Software Testing in Small IT Companies: A (not only) South African Problem

Stefan Gruner, Johan van Zyl

Department of Computer Science, Faculty of Engineering and IT, University of Pretoria, Republic of South Africa

ABSTRACT

Small software companies make for the majority of software companies around the world, but their software development processes are often not as clearly defined and structured as in their larger counterparts. Especially the test process is often the most neglected part of the software process. This contribution analyses the software testing process in a small South African IT company, here called X, to determine the problems that currently cause it to deliver software fraught with too many defects. The findings of a survey conducted with all software developers in company X are discussed, and several typical problems are identified. Solutions to those (or similar) problems often already exist, but a major part of the problem addressed in this contribution is the unawareness, or unfamiliarity, of many small-industrial software developers and IT managers as far as the scientific literature on software science and engineering, and especially in our case: software testing, is concerned.We also discuss two prevalent test process improvement models that can be used to reason about the possibilities of process improvement. This contribution is an extended and revised summary of a longer project report [56] which can be obtained from the corresponding author of this article upon request.

CATEGORIES AND SUBJECT DESCRIPTORS:

KEYWORDS: Software process improvement, Software testing, IT management

1 MOTIVATION

Software has become so intrinsic in the fabric of modern society that its value cannot be ignored. This has opened a window for many businesses wanting a ‘slice’ of the software ‘pie’. Moreover, because software engineering is not (yet) a protected profession, anybody with money for investment can hire a few programmers and start his own ‘software business’ — in contrast to, for example, an accounting firm which is required *by law* to be operated by state-certified and chartered accountants. Opportunities like these have given birth to thousands of small software companies (SSC) across the world. Small software companies, with between one and fifty employees, constitute the majority of the global software industry [57].

SSC face many challenges in their pursuit to create acceptable software and to survive in an increasingly competitive market place: The first challenge is that SSC tend not to believe that the same processes and methods used by large software companies are applicable to them as well. The reasons for not adopting more rigorous methods are typically: *cost*, lack of resources, time, and level of complexity [3] [53]. A second challenge facing SSC is

Email: Stefan Gruner `sg@cs.up.ac.za`, Johan van Zyl `johanvanzyl80@gmail.com`

that their processes and organisational structures are mostly informal which usually leads to a chaotic workflow [50] [52]. A possible reason for this is that SSC have to focus on time-to-market to survive and thus often neglect the more resource-intensive rigorous processes. A third challenge is a lack of resources in terms of skills and experience [53] [41]: SSC can often not afford to employ experienced software developers, not even to mention software engineers. A fourth challenge, is that despite the many Software Process Improvement (SPI) programs such as CMM (Capability Maturity Model), CMMI (Capability Maturity Model Integration) and ISO/IEC15504, SSC are either not aware of them, or SPI programs are not tailored towards solving the problems facing small companies [53] [22]. These problems are not specific to developing countries such as South Africa; they are also known in highly developed countries like Germany with its approximately 20,000 software producing companies. A study conducted on the German software industry [9] indicated:

- Only about 30% of all German IT houses followed a somehow 'defined' workflow in their software production process.
- Only about 22% of all German IT houses cooperated with scientific software engineering research institutions for the sake of software quality improvement.
- Only 5% of all German IT houses were on a CMM maturity level of 2 or higher.
- 95% of the approximately 20,000 German IT houses were on the lowest CMM level 1.

The problems facing small software companies mentioned above are similar to the problems faced by company X, a small South African software company with approximately 40 employees, eight of whom are software developers. Their workflow process is 'officially' defined, but not followed consistently in each project. The type of software products developed by company X covers a wide spectrum that ranges from small desktop applications to large web-based content management systems for various platforms. Company X focuses on current customer needs and will attempt any type of development project required by a customer, even if there are no skills in that particular domain. This diverse product portfolio can be attributed to a need for securing revenue in a competitive market.

Not adhering to the defined process during projects can be attributed to a *lack of historical project data* which leads, amongst others, to unachievable and unreasonable deadline estimations in current projects. The project becomes rushed as it falls more and more behind schedule, which in turn leads to ad-hoc shortcuts being taken in the process. The result of this rushed and ad-hoc process are untested software with questionable quality. The other major problem facing the company is the lack of skills and experience, as none of the developers have any industry experience, apart from that which was gained since working for company X. Together with the lack of financial resources, software developers currently employed are university graduates with little or no industry experience. This leads to customer facing projects being a training ground for software developers that spend most of their time performing research to come to grips with new technology. The software process used in company X is the well-known Waterfall model, where the software testing phase follows at the end of the process. The rushed and chaotic nature of software projects at company X causes the testing phase to be largely ignored, and most often used for implementation of missing functionality. This leads to late, untested software products fraught with defects. This is not a sustainable model as frustrated and unsatisfied customers will not return for future cooperation.

The contribution of this article is a critical assessment of the software testing process followed at company X as well as a proposal for an improved process, based on the software engineering literature as well as on insight from experience and interviews with industry test practitioners. Since company X is a *typical* case, representative of many others, this contribution should be of interest to a wide range of readers from the IT industry as well as from academic software science and engineering institutes. As an 'added value' this article also offers a rich literature database under the following two aspects:

- The aspect of *small IT companies* (and their software quality procedures) is covered by references [2] [3] [11] [12] [22] [25] [26] [32] [36] [39] [41] [50] [51] [52] [53].
- The aspect of *internationality* (for comparing our South African situation with the situations in countries such as *Australia, Brazil, Canada, Germany,* or *South-Korea*) is covered by references [9] [19] [34] [37] [51].

2 PROBLEM ANALYSIS

To study the current testing practices of company X more closely, several interviews were conducted with all its developers [56]. The questionnaire used in the interviews contained fifteen open-ended and two close-ended questions. The questions were clearly explained to the participants. Interviews lasted for about 30 minutes. The last question required that the developer 'model' the testing process as he currently performs it. Participants were given one full day to complete this question. The main objective of the interviews was to determine the current state of testing practices in company X and identify concrete problems within the current testing process. More specifically, the questionnaire focused on eliciting information from the following three categories [56]:

- the level of software testing skills and knowledge of the developers in company X,
- the company's managerial commitment to software testing as perceived by its developers,
- the current software testing process used in the company.

Unfortunately, this survey was of a rather 'qualitative-interpretative' character, since only eight people participated in the survey, which is per-se not a statistically significant number. However, since company X is *small*, those eight people comprised 100% of the company's software developing members of staff at the time when those interviews were conducted. Seven out of those eight developers are university graduates with qualifications ranging from three-year diplomas to four-year bachelor degrees in Information Technology and Computer Science or related fields. Two of those were, at the time of the survey, still in the process of completing their studies, while one developer dropped out.

The summaries of the findings in the initial problem analysis, which are described in much finer detail in [56], are given below. As far as the *individual testing skills* in company X are concerned [56]:

- The developers did not receive rigorous training in software testing at tertiary (university or college) level: The little training received by some of the developers was only a superficial overview of testing and was not explored in depth or practically exercised.
- No further training courses or seminars have been attended after their tertiary education was completed: This meant that none of the developers were properly trained in software testing in practice.
- No internal training programs were conducted by company X to further educate its developers.
- Unit tests, which are usually performed in the testing process, were not performed by six out of the eight developers: The main reason given was the lack of knowledge, which can be attributed to the first three problems.
- No test tools were used to assist the developer in the process of software testing: This also implies no test automation and tedious, repetitive manual 'debugging' work.
- Software testing as it is currently done is not integrated into the software process. Almost all of the developers in company X only performed their 'debugging' after the implementation phase.
- Six out of eight of the Developers in company X did not know that testing entails more than just the execution of code: To them, there did not exist any conceptual difference between testing and 'debugging'.

As far as the commitment of company X, especially in terms of its management, towards a proper software testing process is concerned, our initial problem analysis has identified the following shortcomings [56]:

- The managers of the company did not regularly read the scientific literature on software testing and did also not sufficiently inform themselves by other means (e.g. seminar participation) about the current state of the art in this field.
- The company did not provide developers with a company-wide testing policy: There were no testing objectives or principles or check-lists to indicate the view or objectives of the company about how to do software testing.
- No proper testing environments were given to the developers for testing: Developers used virtual machines on their notebook or desktop computers to perform their testing. These virtual machines were not properly managed, such that any test results were not sufficiently repeatable. They were mostly shared amongst several developers and thus did not present any pristine testing environment.
- No test templates, historical data, or documentation from previous software projects were provided: Developers sometimes created their own test documentation ad-hoc, which did not adhere to any proper standards.
- None of these test levels that comprise the testing process were clearly defined. The tasks that should be performed within each test level were also not defined by the company.

The combined effects of these shortcomings, both on the level of the individual employees as well as on the managerial level of company *X* as a whole, are quite typical:

- Far too little time (only about 10%) per project was dedicated to testing.
- The testing process itself could not be monitored because most developers had only sketchy documentation about their testing progress. No test results were available for future reference, as those sketches were not part of the deliverables of the project. There was thus no quantitative measurement of the current testing process.
- Moreover, there was a lack of planning on how the testing on a project should be performed. The testing was mostly done ad-hoc as 'debugging', and the decision on what needs to be tested was solely at the discretion of the individual developer.
- Few or no test cases were written, due to a lack of testing skills and in depth theoretical knowledge, such as mentioned by Amman and Offut [1]. This also implied that most code written for testing purposes formed part of the production code; in other words, there was no clear separation between production code and test code.
- Defects were not captured in a defect tracking system where they could be properly monitored, managed, and prioritised. Developers mostly wrote the defects they encountered on pieces of paper.
- There was no dedicated test team or testers available: all testing activities were solely performed by the developers themselves, with the usual 'developer bias' towards showing the absence of defects rather than their presence.
- Testing was thus systematically geared towards quick and shallow user-acceptance tests, which correspond to the requirements specification of a software system, whereas the deeper small-scale tests, which correspond to the software architecture and its units' levels, were too often omitted.

The following section discusses possible solutions to the concrete problems identified here.

3 MATURITY LEVELS IN TESTING

Small IT companies, of which company *X* is only one example, typically operate at a low or even very low level of maturity as far as their software testing processes are concerned. Test *standards* such as the international *IEEE829* or the British *BS7925-2*, already exist, but they do not provide sufficient 'progress guidelines' for sub-standard IT houses which wish to make progress *towards* the application of such standards. To be able to measure their process improvement progress, IT companies need *crite-*

CTP Criterion	*TMMi Category*	*TMMi Level*
Analysis of quality risks	Test monitoring and control	2
Test estimation	Test planning	2
Test planning	Test policy and strategy	2
Test planning	Test planning	2
Test team building	Test planning	2
Design and implementation of test support systems	Test design and execution	2
Design and implementation of test support systems	Test environment	2
Running and tracking of tests	Test monitoring and control	2
Running and tracking of tests	Test design and execution	2
Managing of bugs	Test monitoring and control	2
Discovery of test context	Test life cycle and integration	3
Test planning	Peer reviews	3
Test team building	Test organisation	3
Test team building	Test training programme	3
Running and tracking of tests	Non-functional testing	3

Figure 1: Mapping between Black's CTP Criteria and TMMi Categories at medium Levels of Maturity

ria against which to check the observable phenomena.

