Software Qual J (2016) 24:519–548
DOI 10.1007/s11219-015-9289-z

Risk orientation in software testing processes of small and medium enterprises: an exploratory and comparative study

Michael Felderer[1] · **Rudolf Ramler**[2]

Published online: 14 August 2015

Abstract Risk orientation in testing is an important means to balance quality, time-to-market, and cost of software. Especially for small and medium enterprises (SME) under high competitive and economic pressure, risk orientation can help to focus testing activities on critical areas of a software product. Although several risk-based approaches to testing are available, the topic has so far not been investigated in the context of SME, where risks are often associated with business critical issues. This article fills the gap and explores the state of risk orientation in the testing processes of SME. Furthermore, it compares the state of risk-based testing in SME to the situation in large enterprises. The article is based on a multiple case study conducted with five SME. A previous study on risk-based testing in large enterprises is used as reference for investigating the differences between risk orientation in SME and large enterprises. The findings of our study show that a strong business focus, the use of informal risk concepts, as well as the application of risk knowledge to reduce testing cost and time are key differences of risk-based testing in SME compared to large enterprises.

Keywords Test process improvement · Test management · Software risk management · Software testing · Risk-based testing · System testing · Software quality · Multiple case study · Small and medium enterprises · SME

✉ Michael Felderer
michael.felderer@uibk.ac.at

Rudolf Ramler
rudolf.ramler@scch.at

[1] Institute of Computer Science, University of Innsbruck, Innsbruck, Austria

[2] Software Competence Center Hagenberg, Hagenberg im Mühlkreis, Austria

1 Introduction

In most countries, SME represent up to 85 percent of all software organizations (Richardson and von Wangenheim 2007). However, to persist and grow, these small organizations need efficient and effective software engineering solutions. But the proper implementation of software engineering techniques is a difficult task for SME as they often operate on limited resources and with strict time constraints (Mishra and Mishra 2008). In particular, software testing is a strenuous and expensive process consuming up to 50 % of the total development costs (Harrold 2000; Bath and van Veenendaal 2014), thus providing a perfect starting point for improving software process and product quality in small and medium enterprises (SME). In particular for SME, software testing plays a key role in balancing between quality, time-to-market, and cost of software-based products and services to guarantee the company's success. Therefore, testing requires the adequate consideration of risks to focus testing activities on those areas that trigger the most critical situations for a software product and further for the overall business.

Lately, the international standard ISO/IEC/IEEE 29119 Software Testing (2013) on testing techniques, processes, and documentation even explicitly mentions risks as an integral part of the testing process. Also several risk-based testing approaches which consider risks of the software product as the guiding factor to support decisions in all phases of the test process have been proposed in the literature (Erdogan et al. 2014; Felderer and Schieferdecker 2014). But so far, risk-based testing has not been investigated in context of SME, where risk orientation in the software test process can have a business critical impact.

This article fills this gap as its research objective is to explore the state of risk orientation in testing processes of SME and to compare it to the state of risk-based testing in large enterprises. For this purpose, the research questions (1) how is software testing organized in SME, (2) what is the role of risk in software testing, and (3) what are the differences of risk-based testing in SME and large enterprises are empirically investigated by a multiple case study (Yin 2014). The case study methodology is well suited for this kind of research objective, as the object of study is a contemporary phenomenon, i.e., risk orientation in testing processes of SME, which can only be studied in its context and not in isolation (Runeson et al. 2012). Five SME have been selected as cases for this multiple case study. Case A develops a commercial off-the-shelf software product for enterprise resource planning, Case B develops and manufactures biometric access systems, Case C develops a commercial off-the-shelf software product for document management, Case D provides recruitment and training services as well as a software platform to support these, and finally Case E is a solution provider offering billing and cashless payment systems for restaurants and canteens. For comparing the findings of risk orientation in SME to large enterprises, results from a previous multiple case study on state of the art of risk-based testing in large enterprises (Felderer and Ramler 2014b) were taken as reference. The key results of the previous multiple case study, which we refer to in this study, are presented in Sect. 2.3. The results of our study show that a strong business focus, the informality of the applied risk concept, and the usage of risks to reduce testing cost and time are key differences for risk-based testing in SME versus large companies.

