could be raised. Another idea is to have a test center with a high level of authority but a pro-active role. As such it would monitor testing activities and act on request. If it however spots problematic activities it would take action. One participant insisted the test center should be authorized to “stop” fallaciously governed projects.

From the above considerations it is clear that setting up a test center is dependent on the way a company is run and by its philosophy. We tried to motivate our recommendation according to that. The actual implementation of it is however out of scope of this paper and has to be carefully adjusted to each company.

5.4. Test Controlling and Performance Measurement

Our most advanced and demanding recommendation is to install a software test controlling. This should go along with setting up a performance measurement system. It is used to process key figures gained from monitoring and documenting software tests. With these means the evolution of testing performance can be seen over time. Potential problems can be identified and countering actions taken.

Controlling as a part of management accounting is well understood and there is a wealth of literature on it [3]. Performance measurements and the required systems have been discussed extensive, too [10], [24]. Albeit, there is no literature distinctly discussing controlling in the context of software testing. Each comprehensive work on testing comments on some aspects of measuring test performance and similar topics but there does not seem to be a holistic view on test controlling, yet.

A lot of data and a number of key figures are needed to successfully judge the status of testing. Moreover, information derived should be available in real time. Introducing a performance measurement system is an advanced task that needs to be refined until it offers its full functionality (see Fig. 5). It is for example possible to start with a report-based system that will be upgraded to a real-time system. Setting up the system and adjusting the processes to it takes a lot of effort and is expensive. Hence, we only recommend it for large projects or very large enterprises. It can be

![Figure 5. Classification of the fourth recommendation](image)

used for individual software development but is especially helpful if standard software is produced. There should be continuous releases of that software product or at least long-time projects. We only discuss organizational aspects of a measurement system; technical aspects are out of scope.

The main prerequisite for test controlling is a comprehensive, complete and reliable documentation of all testing activities. A test documentation standard is described in [17]. Guidelines for proper test documentation can also be found in the literature, e.g. cf. [11]. The quality of all data collected should be of highest importance. The test controller’s first job is thus to check the data. For example, testers should document any test case implemented and each execution of it. Inspecting this data starts with simple tasks such as checking if required fields of a data collection sheet have been filled. More advanced checks include validation of consistency. To give another example: If a test case has been created in May, it must not be listed to be enclosed in a test case suite of March (of the same year).

Checking the data helps noticing problems early. Consider a testing phase that was planned to last six weeks. 1,800 test cases have been scheduled to be tested during that time. They are similar in complexity and testers are linearly assigned during the time. A warning should be issued if three weeks have passed and only 300 test cases have been reported to be finished. There still is time to run up until the deadline is reached. Without a controlling noticing such problems is based on luck or employee’s alertness. Any abnormality found should be discussed with the developers and testers. This aids the controller in better understanding a project. And it helps developers and testers to avoid similar problems. Hence, abnormalities will not become problems.

Improving the documentation offers further amenities. Errors and inconsistencies are much better to spot. Moreover, modules not sufficiently tested are likely to be identified. Consequently, not only the quality of testing can be im-

![Figure 5. Classification of the fourth recommendation](image)
proved but also the efficiency increased. Well documented test cases are perfect for – ideally automated – regression tests. If the way defects are found and fixed is documented, similar defects can be easily solved in future projects.

Even for medium-sized projects a number of several hundred test cases can be expected for every phase of testing. This figure grows rapidly with large projects. If data of high quality is collected for any test case (adequate tool support is needed to do so), it can be aggregated to get key figures. Example figures are the number of defects found per test case or test case suite executed, or the numbers of defects found per phase of testing. If the latter figure does not decrease with the later phases, this hints to serious problems. Further information can be deduced if defects are categorized, e.g. by severity or by their impact on components of the software in development. An example result could be a high number of defects in the user interface. Correctly interpreted, counter actions can be taken. Defect data can also be included with personnel statistics helping to effectively assign developers and testers.

The value of key figure analysis can be further increased by watching figures over time. A company implementing actions like those proposed in this paper can observe consequences and judge their effectivity. For formalization, statistical tests of significance can even be applied. If desired, the full power of statistics can be used to analyze the data. Trend analysis for example is a valuable tool to demonstrate quality improvement to the company’s executives.