The *Test Maturity Model (TMM)/Test Maturity Model Integration (TMMi)* framework [46] is well-known amongst academics, but little known (if not even completely unknown) amongst small IT houses such as company *X*. On the other hand, the testing handbook by *Black* with its *Critical Testing Processes (CTP)* framework [5] specifically addresses industrial practitioners and may thus be assumed to be wider known amongst industrial software developers. Black [5] describes the CTP processes as *lightweight check-lists* and not /bureaucratic regulations. Together with Black's 20 years experience in the testing industry, not to mention the concise, practical guidelines on implementing a test process improvement project, made the CTP a good candidate for this study. Whereas the CTP can be seen as an *agile*, easy-to-implement framework, the TMMi follows a more *heavyweight* approach with a strong foundation in testing principles, best practices, and general testing goals. Their complementing factors made these two frameworks ideal candidates for this study. One of the *problems* in this context is that the CTP criteria provided by Black do not immediately let us see to which TMMi level (which we take as a reference framework) they correspond. Small IT houses such as company *X*, which usually start at the lowest possible TMMi level (1), would thus not know which higher TMMi level they would reach by following Black's criteria. To tackle this problem, we have described and explained a *mapping* [56] between the categories of TMMi and Black's criteria, which is summarised in *Figure 1*. This mapping takes only the medium TMMi levels, (2) and (3), into account because it is highly unrealistic for small IT companies to reach out for the higher TMMi levels, (4) and (5), which are often not even reached by large international industrial software corporations. It is worth noting that there are also several CTP criteria and recommendations in [5] which cannot be directly mapped to TMMi categories at the above-mentioned levels. Companies using Black's CTP criteria should also aim at applying for an official TMM/TMMi certificate. Although the CTP positively contributed to the improvement process, this study put more emphasis on the TMMi improvement model. The reason is that Company *X* preferred to align itself with a fast-growing industry standard.

The TMM/TMMi classification was modeled with the older, widely known CMM/CMMi classification in mind, after it had been discovered that even those IT houses which enjoy the benefits of a high CMMi level (in general) can still be 'pretty bad' as far as their software testing procedures (in particular) are concerned. In other words, a high CMMi level does *not* necessarily imply a high TMMi level as well. Purchase and procurement managers should be aware of the possibility of such a discrepancy between CMMi and TMMi, and should thus not be lured into purchase contracts merely on the basis of a high CMMi certificate.

An IT house at the lowest possible TMMi level can be characterised by the following description: "At TMMi level 1, testing is a chaotic, undefined process and is often considered as part of debugging. The organisation does not provide a stable environment to support the processes. Tests are developed in an ad-hoc way after coding is completed. The objective of testing at this level is to show that the software runs without major failures. In the field, the product does often not fulfil its needs, is not stable, or is too slow to work with. Within testing, there is a lack of resources, tools and well-educated staff. At TMMi level 1, there are no defined process areas. Products also tend to be delivered late, budgets are overrun and quality is not according to expectations" [46].

If this is the case in any IT house, whether big or small, improvements should be attempted as soon as possible. Many 'standard' solutions to those well-known problems can be found in the software engineering literature, but they must still be adapted to the situations in each case.

4 LITERATURE SOLUTIONS

In section 2, several problems of the case study object, company X, we summarised together with their software testing procedures, including their *meta* problem of not knowing that their problems are rather common or 'typical', and that the scientific software engineering and software testing literature already provides many quite easily applicable solutions to those kinds of problems. Therefore, in this section, we re-visit the previously identified problems in the context of company X and we point to several relevant papers (related work) the authors of which have also dealt with similar problems. Consequently, the papers mentioned in this section can be regarded as 'tools' for improving the software testing procedures in company X, which is a typical responsibility of IT management.

4.1 Testing Training and Education

All the developers in company X, except for one, had completed their tertiary education in Computer Science related fields. The developers commented in our interviews that little or no software testing principles had been taught to them at university. Indeed, according to [42] and [13] only a small portion of undergraduate Computer Science/Informatics curricula is allocated to the topic of testing. They recommended that more testing should be taught at tertiary level. Paper [20] agrees that university training on software testing is currently not sufficient. A study conducted by [34] indicated that 27 out of 65 participating organisations reported that less than 20% of their test team received training in software testing at university. Paper [34] concluded that there is either a lack of practical skills delivered to tertiary students in software testing courses, or a complete lack of software testing courses provided by universities. Paper [42] recommends that more testing be taught and that there be more communication between industry practitioners and academics to ensure that undergraduate curricula provide students with a solid foundation in software testing. These papers are all hinting at *long term* solutions to a widely recognised problem. In the immediate situation of company X, however, they are not directly applicable. Therefore, such IT houses must in the meantime go through the effort of sending their technical staff to commercially available courses and seminars, as they are often offered by consulting houses and other IT support companies. In South Africa, also the *Computer Society of South Africa (CSSA)* (especially its *SIGiST* special interest group in software testing) is active with public seminars in this field.

In this context a study conducted by [34] indicated that 47 out of 65 IT organisations provided opportunities for their testing staff to be trained in software testing. Among the organisations investigated in this study, 37 out of 65 organisations preferred commercial training courses, as opposed to 25 organisations that preferred internal training courses. Another study [19], conducted in Canada, indicated that 31% of the respondents received rigorous training on software testing. Paper [49] advises that small to medium enterprises (SME) should invest in test training courses for developers or testers. The 'International Software Testing Qualification Board' (ISTQB) provides professional software testing certifications in almost 40 countries across the world, including our country, South Africa.

The ISTQB syllabi are organised in three levels: 'Foundation', 'Advanced', and 'Expert'. The Foundation certification syllabus provides a solid introduction to software testing and is well suited for people entering the field of software testing. The Advanced certification syllabus provides a mid-level certification to practitioners with at least 5 years experience. Paper [6] suggests that prospective testers do not even have to attend expensive training, but can gain free access to the ISTQB syllabi from the ISTQB website and only write the exams at accredited testing centres.

The South African Software Testing Qualification Board (SASTQB) was recently established in 2008, as the 40^{th} National Board member of the ISTQB. There are at least three accredited ISTQB training providers in South Africa, and we recommended that all the developers from small IT companies, such as company X, attain the Foundation and Advanced level certifications.

4.2 Test Planning and Test Cases

During the time of this case study, testing in company X was not planned at all. Developers mostly started testing without any idea of how, when, or what needed to be tested. In the context of similar 'emerging' software organisations, the authors of [25] performed a study that analysed the current testing practices of a small software company. The results of that study also indicated that no upfront planning of the testing process was done. According to [25] this resulted in deadlines being overrun and increases in the project costs. The authors of [10] suggested that test planning is an essential component of a testing process as it ensures that the process is repeatable, defined, and managed. This view is also supported by [17], which states that test planning contributes significantly to improve the test process; also [5] has argued in this direction.

On the highest, most general level of test planning is the so-called 'Master Test Plan' (MTP). It provides an overall view of the test planning and test management of a project as a whole. According to standard IEEE 829, this entails activities such as:

- selecting the different parts of the project's test effort,
- setting objectives for each constituent part of the test process identified,
- identifying the risks, assumptions and standards for each part, and
- identifying the different levels of test and their associated tasks, as well as the documentation requirements for each level.

On the small scales of test planning we eventually find the task of defining *test cases* which can be regarded as 'micro-requirements' that must be fulfilled by the various software units under test (see also the following sub-section 4.3). Standard IEEE 610 defines a test case as a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a program path or to verify compliance with a requirement.

In our case study, the software developers at company X also did not design nor execute any explicitly defined test cases. This was mainly due to a lack of general knowledge in testing as well as a lack of guidance from a test policy and test strategy (see also sub-section 4.6). In the case study of [21] amongst twelve software organisations, nine out of them did not enforce the use of test case selection methods. It was left to developers and testers to decide which test cases were selected. In [16] it was argued that the absence of clearly defined test cases in a test process impedes the improvement of the test process. They recommended that templates be available that provides guidelines on the most common steps to create a test case. Another study, conducted for paper [24] investigated the test processes followed by developers amongst six organisations: it revealed that only half of those organisations created test cases for testing their software. None of those organisations in that study developed any test cases before the product programming began.

Test Driven Development (TDD) is a software practice whereby unit tests are defined before product program code is implemented. The code for the unit test is gradually added until a test is passed. In [18] it was argued that code developed using TDD had superior quality compared to code developed using traditional approaches such as the waterfall-like model applied in company X. The same study also claimed that 78% of the professional programmers interviewed believed that TDD improved their productivity.

On the other hand, there is also recent evidence which indicates that not all is well with TDD: In a comparative case study conducted amongst five small-scale software projects [43], it was found that "an unwanted side effect can be that some parts of the code may deteriorate. In addition, the differences in the program code, between TDD and the iterative test-last development, were not as clear as expected. This raises the question as to whether the possible benefits of TDD are greater than the possible downsides. Moreover, it additionally questions whether the same benefits could be achieved just by emphasising unit-level testing activities" [43]. This leads us now to the topic of *unit testing*.

4.3 Unit Testing

IEEE standard 610 explicates unit testing as 'testing of individual hardware or software units or groups of related units'; in Object oriented programming (OOP) a unit may represent a single method in a class, or the whole class itself. IEEE standard 1008 defines an integrated approach to structured and documented software unit testing in this context.

None of the developers in company X unit-tested their program code in detail. According to [23] [24] [33], however, programmers typically write unit tests as opposed to testers. Those papers also emphasise the importance of unit testing and state that it is the most efficient test to conduct due to its low granularity and high code coverage. In [24], a study conducted in six software organisations on the testing processes of programmers, revealed that even the definition of test process did not imply that programmers had a concrete 'recipe' for conducting unit and unit integration testing. According to [4], unit testing is seen as an essential phase in software testing. According to [33] [4], unit testing allows individual units to be inspected that can reveal faults early, which would otherwise be difficult to find in system or user-acceptance testing. In [4] it is also noted that unit testing is generally not performed well and sometimes not performed at all due to cost implications, as it was also the case with our study object, company X. In this context the authors of [49] recommend the following IT *management tactics* for improving the quality of unit testing done by programmers:

- *Force* developers to perform unit testing by emphasising that unit testing is their responsibility.
- *Motivate* developers by having regular meetings and seminars to share information.
- *Allow* developers the freedom to perform unit testing as they feel convenient: Thereby, allow them to experiment with different test tools and technologies available.
- *Provide* developers with the necessary resources: This includes *time* specifically allocated to the *study* of testing methods and to practice the learned methods in various exercises.

4.4 Test Automation

In our case study, company X did not use any test tools to assist in the automation of test tasks. In a study by [34] it was found that 44 out of 65 organisations used test automation tools. Interestingly, 30 of those 44 organisation thought that the use of automated test tools was beneficial. According to [34], the biggest factors against the use of test tools were cost, time and difficulty of use. This view is also supported by [14] who states that test tools can be very expensive and involve a lot of effort from the testers. They suggest that organisations may consider using test tools when they have mature test practices or have a large number of tests. Organisations may do well in software testing without the use of test tools, too [14]. Also in [29] it is said that using test tools could pose a problem for organisations having the following characteristics:

- Lack of well-defined testing process.
- Ad-hoc testing.
- Organisational culture does not support and embrace testing.