The remainder of this article is structured as follows. Section 2 provides background and related work on risk-based testing, software process improvement, and testing in SME, as well as results from previous studies. Section 3 presents the research design including the research questions, the case selection as well as the data collection, analysis, and

validity procedures. Section 4 describes and analyzes the studied cases. Section 5 presents the results and answers to the research questions based on the findings derived from the studied cases. Section 6 discusses threats to validity. Finally, Sect. 7 summarizes and concludes our work.

2 Background and related work

This article investigates risk-based testing in SME. Hence, in this section, we discuss the background and related work on risk-based testing as well as on related software process improvement in SME. Furthermore, we provide an overview of results from a previous study on risk-based testing in large enterprises (Felderer and Ramler 2014b).

2.1 Risk-based testing

Risk-based testing (RBT) is a testing approach which considers risks of the software product as the guiding factor to support decisions in all phases of the test process (Gerrard and Thompson 2002; Felderer and Schieferdecker 2014). A *risk* is a factor that could result in future negative consequences and is usually expressed by its likelihood and impact (ISTQB 2012). In software testing, the *likelihood* is typically determined by the probability that a failure assigned to a risk occurs, and the *impact* is determined by the cost or severity of a failure if it occurs in operation. The resulting *risk value* or *risk exposure* is assigned to a *risk item*. In the context of testing, a risk item is anything of value (i.e., an asset) under test, for instance a requirement, a component, or a fault. Based on the risk exposure values, the risk items are typically prioritized and assigned to *risk levels* defining a *risk profile*. The

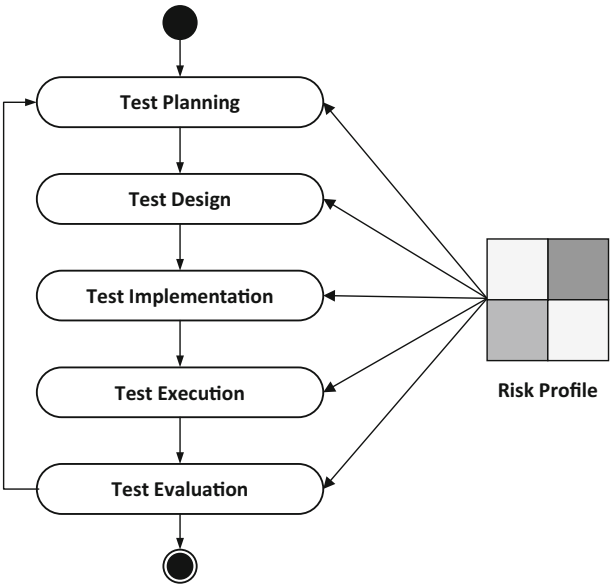

Fig. 1 Risk-based testing: risk profile supports decisions in all phases of the test process

information collected in the risk profile is used to support decisions in all phases of the test process (see Fig. 1).

A *test process* contains the core activities *test planning*, *test design*, *test implementation*, *test execution*, as well as *test evaluation* (ISTQB 2012). Test planning is the activity of establishing or updating a test plan. A test plan is a document describing the scope, approach, resources, and schedule of intended test activities (ISTQB 2012). During the test design phase, the general testing objectives defined in the test plan are transformed into tangible test conditions and test cases. Tests are then implemented which contains remaining tasks like preparing test harnesses and test data, or writing automated test scripts which are necessary to enable the execution of the implementation-level test cases. The tests are then executed, and all relevant details of the execution are recorded in a test log. During the test evaluation and reporting phase, the exit criteria are evaluated and the logged test results are summarized in a test report. Development projects typically contain several test cycles, and therefore, all or some phases of the test process are performed iteratively.

Recently, risk-based testing attracted considerable interest in research (Felderer and Schieferdecker 2014; Felderer et al. 2014), and several risk-based testing approaches have been published as shown in a current systematic literature review on risk-based testing (Erdogan et al. 2014). In this review, 22 risk-based test approaches were identified, among these some risk-based testing approaches addressing the integration of risk analysis into test processes on a general level (Amland 2000; Felderer et al. 2012; Redmill 2004).