The desired state is a system that controls the whole testing process. As an early warning system it can report if afore defined tolerances are exceeded; as a pro-active pool of information it can enable managers to assign personnel. Bottlenecks can be spotted early or avoided completely. At length, the testing process becomes notably more efficient.

The above described benefits can be realized step by step. A first implementation could be a set of spreadsheets which are filled out manually and used to generate weekly reports. Eventually, an own system should be set up and connected with tools for testing, a database for test cases and similar software. With each stage of expansion the availability of qualified information is increased, as should be their actuality. At the same time, the experience of test controllers will grow so they become highly skilled in judging about the figures the system calculates. We strongly recommend the implementation of such a system for any larger enterprise that is developing software. Parts of it could be implemented by smaller companies, too. After all, a performance measurement system is the only objective mean to judge about the value of testing. As a participant noted, it is very valuable to get information “at the push of a button”. The system could thus be a strong competitive advantage.

To our knowledge only few companies operate systems with functionality that is comparable to the one sketched above. We also do not know of any tool available to completely fulfill these tasks even though some commercial test management tools offer a lot of reporting functionality. We suggest automatizing as much functionality as possible in order to relieve testers of repetitive tasks. Furthermore, automatization of many of the controllers’ tasks is possible, e.g. using regular expressions to verify inputs. The vision we gained throughout the project is a joint system for developers, testers, controllers, and management. It could serve each role with the information needed to fulfill her task. As such, it could incorporate test case management, project planning, time management, task assignment, and a management dashboard.

6. Conclusion and Future Work

We have presented results from a recent project we undertook with regional companies. Based on detailed qualitative interviews with experts from the companies, we analyzed the status quo of software testing in those companies and captured their best practices of testing. Using the results we built a framework to classify recommendations on how to achieve better software quality through testing. We proposed four major actions to take in order to improve the testing process. Each one was classified using the framework.

With the experience of classifying about 30 recommendations (which are yet to be published), the framework has been on trial and results are very promising. Future work could include a refinement of it as well as more detailed comments on its usage. It also could be checked whether it can be combined with other approaches to even better categorize best practices.

By doing the interviews and by thoroughly analyzing the data we gained, we can conclude that there is a discrepancy between the literature’s view on organizational and strategic aspects of software testing and the actual implementation in industry. As a main result of the project, we recommend an adjusted view on software testing. Academics – and probably anyone not directly involved in software testing in a development project – have to realize that the reality of software testing is not yet captured in the literature. Some recommendations described in books do not seem to be implementable at all. Practitioners have to realize that it is ineffective to try to optimize software testing by laboriously checking different strategies. Rather, learning from each other is a very promising approach. Due to the importance of testing and the impact of possible improvements testing is proposed to be taken to a strategic level.

Especially for small companies, it may seem impracticable to define a test process and implement sophisticated testing methods. However, testing has to be seen as an integral part of development. It contributes to value creation by enabling a company to produce software of higher quality. We thus strongly advise not to recognize testing as a driver
of costs but to implement a culture of testing. Following companies that announce their quality management to be certified to comply with the DIN EN ISO 9000 family, enterprises should advertise the fact that they are testing with high effort. Eventually, this can influence the company’s recognition by the public and underline its focus on quality.

While recommendations have been drawn, we continue the project this paper is based on. Next steps include thoroughly discussing results with the participants. Ideally, a further discussion with practitioners – and with researchers [37] – will stimulate the research’s progress. Comments can help to refine the recommendations and especially give more details on their applicability. Further work will concentrate on analyzing the status quo as an expected gap between small and large enterprises was reconfirmed in the project.

Since the work was purely qualitative, the ideal supplement would be a quantitative analysis. “Software quality improvement should be seen as an investment” [29] and the recommendations’ impact and the possibly level of improvement can only be measured empirically. Both a general questionnaire aiming at a high level of participation could be set up and in-depth analysis could be done with some companies. Latter could be used to calculate a “return on software quality” [29] to give strong empirical proof for the recommendations’ effectiveness.

References