These three issues were relevant for company X too, as they had no structured testing process, their testing activities were ad-hoc and the company had not yet embraced a 'culture' of software testing.

Although testing tools and automated testing lighten the workload by eliminating repetitive manual tasks, the difficulties identified by the literature suggest that manual testing will be sufficient for small companies, such as company X for the time being. In any approach to automated software testing one must also take into account the possibility of the so-called '*Heisenbugs*': these appear to be (pseudo) 'defects' which are caused by the test apparatus used for automated software testing, not by the software under test itself. Because of the possibility of those 'Heisenbugs', automated software testing must never be conducted naively: the testers need to know their test apparatus well, and they also need to have some theoretical knowledge about the phenomenon of 'Heisenbugs', their causes and their potential effects.

4.5 'Static Testing' by Program Code Reviews

The only form of testing being conducted in company X during our case study was 'dynamic' testing which is defined by the *ISTQB* [47] as testing that involves the execution of the software of a component or system. This implies that code must already be available before testing can commence. 'Static' testing on the other hand does not entail the execution of code, but refers to activities such as program code reviews.

A study conducted on software testing maturity in the Korean defence industry [37] indicated that static test techniques were identified as having a low maturity on the test process improvement (TPI) maturity matrix [27]. A study conducted by [24] on the testing processes in six software organisations revealed that only two organisations conducted code reviews against company coding standards. Only two organisations conducted peer reviews and only three organisations did inspections. Although the sample size of this study was small, it might be regarded as 'anecdotal evidence' for the lack of static testing techniques in many software organisations.

In [24] it was also recommended that organisations encourage the application of static testing alongside dynamic testing. Similarly, [33] mentioned reviews of functional requirements and design as one two the key factors that indicate an organisation's test maturity. There it was said that functional requirements reviews improve the correctness of requirements even before the software is designed. In terms of design reviews it was argued in [33] that conducting reviews of the design can help to eliminate flaws in the design before coding commences.

In a recent study conducted by [16] amongst ten software organisations to determine which factors impede the improvement of testing processes, the establishment of review techniques ranging from informal peer reviews to rigorous code inspections was suggested as an approach to improving the testing process. In [30] it was argued that reviewed documents can result in a 65% reduction in defects that could otherwise have been carried forward to later phases of the development cycle.

For small IT companies with limited resources (such as company X in our case study) we would recommend on the basis of our experience *three* reviews throughout the software cycle:

- The first review should be conducted during the requirements phase to try to ensure that all specified requirements are testable.

- The second review should be conducted after the design phase to try to ensure that the design does not contain flaws which would propagate to the programming phase.
- The third review should be a program code review held during the implementation phase to try to ensure that developers are programming according to company coding standards and to identify possible weaknesses or bad programming practices.

4.6 In-House Guidelines: Policy and Strategy

Several of the developers in company X indicated in our semi-structured interviews that they did not know how to perform testing, or what is expected of them in this regard. They also indicated that the company did not provide them with in-house guidelines on how the IT management would want their testing to be conducted.

According to [48] a *test policy* answers questions such as 'Why do we test?', 'How important is quality?', and 'Are we time-driven or quality-driven?'. It was further suggested in [48] that a test policy forms a critical basis for commitment to all test-related activities in an organisation. Similarly it was suggested in [38] that an explicitly defined test policy should provide a framework for planning, organisation and execution of all test activities within a company. The author also warns that if a test policy is only given implicitly it can have the following adverse effects [38]:

- Testing is not consistent over different departments: This will cause the testing approach to be defined individually on projects and the results will not be comparable between projects.
- The test policy will not be recognised and adhered to by the people involved.
- Managers will struggle to properly track a project's testing status, such that performance will differ based on the testing approach followed individually.

For companies such as our case study company X we therefore suggest that the test policy be one of the first improvements to be introduced. Any test policy will emphasise the importance of testing and software quality. This will create an awareness and sense of responsibility amongst its employees, especially the software developers, that testing is viewed as a critical part of the software cycle.

In contrast to a test policy (as mentioned above), a *test strategy* is defined by the ISTQB as a high-level description of the test levels to be performed and the testing within those levels for an organisation. According to [21] a test strategy thus provides an organisation with responsibilities within each test level and also advises the organisation on how to choose test methods and coverage criteria in each phase. There was no such test strategy in company X to guide developers on how to perform testing at the time our case study was conducted. Interestingly, from the study conducted by [21] only four out of the twelve organisations under study used explicitly defined test strategies. Three organisations had implicitly defined strategies that formed part of some standard regulating the testing aspects of a product.

Another test process maturity study [24] conducted in 2007 amongst six software organisations indicated that there was a definite lack of testing guidelines, too. This study argued that developers did not know what to test or how to test and provides an ideal opportunity for implementation of a test strategy. According to [48] a test strategy is based on an organisation-wide test policy; thus test policy and test strategy are closely related to each other. The strategy must be aligned with the organisation's software development process as various test activities and procedures are defined for each phase of the software development process. These activities typically include user acceptance testing planning at the requirements phase, system testing planning at the architecture phase, integration testing at the design phase, and unit test design and execution during the implementation phase; (see the well-known V-model for comparison).

4.7 Test Specialists

Closely related to the issue of in-house guidelines (test policies and test strategies) is the question whether or not an IT house employs specialist software testers which are not also playing the role of software programmers at the same time. There is no 'one-size-fits-all' solution to this problem; for some organisations or project it might be better to keep testers and programmers completely separate, whereas for other organisations and projects it might be better to have programmers and testers 'mingled' or in flexibly interchangeable roles. This also depends on the maturity level (see section 3) of the organisation; anyway the in-house guidelines must clearly answer this organisational question and should not leave it dangling in an undefined limbo state.

At the time of our case study, company X did not employ any dedicated software testers, and all test-related activities were performed by its software developers. As it was also recognised by [15], small companies have limited resources and therefore do not (or even cannot) invest in separate quality assurance staff. Another software process improvement study conducted in a small IT house [3] also indicated that not enough resources were available to establish separate quality assurance group. This coincides with the view of [32]. In the already mentioned study [24], however, five out of the six organisations *had* independent test teams. This study, however, did not specify the size of the organisation, but indicated that some of the organisations had up to only twenty developers; the ratio between developers and testers indicated that the organisations had between one to four developers per tester. Company X at the time of our case study had only eight developers.

In [25] it was found that having a dedicated test role in projects allowed for testing to become more structured as well as allowing developers to focus more on fault finding at the unit level than at the system level of testing. A survey conducted by [34] in Australia regarding software testing practices indicated that 44 out of 65 organisations investigated had an independent test team indeed. Also [49] recommended that even small to medium sized enterprises (SME) should employ at least one person that is dedicated to testing. If this is however not possible, then one may also follow [40] which recommended that some developers in the team be assigned the role of testers temporarily. This would imply that the same developer may not tests his own program code such that there is some degree of independence between development and testing. However, if an IT house wishes to reach TMMi level 3 (see section 3 above), then dedicated testers must be members of a separate quality assurance group.

4.8 Test Timing: Early versus Late Testing

Company X in our case study followed the 'Waterfall' approach to software development with the testing phase being situated between programming and deployment. This practice is in contrast to newer software engineering practices. A study described in [25] on testing in emerging software organisations (similar to company X) revealed that testing was mostly being performed too late in projects. In [28] it was argued that *testing has a life cycle of its own* and must therefore run concurrently with the software development process. They, too, suggest that testing must start already during the requirements phase of a project. This is also emphasised in [31]. According to [45], between 20% of the testing effort can be spent on planning, 40% on preparation and the remaining 40% should be spent on test execution. This clearly implies that testing cannot occur only at the end of the software development life cycle. In [30] it was suggested that testers be involved in reviewing requirements documents as well as design documentation. According to [30] [47] defects found early in the software life cycle will cost much less to fix than the ones detected in the later stages of the development process.

Theoretically, not much testing will be needed if software was properly and formally specified at the beginning of the project cycle, but practice usually differs from theory. As a 'rule of thumb' one can say that about 50% of the total project time should be invested into testing [23]. This means: if in some project less than half of its time is spent on testing, then it is most likely this project itself is in dire straits (and very unlikely that specifications at in the beginning of the project had been done so exceptionally well that hardly any testing is needed in the end any more.) This further implies, typically, that when a project runs out of time and goes beyond schedule, which most projects actually do [8], then the most important activity of testing is usually the very first victim to be sacrificed at the altar of the deadline cult. In our case study at company X we found that less than 10% of the average total project time was actually spent on software testing, which is five times less than recommended. For comparison, also [25] found that testing activities were given too little time and too few resources in a project, whilst [24] found, too, that developers did not pay enough attention to testing, again mainly due to a lack of available project time.

Business people often say that 'time is money', thereby hoping and believing to save money by saving project time. For this short-sighted dogma very high prices will have to be paid later, when previously undetected software 'bugs' (due to hasty testing) will creep to the daylight after deployment and have to be eliminated at even higher costs. For this problem, however, we do not have any technical solution in our software engineering tool box; this is a problem of the business mind. All project stakeholders, especially on the non-technical side, need to understand in their minds that quality requires time; it is indeed the responsibility of the software engineer, in his role as the technological expert, to warn the businessman on the project against any unrealistic presumptions or wishful thinking on the basis of a short-sighted 'time is money' dogma.

Of course we will never have an infinite amount of testing time available and we thus need to allocate our available time as wisely as possible; this implies that we should also have some reasonable *priority declarations* in place which state what parts of a system under test should be tested first with highest priority, and what parts of the system can be tested later with lower priority.

4.9 Test Priorities and Critical Tasks

Related to the question of test timing is the question about which tasks to do first, with highest priority, in an available frame of time. Our ability to assign such test priorities appropriately, however, depends on our depth of understanding of the 'most critical' parts and 'bottle neck' sections of the software system under consideration. This requires skills, and tools, for the analysis of the *structure* (and structural complexity) of the software system before the various test tasks can be prioritised. In our case study, the software developers in company X indicated that they did not know what to test nor how to prioritise their test activities to test the most 'complex' program code first, or the code most likely to contain defects, or those highly connected sections of the code in which a defect would have the most far-reaching 'ripple effects'.

In order to address such problems, [44] recommended static code analysis and *software visualisation* techniques. Static analysis is a fault-detection support technique that aims to evaluate a system or component based on its form, structure, content, or file documentation, whereby the component or system's code itself is not executed [24]. Tools are available to automatically perform static analysis and we refer to these tools as Automatic Static Analysis (ASA). ASA tools automate the detection of flaws and poor code constructs by parsing the source code and searching for certain patterns in the code [55]. According to [44] a typical pipeline for a static analysis tool includes the following components:

- *Parser*: analyses the raw source code and produces and low-level representation thereof. The source code is usually represented as a syntax tree.
- *Query engine*: checks the source code for certain facts by scanning the syntax tree for specific patterns. These patterns include showing variables that are used before they are initialised, code modularity, and cyclomatic complexity to name a few. Thereby, *cyclomatic complexity* is the measure of the complexity of a software module, and a module is defined as any single method or function that has a sin-

gle entry and exit point [54].