Amland (2000) defines a risk-based testing approach that is based on the generic risk management process of Karolak (1995) comprising the steps planning, identification of risk indicators, identification of the cost of a failure, identification of critical elements, test execution, and estimation and completion.

Felderer et al. (2012) take the standard test process of the ISTQB (2012) as a starting point, integrate risk identification and risk analysis into it, and extend the activities of the standard test process by risk-specific elements.

Redmill (2004, 2005) addresses the loose integration of risk analysis and testing to make testing more effective. The author proposes a single-factor risk analysis, either based on probability or impact, or a two-factor risk analysis, in which probability and impact are combined.

Although several risk-based testing approaches have been proposed (Erdogan et al. 2014), only a few empirical studies on risk-based testing are available (Yoon and Choi 2011; Souza et al. 2010; Felderer and Ramler 2014a, b).

Yoon and Choi (2011) propose a test case prioritization strategy for risk-based testing and evaluate its effectiveness on the basis of data from a traffic conflict avoidance system.

Souza et al. (2010) indicate in a small case study that risk-based testing focuses on the parts of a software that are more likely to fail. The risk-based testing approach is based on their risk-based test process RBTProcess (Souza et al. 2009) which consists of the phases risk identification, risk analysis, test planning, test design, test execution, as well as test evaluation and risk control.

Felderer and Ramler (2013, 2014a) provide an empirical case study on introducing risk-based testing into established test processes. In a retrospective analysis, the authors investigate the applicability and the potential benefits from introducing risk-based testing in an industrial project.

Finally, Felderer and Ramler (2014b) provide a multiple case study on risk-based testing in industry. In their article, the authors analyze how risk is defined, assessed, and applied to support and improve testing activities in projects, products, and processes on the

basis of three industrial cases taken from large enterprises. As this study forms the basis for this article, its results are discussed in more detail in Sect. 2.3.

Risk-based testing is also an important topic from the point of view of test process improvement. Felderer and Ramler (2014a) present an approach for the stepwise introduction of risk-based testing into an established test process and discuss benefits as well as prerequisites of this integration in the context of an industrial project. General maturity models for test processes like Test SPICE (Steiner et al. 2012), Test Process Improvement (TPI Next) (Koomen and Pol 1999; Koomen et al. 2006), or Test Maturity Model integration (TMMi) (van Veenendaal et al. 2008) do not explicitly address risk-based testing, but can also be applied to risk-based testing. However, risk-based testing aspects specific to SME have not been addressed so far.

2.2 Software process improvement and testing in SME

Although SME represent up to 85 percent of all software organizations and require efficient and effective software engineering solutions (Richardson and von Wangenheim 2007), relatively few publications present software engineering solutions or studies focusing specifically on SME.

Most software engineering work specific to SME has been performed in the area of software process improvement. Pino et al. (2008) present a systematic literature review of published case studies on software process improvement effort in SME. Overall 45 primary studies were obtained. Results show that formal process improvement models like CMMI (Ahern et al. 2008) or SPICE (Dorling 1993) can be applied only with difficulty in small companies and are adapted by SME when undertaking their improvement efforts. Even for the standard ISO/IEC 29110 (2011), which particularly aims at assisting very small entities having up to 25 people to benefit from standards designed to address their needs, barriers to adoption were recently reported (Sanchez-Gordon et al. 2015). Furthermore, introduced improvements are measured through informal and non-objective processes, based on the employees' perception and not through formal measurement processes. Therefore, measurement programs (Kautz 1999) and specific assessment methods (Mc Caffery et al. 2007; Pino et al. 2010) for software process improvement in SME were defined. These methods enable a focused and tailored improvement path based on a company's operational context and business goals. Furthermore, García et al. (2012) even provide a Web-based tool to manage software process improvement in small enterprises. Finally, Mishra and Mishra (2008) provide criteria to compare software process improvement methodologies for SME and compare the significant characteristics of five methodologies. The authors conclude that each one has benefits and limitations. Thus, an SME should select a suitable process improvement methodology taking its business goals, models, characteristics, and resource limitations into account.