- *Presentation engine*: represents query results graphically to show the patterns uncovered by the query engine.

The authors of [44] integrated their own query engine with SolidFX, an existing Interactive Reverse-engineering Environment (IRE). This IRE has the same look as modern Integrated Development Environments (IDE) such as Eclipse or Microsoft Visual Studio. They applied their query engine together with SolidFX and Call-i-Grapher to a number of industrial C and C++ projects. Thereby, their tool revealed the following interesting characteristics:

- Complexity 'hot-spots' based on metrics such as McCabe's cyclomatic complexity, and lines of code commented;
- The modularity of a system and its constituent modules based on its call graphs;
- The maintainability of a system based on specified metrics such as lines of code, lines of commented code, and cyclomatic complexity.

Similarly it was argued in [54] that cyclomatic complexity should be limited because complex software is harder to maintain, harder to test, more difficult to comprehend and more likely to cause errors. Since such software visualisation tools are commercially available and specifically designed for industrial application [44], we recommend that also small IT houses should make themselves familiar with them. High test priorities can then be given rationally to the 'hot spots' identified by those scanner tools.

4.10 Test Repeatability and Environments

Experimental repeatability is one of the key characteristics of the *scientific method.* This also holds for software testing, if we regard a software test run in analogy to a 'scientific experiment'. One of the methodological preconditions of scientific repeatability is the conduction of experiments under well-defined, isolated and well-controlled *laboratory conditions.* This is also true in the area of software testing, where isolated and well-maintained 'test environments' resemble a scientific laboratory. Though it is well known that not all software testing experiments are repeatable in this strict sense —distributed and concurrent systems notoriously display nondeterministic behaviour, because of the wide network environment on which they run which is typically beyond our control— most of the 'ordinary' software testing experiments can indeed be made repeatable if only those well-controlled laboratory conditions are fulfilled.

In our case study, on the contrary, the test environment available at company X was found not adequate to support their different software projects in a scientifically acceptable way. The developers in our case study mostly used virtual machines running on their personal notebook computers to perform testing, which means that the 'test laboratory' was not carefully controlled. Indeed, due to the costs of hardware and support software tools, small organisations often do not have the means to obtain and maintain dedicated test environments [32]. In the already mentioned study [24] conducted between six organisations on their implemented testing processes, one of the difficulties listed was that of creating test environments that closely represented production environments.

The author of [33] also developed a set of twenty questions to determine the testing maturity of an organisation: The first question in his questionnaire enquired whether an organisation consistently creates an environment for development and testing that is separate from the production environment. He argued that the benefit of such a separate testing environment allows an organisation to avoid making changes to production environments ad-hoc, as well as the possibility to develop and test software completely before it is deployed in a production environment.

Although a test environment is an important part of software testing as identified by the literature, it is not always possible (or practically feasible) to use a pristine test environment in and for every project. The main reason for this is cost. Nevertheless it should be possible also for small IT houses to exercise better control over their available resources, such that the development environments and testing environments are kept separate even if they are running on the same hardware devices, and that the scientific 'laboratory conditions' (for the sake of repeatability) are at least 'approximated' as far as possible.

It is indeed the responsibility of the IT management in a company to fully appreciate that software *testing* is de-facto the *most scientific* of all software engineering activities, whereas other software engineering activities still resemble much of an 'art' or 'craft' [35]. However, the 'power of science' cannot be unleashed if IT managers are not aware of it.

4.11 Test Documentation

During our case study, little or no test documentation was created in company X. The few test plans that have been created were not done according to any industry standard such as, for example, IEEE 829. A study conducted in Finland [41] about software process improvement in small companies also indicated that there were problems with software test documentation; however that study did not elaborate on the reasons for this problem. The authors of [36] too assessed the software process of a small company and found that no written test plans or reports existed. According to them this lack of documentation caused difficulty in reproducing the defects in order to eliminate them. It was also noted in [36] that once these defects were fixed, the problem and solutions were never recorded at all. This caused the development team to investigate similar problems from scratch again and again, without having access to past documentation for historic orientation.

The IEEE 829 standard provides guidelines for software test documentation which can be used as templates in various testing projects. Those are not too difficult to handle and should thus also be suitable for small IT houses with short resources. There are templates for a 'Master Test Plan' (MTP), 'Level Test Case' (LTP), 'Level Test Log' (LTL), 'Anomaly Report' (AR) as well as a 'Master Test Report' (MTR). Those templates should be stored in the document management system of an IT house such that all relevant employees have access to them. The use of proper test documentation must also be emphasised in the test policy document of a company, and must —last but not least— be proactively enforced by its IT management.

4.12 Test-Related Documentation

A *test basis* is defined by the ISTQB as "*all documents* from which the requirements of a component or system can be inferred". Here it becomes obvious that testing is closely linked to the entire software project and cannot be seen only in isolation. Indeed, the author of [7] argued that insufficient test cases could be attributed to the low quality of the test basis. The software developers of company X in our case study also argued that better requirements specifications be created so as to improve the testing process. Although the requirements specification is strictly speaking a software development process component and not a test process component, it has a direct effect on the quality of the test effort.

According to [15], small organisations do typically not have clearly defined and stable requirements on software projects either. In a study [41] conducted in 2007 regarding software process improvement (SPI) in a small companies in Finland, one of the problems identified was that of requirements documents that were not good enough. It was found that structured documents such as requirements documents were created on projects, but their contents and depth were deficient. A possible reason for the haphazard requirements documentation might be attributed to a lack of research in requirements engineering for small companies. According to [2] there are not many papers published in requirements engineering conferences that addresses the requirements processes of small IT houses specifically. The authors of [2] further argued that this may be due to mistaken assumptions that small companies are no different than their larger counterparts or that small companies do not present noteworthy research challenges. Although not referring specifically to requirements engineering or testing, also paper [53] emphasise that not enough publications exist that provide software engineering solutions specifically for small companies.

Anyway it is obvious that requirements engineering must be done properly in support of test engineering, and it is also clear that the existing 'general' literature on requirements engineering is relevant until more 'special' results become known on requirements engineering for small IT houses in particular.

4.13 Test Tracking and Recording

Related to the issue of test documentation is the issue of test tracking and recording (in data bases) for future reference, with the purpose of learning the lessons from past experiences. Recording the history is also a precondition for being able to gauge the success of any process improvement measurements being undertaken. For example: if we are not aware which ratio r of our code units are currently defective (long term average), and we are now enforcing a new test policy P which leads to a new average ratio r' of defective units, then we cannot even tell if the introduction of P was a successful measurement in terms of $r > r'$. This was at the time of our case study the situation in company X.

The reason for this lack of measurability was the absence of any defect tracking system at company X at the time of our study. Once a developer at company X detected a defect, he would either fix it immediately (and then forget about it), or try to remember the defect for later fixing, or scribble a private note on a piece of office paper, which would sooner or later land in the litter bin, too. On the contrary, [33] suggested that a defect tracking system allows an organisation to assign ownership to defects and to store the knowledge about defects, until they are resolved, and even beyond. According to [33] can also indicate the status of the test effort by reporting on the number of defects found, number of defects resolved, as well as the yet to-be-fixed defects. These metrics correspond well with those suggested by [25]. Defect tracking system such as 'Bugzilla' or 'Track' are easily accessible defect tracking systems, freely available from the Internet, such that even small IT houses with limited financial resources can —and should— take advantage of them.

The already mentioned Anomaly Report template from standard IEEE 829 can also be incorporated into a defect tracking system to ensure that all necessary defect data are captured for future reference. The defect tracking system will also automatically store some of the test metrics that can be used to monitor the test process progress.

According to [28], metrics are measurements, collections of data about project activities, resources and deliverables. Paper [28] can be used as 'tool' to assist in the estimation of projects, measure project progress and performance, as well as quantifying project attributes. The authors of [28] recommended that the following test metrics be tracked in every project:

- Number of test cases,
- Number of test cases executed,
- Number of test cases passed,
- Number of test cases failed,
- Number of test cases under investigation,
- Number of test cases blocked,
- Number of test cases re-executed.

In addition to these, we recommend to compare also the *efforts* of software testing versus the efforts of having to deal 'a-posteriori' with customers' complaints about buggy software after product delivery and deployment: We assume that any improvement of an IT house's testing procedures is in the first place an 'investment' on the negative side in the books of that IT house; but the return on investment later, in terms of 'customer happiness', might well be worth the effort. Last but not least we recommend that also the *time* spent on testing should be recorded in such data bases for future reference.

A study conducted by [34] in Australia indicated that only 38 out of 65 of the organisations investigated used test metrics for such or similar purposes. 21 out of those 38 organisations agreed that metrics improved the quality of the software developed. Similarly, [7] also advises that metrics are an important part of the test process as it provides valuable insight into the progress of test projects. Company X did not work with any kind of metrics and was therefore unable to make reasonable estimations on future software projects. In this context [7] also argues that metrics provide a basis for test estimation and help to identify process improvement opportunities. On the other hand, the authors of [48] have warned against using too many metrics when embarking on a test process improvement initiative, as this could demand too many resources for monitoring process.

It should be emphasised that the combination of numeric metrics and historic records, especially in the area of software testing, is likely to make future software projects *more predictable* for project planners. An 'emerging' IT house with few resources, such as company X, should wisely start with only a small number of metrics applied, and then stepwise increase the amount of quantitative measurements being taken while the testing process of the company matures. (The path from 'qualitative understanding' to 'quantitative measurements' has indeed been walked by various scientific disciplines, though at different pace and velocity, during the history of science.)

5 TEST WORKFLOW IMPROVEMENT

The first step in improving the test workflow in a small IT house, such as our case study company X, is to gain clarity about what workflow *implicitly* exists in such a company which does not have any workflow explicitly defined. On the basis of the interviews which were conducted with the employees of company X [56], we were able to extract a de-facto existing workflow which is shown in Software and Systems Process Engineering Metamodel (SPEM) notation in Figure 2. The 'issues' attached to the process nodes in Figure 2 are the problems which we have described and discussed in section 2 of this paper; for more details refer to [56].

The SPEM diagram of Figure 2 shows intuitively that this is not the picture of a well-organised test workflow. It is the picture of adhoc activities in which there is no methodological distinction between proper testing and mere 'debugging'. The document-like icons represent work products that serve as input or as deliverables of the process. The input work product depicted in Figure 2 is a requirements specification. This document is used by developers to determine the functionality that needs to be implemented. The icons with sharp edges pointing to the right, represent tasks in the process. Developers would use the requirements specification to implement a feature. They would then proceed to build and run the code artifact for this feature to determine whether it performs the intended functionality. In the case of a failure, the developer would proceed to find the fault that caused it. This code-and-fix cycle continues until all functionality has been implemented and no more *apparent* bugs exist. The software product is released to the customer at this stage for further testing by the customer. One should also note that the human-like icons in Figure 2 represent developers. No other human role players are involved in test process and it is completely up to the developer to determine whether the software is ready for release.

The reason why we have taken this de-facto workflow representation is threefold:

- On the *small scale* it enables us to pinpoint 'issues' at each individual node in this process diagram.
- On the *large scale* it enables us to 'see intuitively' the problems with such a workflow as a whole.
- The visual representation of the existing test processes in Company X (workflow) aids understanding and provides a better platform to discuss the 'issues' with the company's non-technical managers, who are ultimately responsible for supervising and enforcing the improvement of the workflow to be applied.

The workflow as depicted in Figure 2 has been somewhat simplified for the sake of clarity; the figure does not include all problems identified in [56]. Also note that the figure only shows the *test* workflow identified in our case study, not the entire software engineering workflow. Such a representation however, would have revealed that the test workflow was unfortunately *isolated* from the other software development phases, instead of being reasonably well *integrated* with them. [1] warns against the isolation of the test process into separate stages of the software lifecycle, and recommends that testing should run parallel to im-

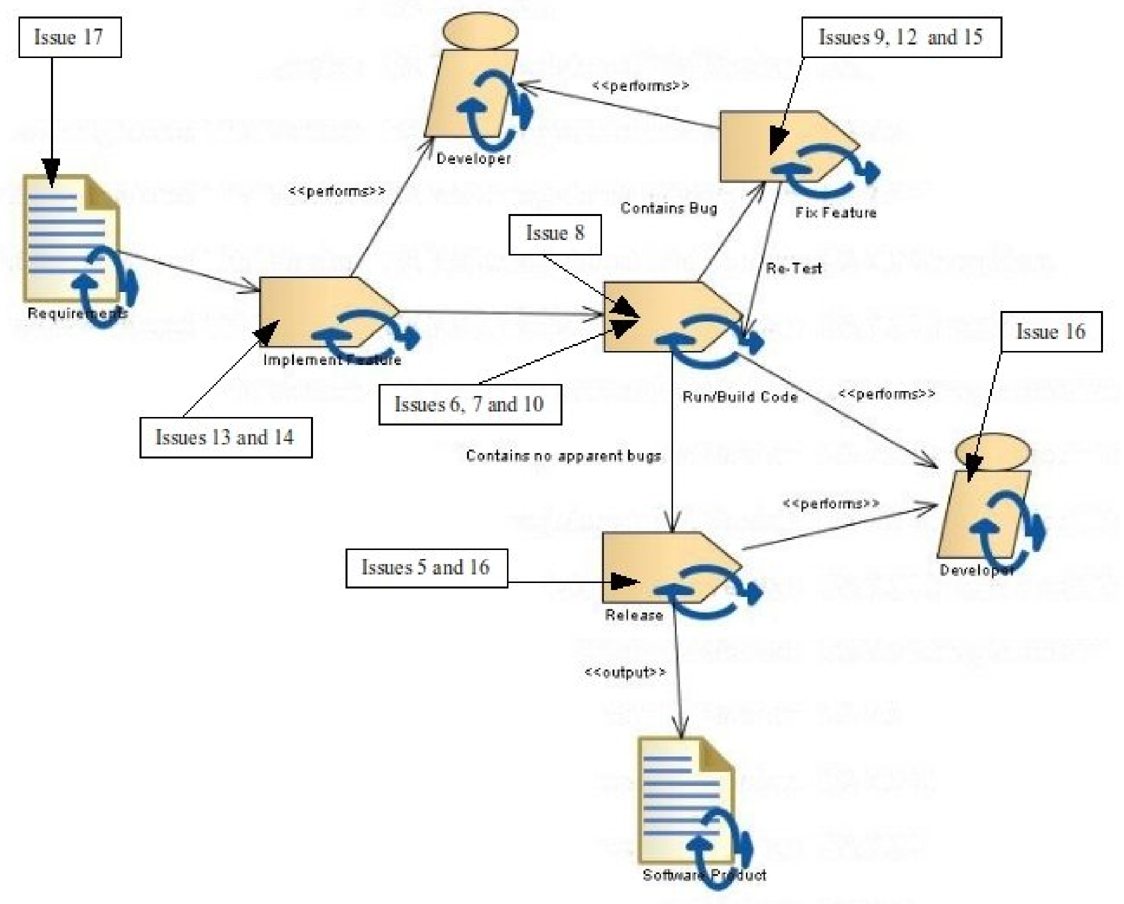


Figure 2: Implicit De-Facto Test Workflow, extracted from Interviews conducted with Employees in Company *X*.

prove the quality at each stage.

Anyway, the de-facto representation taken of an immature test workflow should be the starting point from which a company's IT management can launch its process improvement exercise. As explained above, and as further elaborated in [56], such a test process improvement exercise should take into account the following items:

- Test Policy,
- Test Strategy,
- Test Planning,
- Test Design,
- Test Execution,
- Test Monitoring (Control),
- Test Recording (Storing),
- Test Reporting (Closure).

For each of these activities, further micro-workflows could be defined, depending on the size of the company, its available resources, as well as the size of the project and the number of personnel involved:

- If the company and its projects are only small, then it would be a costly 'managerialist overkill' to define further rigid micro-workflows for those items.
- On the other hand if a company and its projects are larger, then it might well make sense to further increase the precision of the overall testing workflow definition by introducing micro-workflow definitions for those items on a smaller scale.

It, however, must be taken into account that, according to the well-known V-model, different forms of testing correspond to *each* activity

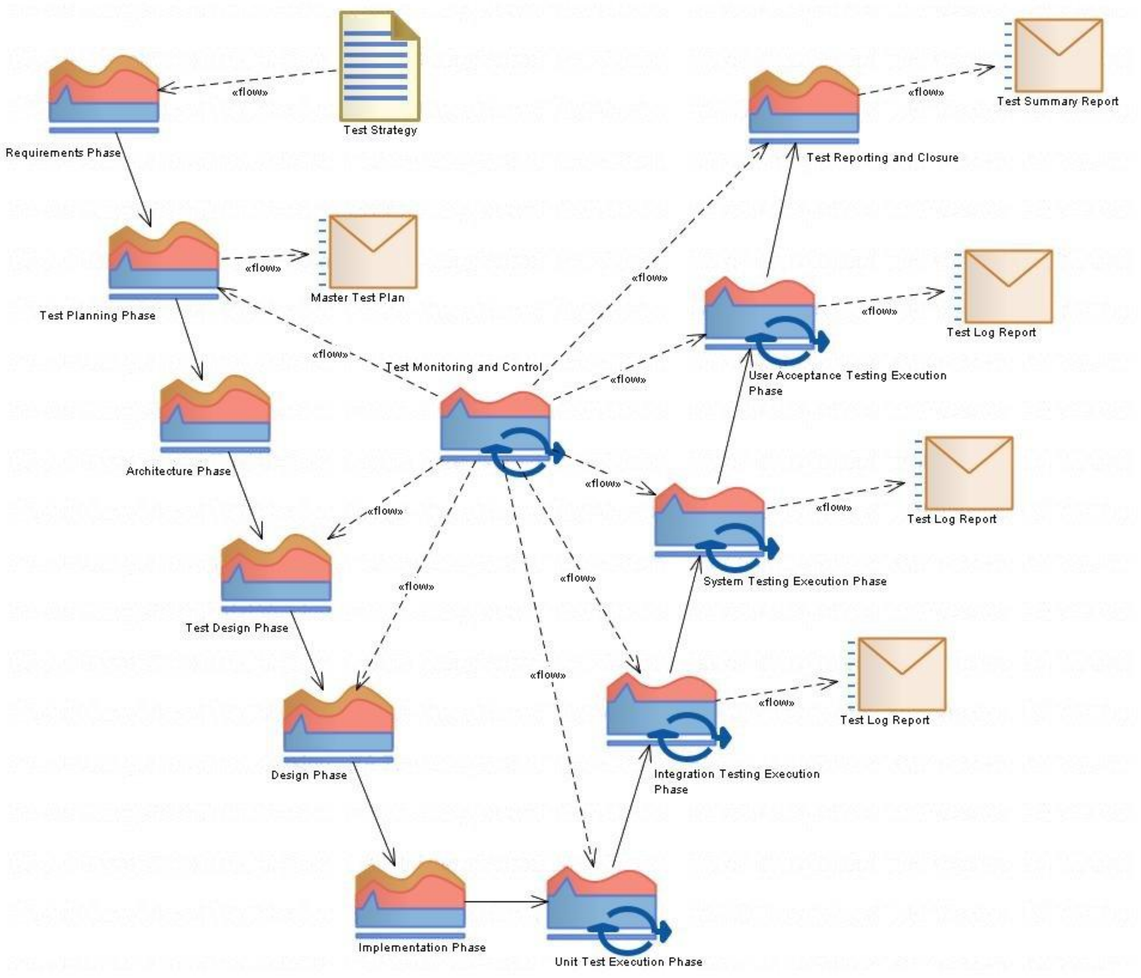


Figure 3: Improved and explicitly defined Test Workflow for Company X, including consideration of the V-Model.

on the software development side: Unit testing corresponds to the activity of programming in the small, integration testing corresponds to the level of software architecture (programming in the large), user acceptance testing corresponds to the activity of requirements engineering, and so on. This can be clearly seen in Figure 3, which depicts an improved testing workflow which we had suggested to our case study company X on the basis of the preceding analysis as further described in [56].

The picture of Figure 3 does not show in all detail that we have also suggested some clearly defined activities and deliverables (represented by the 'envelope' symbols in the picture) for each phase of the new testing process . Many of those suggested solutions for the improved workflow were indeed derived from the process areas of the TMMi model, according to our motivation of increasing the TMMi certification level of small IT houses such as company X. Therefore we should now also reason about the TMMi maturity of the improved workflow, to determine how it would compare to the old workflow maturity (which was at the lowest possible level of the scale).

6 PREPARATION FOR TMMI

With the development of the improved workflow, as shown in the previous section and as further explained in [56], we carefully followed the recommendations made for TMMi level 2. We attempted to include all of those process areas into our new workflow for company X, and this was generally achieved. However, the TMMi is a very comprehensive model with many specific and generic goals, which in turn contains even more specific and generic practices. We attempted to select the most important goals and practices from each process area for company X. It is currently not possible to fulfil all TMMi-2 requirements in each process area, because such would just add to the already heavy laden process improvement project. We therefore suggested in [56] that more goals and practices be added at later stages as the test process in company X becomes stable and more mature.

Especially as far as the TMMi-2 area 'Test Monitoring and Control' is concerned, we could not recommend to company X to fulfil all those requirements at once; such would have overstretched the company's capacity. We presume that this would also be true for many other small IT houses which 'play' in the same 'league' as company X. In fact the TMMi-'Test Monitoring and Control' process area contains 17 specific sub-practices. In our case study we have selected only a few practices to add to our improved workflow due to two reasons [56]:

- The first is that company X is only just beginning with its test process improvement program which needs to be rather simple and relatively easy to implement. We did not want to introduce too much complexity into the process at this early stage, but rather introduce new practices over time.
- The second reason for limiting our monitoring and control phase to basic practices was that company X did not have enough human resources to manage the implementation of these practices.