Risk and testing aspects specific to SME are addressed only in a few publications (Karlström et al. 2005; Azar et al. 2007; Martin and Hoffman 2007; Gleirscher et al. 2014). Karlström et al. (2005) present a minimal test practice framework (MTPF) that allows the incremental introduction of appropriate practices at the appropriate time in small and emerging software companies. MTPF defines the kind of practices that are needed in these companies. It is structured in five categories corresponding to areas in testing and test organization and is levelled in three phases corresponding to the growth of the company. The five categories are problem and experience reporting, roles and organization issues, verification and validation, test administration, as well as test planning. The first phase includes practices that are suitable for a company with approximately 10 employees in

development, the second for approximately 20, and the third for 30 and more. MTPF takes the practice of risk management into account in phase three (i.e., for 30 and more employees in development) in the category test administration. Risk management in MTPF aims to identify problem areas at the beginning of the project and avoid or reduce the impact of issues within these areas. Azar et al. (2007) present an approach to value-oriented requirements prioritization for small organizations, called value-oriented prioritization (VOP). In VOP, company executives identify core business values and use a simple ordinal scale to weight them according to their importance to the organization. By analogy, the framework also supports the identification and weighting of risk categories which represent the organization's tolerance for engaging in those risks and are measured on a negative scale. Using the core business values and risks, VOP constructs a prioritization matrix to score and rank requirements. The approach is clear and especially intended for small organizations which recognize the potential benefit of requirements prioritization but find it difficult to allocate already thin resources to it. Martin and Hoffman (2007) present an open source approach for developing software in a small organization. The approach must have low overhead and be flexible as the considered company implements small and large projects. The approach puts emphasis on communication and documentation as well as on automation. The organization provides a simple framework for adding tests to projects and an easy mechanism for running those tests and for viewing their results. For small or short-lived projects, testing is often limited to build and smoke tests. For larger projects, more effort is put into testing and creating specific regression tests to exercise key capabilities. In general, testing is performed on a daily basis to reduce the cost of fixing defects later. Finally, Gleirscher et al. (2014) report on experiences from introducing automated static analysis in small- and medium-sized enterprises as a measure to mitigate the risk of additional, expensive effort for bug fixes or compensations. Their study of five software projects from five SME shows that the effort required to introduce automated static analysis techniques in SME is small and helps to detect multiple defects in production code, which was perceived to be beneficial by the participating companies. With the help of a quality model, the static analysis results could efficiently be aggregated. The study observed a mismatch between the ratings provided by the quality model and the opinion of the study participants concerning many quality characteristics.

The combination of risk and software testing aspects, i.e., risk-based testing, specific for SME has so far not explicitly been addressed in the literature. Only Karlström et al. (2005) highlight the important role of risk management for test administration in enterprises of moderate size, i.e., 30 and more employees in development. This article provides the first study in the direction of investigating efficient and effective testing for SME taking risk aspects into account.

2.3 Previous results

In a previous paper (Felderer and Ramler 2014b), the authors of this article analyze how risk is defined, assessed, and applied to support and improve testing activities in large enterprises. The paper addresses four research questions, i.e., (RQ1) the notion of risk used in software testing, (RQ2) how risks support the activities of the software test process, (RQ3) how risks are organized from a technical perspective, as well as (RQ4) the benefits of risk-based testing. The main findings gained by cross-case analysis are summarized in Table 1.