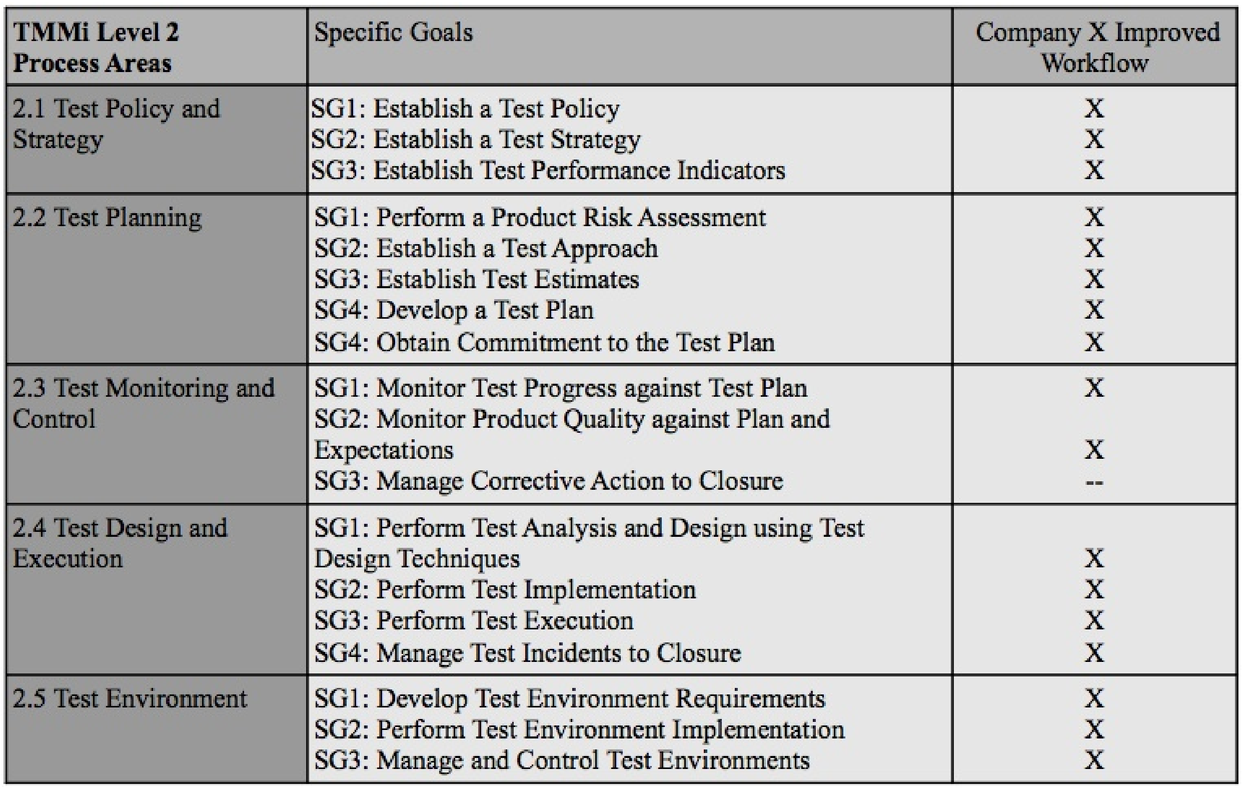
Based on these suggestions, we can however still ascertain that the improved workflow meets the requirements of this process area as at least two of the specific practices are met. This is shown in Figure 4.

As far as the approach to TMMi level 3 is concerned, our case study company X turned out to be too poorly equipped to make this goal realistically achievable in the near future; the same will most likely be true for many other small IT houses 'playing' in the same 'league' as company X. Nevertheless it turned out that company X seemed to be prepared for a *partial* approach to TMMi level 3, namely in the area of 'Test Life Cycle and Integration' (with two sub-goals seeming to be achievable) as well as in the area of 'Peer Reviews' (ditto). This is shown in Figure 5.

In summary, we found that the TMMi is very comprehensive and demands that many practices be implemented in each process area before that process area can be satisfied. We have decided that not all these practices and goals in each process area would be beneficial to the suggested new workflow of company X at the time of our case study. The main reason for this decision was that the implementation of all these practices and goals is a large task that could not be handled with the limited the resources available in company X at the time of the study. We therefore recommended that an independent team be established in company X to take on this task in 'baby steps'. This will be necessary should company X decide to submit their test process for official accreditation with the TMMi Foundation in the future. For purposes of future research we also recommended that the suggested workflow would be implemented as is, to first establish a structured process. Once the test process has matured and is fully integrated into the activities of the software development division of company X, we would recommend that the specific goals still missing (Figure 4, Figure 5) should be tackled. Still the question arises, how much time and effort would be needed to lift a small IT house, such as company X, to TMMi level 2 from the lowest possible level. This question shall be addressed in the following section.

7 TRANSITION EFFORTS

Figure 6 provides a 'to do' list with recommendations for our case study entity, company X, as further explained in [56]. If all these items would be done, then company X would have lifted itself from TMMi level 1 to *almost* TMMi level 2; let us say for the sake of illustration: 'TMMi 1.75' (if such fraction values would ex-



TMMi Level 2 Process Areas	Specific Goals	Company X Improved Workflow
2.1 Test Policy and Strategy	SG1: Establish a Test Policy	X
	SG2: Establish a Test Strategy	X
	SG3: Establish Test Performance Indicators	X
2.2 Test Planning	SG1: Perform a Product Risk Assessment	X
	SG2: Establish a Test Approach	X
	SG3: Establish Test Estimates	X
	SG4: Develop a Test Plan	X
	SG4: Obtain Commitment to the Test Plan	X
2.3 Test Monitoring and Control	SG1: Monitor Test Progress against Test Plan	X
	SG2: Monitor Product Quality against Plan and Expectations	X
	SG3: Manage Corrective Action to Closure	--
2.4 Test Design and Execution	SG1: Perform Test Analysis and Design using Test Design Techniques	X
	SG2: Perform Test Implementation	X
	SG3: Perform Test Execution	X
	SG4: Manage Test Incidents to Closure	X
2.5 Test Environment	SG1: Develop Test Environment Requirements	X
	SG2: Perform Test Environment Implementation	X
	SG3: Manage and Control Test Environments	X

Figure 4: Assessment of Improved Workflow against TMMi Level 2.

ist in the classification).

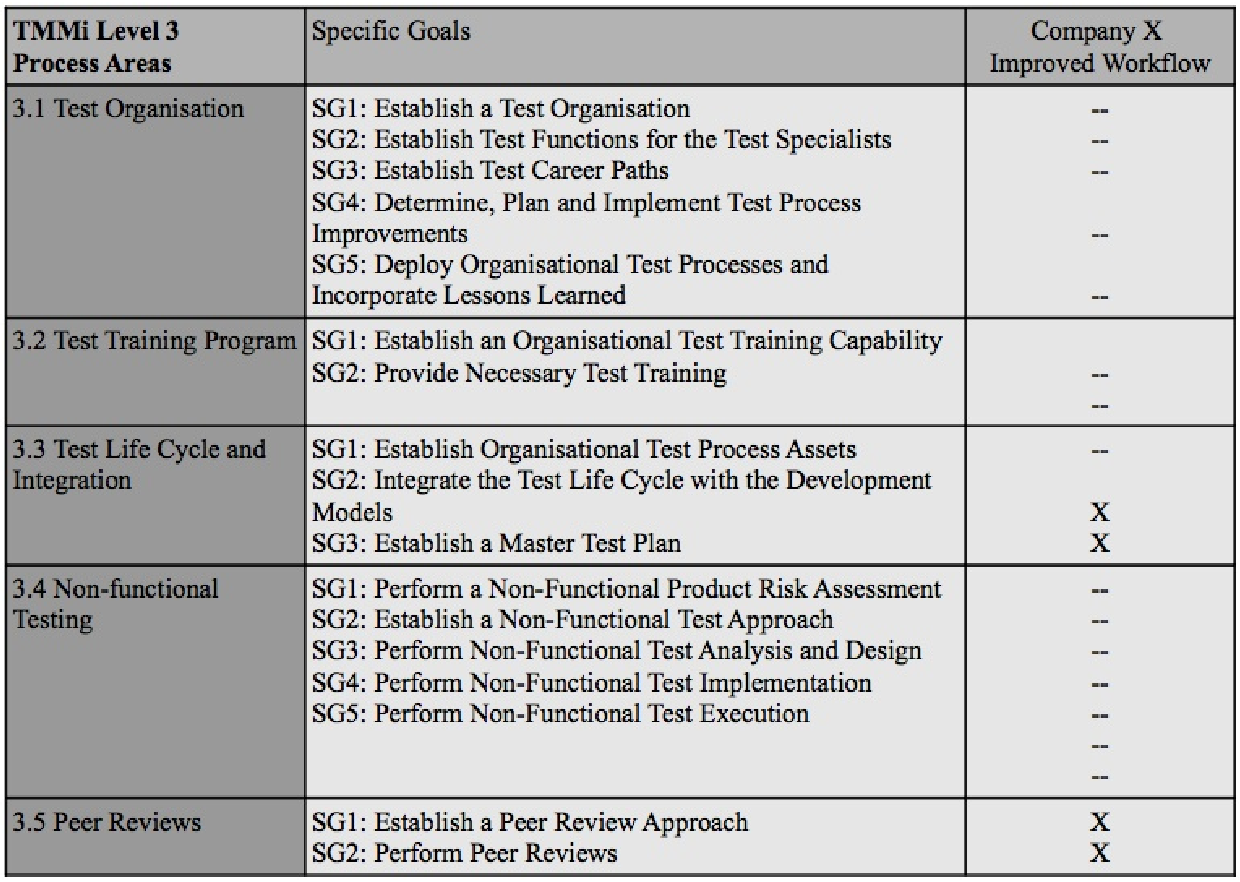
How much effort would it take to make this transition? Clearly this cannot be a trivial one-week task, and indeed [48] suggest that the transition phase could take *up to* two years to reach TMMi level 2. Company X, however, had really no established test practices at all that could be 'improved' or built upon, and therefore we estimate that this effort might take *at least* two years to complete for an IT house such as company X. It was also suggested in [48] that the following phases should be gone through as a preparation for TMMi certification in an organisation:

- initiation,
- planning,
- implementation,
- deployment.

These four phases will be described in the following, which should also explain the long duration (two years) of the whole transition as mentioned above.

The initiation phase is used to determine the current maturity level of the organisation. Based on the results, recommendations must be made to management. Fortunately this research has already analysed the current test process and has already made recommendations. We therefore suggested in [56] that the results of this research be presented to the management of company X. As part of the initiation phase, common goals must be set by the organisation on what they want to achieve with this improvement project. These goals can include predictability, higher productivity, efficiency, and effectiveness [48].