The paper investigates these issues empirically by a multiple case study based on three cases from different backgrounds, i.e., a test project in context of the extension of a large

Table 1 Findings on how risk is defined, assessed, and applied to support and improve testing activities in large enterprises (Felderer and Ramler 2014b)

Research question	Finding
RQ1: *What is the notion of risk in software testing?*	Testing is a measure to reduce risk exposure, the focus is on product risks, the risk concept is dependent on context and scope
	Common definition of risk $R = P \times I$ applies, definition and relation of P and I may remain informal and implicit, usually based on (subjective) estimates
	Degree of formality of risk depends on the application scope, formality increases with wider scope and abstraction level
RQ2: *How do risks support the activities of the software test process?*	Risk is considered in all testing activities, even when not following an explicit risk-based testing approach
	Risk information from testing is not explicitly considered in making release decisions
	Risk information is used to extend testing toward "risky" areas, risk information is used as a guideline to focus testing on "risky" areas
	Risk-based testing is not used to reduce the amount of testing, the overall test effort, or the established test budgets
RQ3: *How are risks organized from a technical perspective?*	Risks at the level of project management are individual entities, risks in testing are calculated from risk information associated with testable entities
	Risk-based testing relies on standard test tool infrastructure, risk management tools are not applied in software testing
	Implementation of risk-based testing uses an extensions to standard test tools
	Tool support for risk-based testing is an aid for collecting and analyzing measurement data from which to derive risk information
RQ4: *What are the benefits of a risk-based testing approach?*	Make testing more effective: detecting additional defects in testing, fewer defects slip through to the field
	Make testing more effective: prioritization for detecting most critical defects first, reduces overall stabilization costs and time
	Risk information used for informed decision-making and new insights to triangulate and refine decisions

Web-based information system, product testing of a measurement and diagnostic equipment for the electrical power industry, as well as a test process of a system integrator of telecommunications solutions.

The first research question investigated the notion of risk in software testing. We found that the concept of risk depends on context and scope, but that there is an agreement that product risks are relevant for testing, which is commonly understood as measure to reduce the risk exposure of the product. It turned out that all cases follow the common definition of risk relating its probability and impact. However, the definition may remain implicit and the degree of formality depends on the application scope, i.e., it increases from project to product, and, furthermore, to process.

The second research question investigated how risks support the activities of the software test process. We found that risk is taken into consideration in virtually all testing activities, e.g., to define test end criteria in the test planning phase, to design test cases, and to select and prioritize test cases for test execution. But risk information used in testing has not been found to play an explicit role in making release decisions. Our investigation also turned out that risk-based testing is not understood as an approach to reduce the amount of testing, but on the contrary, additional test cases are usually designed as a consequence of addressing all risks. Risk-based testing is motivated by the adequate mitigation of risks, more than by the reduction in testing efforts and resources.

The third research question investigated how risks organized from a technical perspective. We found that in testing, risks are calculated from risk information associated with testable entities for identifying critical areas of the software systems, where the testing effort is concentrated on. Risk-based testing relies on the established standard tool infrastructure for testing. These tools usually include conventional test management tools and custom extension, but not specific risk management tools. Risk-based testing is implemented with the help of the available tools or as extension to the existing tool infrastructure.

The fourth research question investigated the benefits of risk-based testing. As central benefits of risk-based testing, we identified using risk information to increase the range of testing for detecting additional defects and to target the most critical defects from the very beginning. In addition, test managers and testers consider the incorporation of risk as beneficial for informed decision-making on resource allocation, defining regression test suites, identifying candidates for test automation, as well as when to end testing.

3 Research design

In this section, we present the research questions (Sect. 3.1) addressed in this article, the case selection (Sect. 3.2) as well as the data collection, analysis, and validity procedures to answer them (Sects. 3.3–3.5). The research design follows the guidelines for conducting and reporting case study research proposed by Runeson et al. (2012).

3.1 Research questions

The overall goal of this article is to explore the state of risk orientation in testing processes of SME and to compare it to the state in large enterprises. For this purpose, the following research questions are investigated.

(**RQ1**) How is software testing organized in SME? This research question investigates what roles are involved in testing, what activities they perform, and what artifacts they use as input or produce as output.

(**RQ2**) What is the role of risk in software testing of SME? This research question is addressed by the following four sub-questions.

(**RQ2a**) What is the notion of risk in software testing? This research question addresses the commonalities and differences of the risk definitions used in practice.

(**RQ2b**) How do risks support the activities of the software test process? This research question addresses the various ways in which risk information is used for supporting software testing.