The planning phases involve the creation of a project team that will be responsible for the improvement project. A common problem seen by [48] is that organisations do not invest enough resources and attention into an improvement project. Each of the recommendations is assigned to a workgroup that implements and deploys the recommendations. We recommended in [56] that company X should establish a steering committee with a test project improvement manager. Specific time and resources must be allocated to the

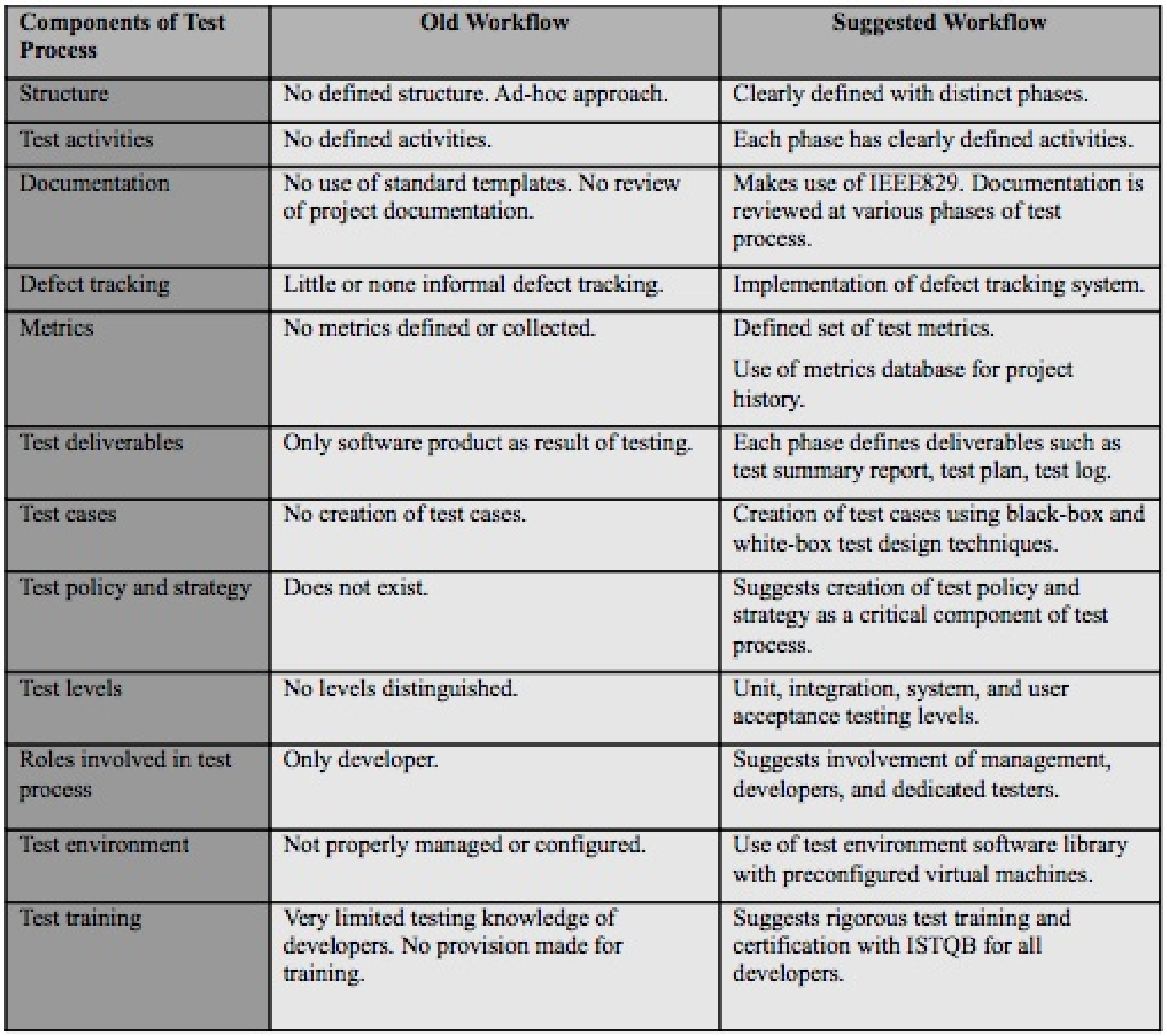


TMMi Level 3 Process Areas	Specific Goals	Company X Improved Workflow
3.1 Test Organisation	SG1: Establish a Test Organisation SG2: Establish Test Functions for the Test Specialists SG3: Establish Test Career Paths SG4: Determine, Plan and Implement Test Process Improvements SG5: Deploy Organisational Test Processes and Incorporate Lessons Learned	-- -- -- -- --
3.2 Test Training Program	SG1: Establish an Organisational Test Training Capability SG2: Provide Necessary Test Training	 -- --
3.3 Test Life Cycle and Integration	SG1: Establish Organisational Test Process Assets SG2: Integrate the Test Life Cycle with the Development Models SG3: Establish a Master Test Plan	-- X X
3.4 Non-functional Testing	SG1: Perform a Non-Functional Product Risk Assessment SG2: Establish a Non-Functional Test Approach SG3: Perform Non-Functional Test Analysis and Design SG4: Perform Non-Functional Test Implementation SG5: Perform Non-Functional Test Execution	-- -- -- -- -- -- --
3.5 Peer Reviews	SG1: Establish a Peer Review Approach SG2: Perform Peer Reviews	X X

Figure 5: Assessment of Improved Workflow against TMMi Level 3.

workgroup so that this project is separate from day-to-day work activities. Proper communication during this project is important as many changes will be introduced. All employees of company X need to know which changes are going to be implemented and why these changes are necessary. It was also recommended in [48] to issue a periodic newsletter that communicates progress of the project. This will assist with the buy-in from employees into the project. The use of an external consultant to take ownership of the project is also recommended by [48] to guide the entire process. We do not currently foresee this recommendation as feasible for very small companies, such as company X, due to the cost implications. However, the creation of an internal project team to manage this process improvement project can be critical to its success. Furthermore [48] recommended that the processes which the test process depend on must also be taken into consideration. As the suggested test process is tightly integrated into the overall software development process, this process must also be changed to accommodate the changes in the testing process. We have seen throughout our case study that the software development process will be affected by the suggested testing process. Therefore we suggested in [56] that a separate improvement project should be launched for the software development process improvement at large. Such a project could be managed by the current software development manager such that no additional resources would be needed in this case. We also recommended that the newly suggested test process be documented thoroughly and be made available to all stakeholders of the company.

The implementation phase of a process improvement project realises the recommendations made in the previous phases. We agree with [48] that the test policy and strategy are the first and most important changes to be im-



Components of Test Process	Old Workflow	Suggested Workflow
Structure	No defined structure. Ad-hoc approach.	Clearly defined with distinct phases.
Test activities	No defined activities.	Each phase has clearly defined activities.
Documentation	No use of standard templates. No review of project documentation.	Makes use of IEEE829. Documentation is reviewed at various phases of test process.
Defect tracking	Little or none informal defect tracking.	Implementation of defect tracking system.
Metrics	No metrics defined or collected.	Defined set of test metrics. Use of metrics database for project history.
Test deliverables	Only software product as result of testing.	Each phase defines deliverables such as test summary report, test plan, test log.
Test cases	No creation of test cases.	Creation of test cases using black-box and white-box test design techniques.
Test policy and strategy	Does not exist.	Suggests creation of test policy and strategy as a critical component of test process.
Test levels	No levels distinguished.	Unit, integration, system, and user acceptance testing levels.
Roles involved in test process	Only developer.	Suggests involvement of management, developers, and dedicated testers.
Test environment	Not properly managed or configured.	Use of test environment software library with preconfigured virtual machines.
Test training	Very limited testing knowledge of developers. No provision made for training.	Suggests rigorous test training and certification with ISTQB for all developers.

Figure 6: To-do list from old workflow to new workflow.

plemented. This will provide direction on the whole improvement project. We then recommend that each process area of TMMi level 2 (and, later, 3) that was satisfied with the suggested test process be implemented. We also recommended in [56] that the structure of the recommended documentation be created. All the templates from the IEEE standard 829 should be adapted to the needs of company X and stored in a document management system. A document management system is already in use in company X; therefore this practice should not present any great difficulties.

According to [48] the deployment phase of the improvement project is the most difficult one to do. All the documentation and process guidelines are futile if employees do not adhere to them. We therefore recommended in [56] that company X deploy the suggested test process on an internal software pilot project first. This will allow management to carefully monitor the progress of the test project without running the risk of possible failure with a customer-facing delivery project. Only when the basic practices of the test process have been implemented in small steps and the test process is producing the desired outcomes should it be implemented in customer-facing delivery projects. We estimate that this pilot implementation could take between six and eight months for a company similar to company X.

8 FUTURE WORK

Though our case study conducted with company X can be regarded as comprehensive, some aspects were not taken into account; they remain open for future work. For example: we did not investigate the test process from the perspective of the management of company X, because our study started from the interviews with the software developers (though permission was given by the management, which means that the management was aware of the fact that such a study was being carried out). However, an informal 'chat' with the Chief Executive Officer (CEO) of company X regarding this case study was encouraging: The CEO seemed convinced that those process improvements, as they were elaborated in [56] and summarised in the previous sections of this article, should be introduced in company X as soon as possible. This generally positive attitude by the CEO might be regarded as an indicator for the awareness of the current problems being experienced with the software process, more specifically the test process, at a too low level of TMMi.

Another perspective that was not taken into account in our case study was the perspective of the customers who were buying the (often defective) software products from company X. We did not analyse the customer's view of the products they purchased and their perceived quality thereof.

This case study only focused on one small company that might perhaps not be fully representative of the test processes of other small IT houses. Though there is much 'anecdotal evidence' of the similarity of many other small IT houses with our case study company X, we can —strictly speaking— not conclude from this one case study that software testing is a general problem facing many other small IT houses.

It is a task for future work to actually implement the suggested testing process in company X, and then to study the effects of the implementation, such that we would be able to judge if (and, if yes, to what extent) the process improvement exercise was successful. This would especially entail the consideration of the following parameters (and their possible correlations): software developers' disgruntlement, software defect rate, and customers' satisfaction, as briefly outlined in the following:

- Software developers in company X were, at the time of our case study, disgruntled with the current process because there was often considerable re-work to be carried out after delivery of defective software. The customers would obviously send complaints about such products, such that the re-maintenance of those defects disturbed the software developers' schedules while they were already working on new projects, which meant that the new projects got into delay by the need to fix the defects from previous projects. It is well known that employee disgruntlement can have negative effects on the company as a whole.
- Software defect rates at company X were not counted during the time of our case study. Without such figures, however, it is not possible to measure the effects of any suggested test improvement exercise. Defect measuring is thus the precondition for any subsequent initiative. The introduction of defect rate counting to company X would therefore be the first step of future work into the direction of a long-term study.
- Future work would need to determine how satisfied customers are with the quality of the current software. Once the suggested process has been implemented, one would have to ask the customers again in order to find out if they are happier now with the less defective software than previously. This study would, however, require also some recurring long-term customers; a study like this would be difficult to conduct on the basis of one-time customers who only buy one product and then never come back.

However, the ultimate goal for future work would be the achievement of getting company X TMMi-accredited for level 2; this could be regarded as sufficient evidence for the effectivity of the improvement measurements described in [56] and in the previous sections of this article. Since official TMMi accreditations are still quite rare —especially here in South Africa— a small IT house which is able to present its TMMI certification diploma to prospective customers should be able to gain a considerable competitive advantage over its countless competitors on the crowded software market.