(**RQ2c**) How are risks organized from a technical perspective? This research question investigates the implementation aspects involved in risk-based testing, addressing the modeling and the tool support related to risk.

(**RQ2d**) What are the benefits of a risk-based testing approach? This research question summarizes the positive effects that are actually realized or expected to be observed in future when considering risk for testing purposes.

(**RQ3**) What are the differences of risk-based testing in SME versus large enterprises? This research question compares the findings on risk-based testing in SME provided as answer to RQ2 to findings on risk-based testing in large enterprises achieved in a previous study (Felderer and Ramler 2014b).

These research questions served as guidelines for interviews and document analysis when studying the five cases presented in this article.

3.2 Case selection

Five SME have been selected as cases for investigation. They were selected from the wider set of industrial collaboration projects of the authors and all participated in a joint research project on testing in SME. Together, the five cases represent a wide range of software development and testing in SME: Case A develops a commercial off-the-shelf software product for enterprise resource planning, Case B develops and manufactures biometric access systems, Case C develops a commercial off-the-shelf software product for document management, Case D provides recruitment and training services as well as a software platform to support these, and finally Case E is a solution provider offering billing and cashless payment systems for restaurants and canteens. Table 2 provides a tabular overview of the five cases according to the criteria domain, core business, core software (SW), overall number of employees and employees in software development, frequency of software releases, as well as the integration of software development in the organization. In the following subsection, each case is then discussed in detail.

3.3 Data collection procedure

The case study is based on data from the five selected cases described in detail in Sect. 4. For answering the research questions, the study relies on multiple sources of evidence. Product documentation, process descriptions, requirements specifications, design specifications, risk assessment documentations, test plans, test reports, release notes, and issue reports were analyzed. In addition, semi-structured interviews were conducted, based on a list of open interview questions derived from the research questions. Interviewed personnel of the SME were chief executive officers, project managers, developers, and testers. Furthermore, the five case companies were also involved in a joint research transfer project, which gave the authors the opportunity to observe and discuss the processes, methods, and tools they apply in their daily work and to gain a good understanding of the context in which the companies operate.

3.4 Analysis procedure

The analysis across the five cases was conducted using qualitative methods. The analysis was performed by two researchers and was based on the collected data including previous results, documents, tools, as well as interview and workshop documentation. The analysis

Table 2 Overview of cases

	Case A	Case B	Case C	Case D	Case E
Domain	ERP software	Access systems	Document management	Training and recruitment	Payment systems
Core business	SW product + Service	HW incl. SW	SW Product + custom development	Service + SW platform	Solution incl. SW + HW
Core SW	SW product	Embedded SW	SW product	SW configuration and operation	SW product
Employees (in SW)	15 (4)	40 (10)	10 (8)	40 (4)	15 (5)
SW releases	Main releases 2 to 4 time per year; service releases on demand	Adjusted to HW product cycles (yearly)	Between one and four weeks	On demand	Delivery as custom projects
SW in organization	Top level	Part of R&D	Top level	Part of IT department	Top level

was carried out following a reviewed protocol and keeping a clear chain of evidence (Yin 2014), i.e., allowing to trace the derivation of results and conclusions from the collected data. The analysis results are presented in Sect. 5. As a first step, the two researchers analyzed and compared the results of the open-ended interviews, where each company provided information regarding its organization, products, software development and test process, as well as its concept or risk. Together with an analysis of documents (i.e., product documentation, process descriptions, requirements specifications, design specifications, risk assessment documentations, test plans, test reports, release notes, and issue reports) and tools (i.e., for project management, test management, requirements management, defect, and issue tracking), the comparison provided the basis for the description of the studied cases, presented in Sect. 4, as well as for the findings on how software testing is organized, presented in Sect. 5.1. On this basis, workshops on risk estimation and risk-based testing activities were organized with all case companies. In these workshops, the actual state of risk estimation and testing in the companies was discussed and assessed together with representatives from each company. The results of the workshops were analyzed and documented by the two researchers. A taxonomy for risk-based testing (Felderer and Schieferdecker 2014) was used as framework and to guide the analysis in categorizing and comparing the risk-based testing approaches. The findings are presented in Sect. 5.2. These findings were mapped to the findings of a previous study on large enterprises (see Sect. 2.3). The mapping of findings is based on an assessment that resulted in classifying the relationship either as contradiction, agreement, or subset relationship. The classification was first performed independently by each researcher and then discussed to validate the classification and to reach consensus in case of deviations. The resulting mappings are presented in Sect. 5.3.