Acknowledgments

We would like to thank the software developers in company *X* for having participated in the survey behind this contribution. Thanks to the participants of the South African Institute for Computer Scientists and Information Technologists (SAICSIT) Postgraduate Student Workshop 2009 for their comments on a related presentation. Last but not least many thanks to *Bernhard Westfechtel* for his expertise.

REFERENCES

[1] P. Amman, J. Offutt, "Introduction to Software Testing". Cambridge Univ. Press, 2008.

[2] J. Aranda, S. Easterbrook, G. Wilson, "Requirements in the Wild: How Small Companies do it". Proc. RE'07: 15th IEEE International Requirements Eng. Conf., pp. 39-48, 2007.

[3] J. Batista, A.D. de Figueiredo, "SPI in a Yery Small Team: A Case with CMM". Software Process Improvement and Practice 5/4, pp. 243-250, 2000.

[4] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams". Proc. FOSE '07: Future of Software Engineering Conf., pp. 85-103, 2007.

[5] R. Black, "Critical Testing Processes: Plan, Prepare, Perform, Perfect". Addison-Wesley, 2003.

[6] R. Black, "ISTQB Certification: Why You Need It and How to Get It". Testing Experience The Magazine for Professional Testers 1, pp. 26-30, 2008.

[7] K. Blokland, "A Universal Management and Monitoring Process for Testing". Proc. ICSTW'08: IEEE Internat. Conf. on Softw. Testing, Verification and Validation, pp. 315-321, 2008.

[8] F. Brooks, "The Mythical Man-Month". Addison-Wesley, 1975.

[9] M. Broy, D. Rombach, "Software Engineering: Wurzeln, Stand und Perspektiven". Informatik Spektrum 16, pp. 438-451, 2002.

[10] I. Burnstein, T. Suwanassart, R. Carlson, "Developing a Testing Maturity Model for Software Test Process Evaluation and Improvement". Proc. Internat. Test Conf., pp. 581-589, 1996.

[11] A. Cater-Steel, "Process Improvement in four Small Software Companies". Proc. ASWEC 13th Australian Softw. Eng. Conf., pp. 262-272, 2001.

[12] G. Coleman, F. McCaffery, P.S. Taylor, "Adept: A Unified Assessment Method for Small Software Companies". IEEE Software 24/1, pp. 24-31, 2007.

[13] S.H. Edwards, "Improving Student Performance by Evaluating how well Students Test their own Programs". Journal Educ. Resour. Comput. 3/3, pp. 1-, 2003.

[14] A. Farooq, P.R. Dumke, "Evaluation Approaches in Software Testing". Techn. Rep. FIN-05-2008, Faculty of Comp. Sc., Univ. of Magdeburg, 2008. http://www.cs.uni-magdeburg.de/fin_media/downloads/forschung/preprints/2008/TechReport5.pdf

[15] M.E. Fayad, M. Laitinen, R.P. Ward, "Thinking objectively: Software Engineering in the Small". Communications of the ACM 43/3, pp. 115-118, 2000.

[16] J. Garcia, A. de Amescua, M. Velasco, A. Sanz A, "Ten Factors that impede Improvement of Verification and Validation Processes in Software intensive Organizations". Software Process Improvement and Practice 13/4, pp. 335-343, 2008.

[17] D. Gelperin, B. Hetzel, "The Growth of Software Testing". Communications of the ACM 31/6, pp. 687-695, 1988.

[18] B. George, L. Williams, "An initial Investigation of Test Driven Development in Industry". Proc. SAC'03: Annual ACM Symp. on Appl. Comp., pp. 1135-1139, 2003.

[19] A.M. Geras, M.R. Smith, J. Miller, "A Survey of Software Testing Practices in Alberta". Canadian Journal of Electrical and Comp. Eng. 29/3, pp. 183-191, 2004.

[20] R.L. Glass, R. Collard, A. Bertolino, J. Bach, C. Kaner, "Software Testing and Industry Needs". IEEE Software 23/4, pp. 55-57, 2006.

[21] M. Grindal, J. Offutt, J. Mellin, "On the Testing Maturity of Software Producing Organizations". Proc. TAIC PART: Testing: Academic and Industrial Conference on Practice and Research Techniques, pp.171-180, 2006.

[22] P. Grunbacher, "A Software Assessment Process for Small Software Enterprises". Proc. EUROMICRO'97, 23rd EUROMICRO Conf., pp. 123-128, 1997.

[23] N. Juristo, A.M. Moreno, W. Strigel (eds.), "Software Testing Practices in Industry". IEEE Software 23/1, pp. 19-21, 2006.

[24] M. Kajko-Mattsson, T. Björnsson, "Outlining Developers' Testing Process Model". Proc. EUROMICRO'07, 33rd EUROMICRO Conf., pp. 263-270, 2007.

[25] D. Karlström, P. Runeson, S. Norden S, "A Minimal Test Practice Framework for Emerging Software Organizations". Software Testing Verification and Reliability 15/3, pp. 145-66, 2005.

[26] K. Kautz, H.W. Hansen, K. Thaysen, "Applying and Adjusting a Software Process Improvement Model in Practice: The Use of the IDEAL Model in a Small Software Enterprise". Proc. ICSE'2000, 22nd Internat. Conf. on Softw. Eng., pp. 626-633, 2000.

[27] T. Koomen, M. Pol, "Test Process Improvement: A practical step-by-step guide to structured testing". Addison-Wesley, 1999.

[28] L. Lazic, N. Mastorakis, "Cost effective Software Test Metrics". WSEAS Transactions on Comp. 7/6, pp. 599-619, 2008.

[29] W.E. Lewis, "Software Testing and Continuous Quality Improvement". 2nd ed., Auerbach Publ., 2004.

[30] D. Maes, S. Mertens, "10 Tips for Successful Testing". Testing Experience: The Magazine for Professional Testers 3, pp. 52-53, 2008.

[31] W. Mallinson, "Testing: A Growing Opportunity", 2002. http://www.testfocus.co.za/featurearticles/Jan2002.htm

[32] K. Martin, B. Hoffman, "An Open Source Approach to Developing Software in a Small Organization". IEEE Software 24/1, pp. 46-53, 2007.

[33] G.E. Mogyorodi, "Let's Play Twenty Questions: Tell me about your Organization's Quality Assurance and Testing". Crosstalk: The Journal of Defense Software Engineering, without page numbers, 2003. http://www.stsc.hill.af.mil/crosstalk/2003/03/mogyorodi1.html

[34] S.P. Ng, T. Murnane, K. Reed, D. Grant, T.Y. Chen, "A Preliminary Survey on Software Testing Practices in Australia". Proc. Australian Softw. Eng. Conf., pp. 116-125, 2004.

[35] M. Northover, D.G. Kourie, A. Boake, S. Gruner, A. Northover, "Towards a Philosophy of Software Development: 40 Years after the Birth of Software Engineering". Zeitschrift für allgemeine Wissenschaftstheorie 39/1, pp. 85-113, 2008.

[36] S. Otoya, N. Cerpa, "An Experience: a Small Software Company Attempting to Improve its Process". Proc. STEP'99 Software Technology and Engineering Practice Conf., pp. 153-160, 1999.

[37] J. Park, H. Ryu, H.J. Choi, D.K. Ryu, "A Survey on Software Test Maturity in Korean Defense Industry". Proc. ISEC'08: 1st India Softw. Eng. Conf., pp. 149-150, 2008.

[38] I. Pinkster-o'Riordain, "Test Policy: Gaining Control on IT Quality and Processes". ICSTW'08: Proc. IEEE Internat. Conf. on Softw. Testing Verification and Validation, pp. 338-342, 2008.

[39] F.J. Pino, Felx Garcia, M. Piattini, "Key Processes to start Software Process Improvement in Small Companies". Proc. SAC'09 ACM Symposium on Applied Comp., pp. 509-516, 2009.

[40] M. Pyhäjärvi, K. Rautiainen, J. Itkonen J, "Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects". Proc. 36th IEEE Internat. Conf. on System Sc., pp. 250-259, 2002.

[41] P. Savolainen, H. Sihvonen, J.J. Ahonen, "SPI with Lightweight Software Process Modeling in a Small Software Company". LNCS 4764, pp. 71-81; 2007.

[42] T. Shepard, M. Lamb, D. Kelly D, "More Testing should be taught". Communications of the ACM 44/6, pp. 103-108, 2001.

[43] M. Siniaalto, P. Abrahamsson, "Does Test-Driven Development Improve the Program Code? Alarming Results from a Comparative Case Study". LNCS 5082, pp. 143-156, 2008.

[44] A. Telea, H. Byelas, "Querying Large C and C++ Code Bases: The Open Approach". Colloquium and Festschrift at the Occasion of the 60th birthday of Derrick Kourie, 2008. http://www.cs.up.ac.za/cs/sgruner/Festschrift/

[45] E. van Veenendaal, M. Pol, "A Test Management Approach for Structured Testing". Achieving Software Product Quality, online collection without page numbers, 1997.

[46] E. van Veenendaal, "Test Maturity Model integrated (TMMi): Version 2.0". TMMi Foundation, http://www.tmmifoundation.org/

[47] E. van Veenedaal, D. Graham, I. Evans, R. Black, "Foundations of Software Testing: ISTQB Certification". Internat. Thomson Business Press, 2008.

[48] E. van Veenendaal, R. Hendriks, J. van de Laar, B. Bouwers, "Test Process Improvement using TMMi". Testing Experience: The Magazine for Professional Testers 3, pp. 21-25, 2008.

[49] T.E.J. Vos, J.S. Sanches, M. Mannise, "Size does matter in Process Improvement". Testing Experience The Magazine for Professional Testers 3, pp. 91-95, 2008.

[50] C.G. von Wangenheim, T. Varkoi, C.F. Salviano, "Standard-based Software Process Assessments in Small Companies". Software Process Improvement and Practice 11/3, pp. 329-35, 2006.

[51] C.G. von Wangenheim, A. Anacleto, C.F. Salviano, "Helping Small Companies Assess Software Processes". IEEE Software 23/1, pp. 91-98, 2006.

[52] C.G. von Wangenheim, S. Weber, J.C.R. Hauck, G. Trentin G, "Experiences on Establishing Software Processes in Small Companies". Information and Software Technology 48/9, pp. 890-900, 2007.

[53] C.G. von Wangenheim, I. Richardson, "Why are Small Software Organizations Different?" IEEE Software 24/1, pp. 18-22, 2007.

[54] A.H. Watson, T.J. McCable TJ, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric". Technical Report: National Institute of Standards and Technology, NIST 500/235, 1996. http://www.mccabe.com/pdf/nist235r.pdf

[55] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J.P, Hudepohl, A.M. Vouk, "On the Value of Static Analysis for Fault Detection in Software". IEEE Transactions on Softw. Eng. 32/4, pp. 240-253, 2006.

[56] Johan van Zyl, "Software Testing in a Small Company: A Case Study". M.IT.-Thesis, Department of Comp. Sc., Univ. of Pretoria, December 2009.

[57] Kautz K, Hansen HW, Thaysen K, editors. "Applying and adjusting a software process improvement model in practice: the use of the IDEAL model in a small software enterprise" ICSE '00: Proceedings of the 22nd international conference on Software engineering; 2000; New York, NY, USA: ACM