3.5 Validity procedure

Based on the guidelines of Runeson et al. (2012), validity threats were analyzed according to construct validity, reliability, internal validity, and external validity. Selected countermeasures against threats to validity were then taken. For example, data extracted from documents were triangulated by interviews and all analysis steps were conducted by two researchers. It was also seen as important that sufficient time was spent in the organizations to understand the cases. The threats to validity and countermeasures are discussed in Sect. 6.

4 Description of the studied cases

In this section, the five studied SME selected as cases for this study are described and analyzed. Due to confidentiality reasons, links to the involved companies were omitted and their names were replaced by alphabetical numbering.

4.1 Case A

Case A is small software company developing a commercial off-the-shelf software product for enterprise resource planning (ERP). The company has specialized on secondary contractors of the building and construction industry in Austria, a market niche where strong know-how about the country-specific standards and regulations is mandatory. In addition to

the software product, the company also offers server and client hardware, consulting services, and support to their customers.

The software product has been realized as client/server system with Java/J2EE technologies. The server hosts the database and provides common services such as reporting or printing to the clients. Different modules running on a rich client platform implement the business functionality of the software product, for example price calculation, accounting, and inventory control.

The organizational structure reflects the company's portfolio. Only four out of the 15 employees are software developers. They are led by the CTO, and together, they cover all roles and activities in software engineering, from requirements elicitation over architecture and design to software testing. However, due to the specialized domain knowledge necessary for developing the software product, the work of the individual developers is usually split according to the product's main modules.

Software development is driven by the ongoing maintenance and evolution of the software product. There is a high volume of short-term requests by customer support and management that involves development tasks. The ability and flexibility to implement these tasks on short notice—in urgent cases by the next day—is a competitive advantage for the company.

Over the last years, also the migration of the software system to the client/server platform took place in parallel to the ongoing maintenance and evolution of the software product. While the maintenance and evolution is a continuous process, the migration of the modules was organized in the form of individual projects. These project-driven development cycles follow an internal roadmap, although interrupts due to urgent product maintenance tasks are common.

Per year about two to three major versions of the software product are released, mostly timed to meet regulatory changes and to fulfill new standards. Besides the main releases, numerous minor versions are released to accommodate the changes made to satisfy customer requests.

The two tracks—software product evolution and software project development—do not follow strict processes or specific development models. Personal communications takes a major role in the coordination of the development work. The basic workflow is implemented in the tool Bugzilla. It is used for managing all development tasks and to synchronize the progress with customer support. A few rules ("rituals") defining the cornerstones of the development work are in place. An example is the manual acceptance testing performed by the CTO before a release.

4.2 Case B

The company studied as Case B develops and manufactures biometric access systems for residential and corporate buildings. The company operates as OEM on the European and international market. Furthermore, the products are also distributed via resellers and partner companies in home and building automation who plan, install, and service access systems as part of their automation projects. With offices in several European countries, the company has more than 70 employees in total.

The product offered by the company is the physical biometric access system, which is available in several different models and designs. The system consists of hardware parts such as scanners, control panels, CPU and RAM, Bluetooth/RFID interfaces, electromechanical locks, and an embedded software system. The software system includes a proprietary embedded OS running several software modules implementing user registration,

biometric algorithms, service functions, etc. For corporate access solutions, a distributed system with server nodes maintaining the user database and access policies has been developed.

The company's R&D department is structured in hardware development, software development, and system testing. About six people are in software development. System testing is done by one person, an ISTQB certified tester. The overall development process is driven by considerations for hardware development and applicable standards for electrical and electronic equipment as well as safety regulations. Requirements, new ideas, and constraints are elicited by product management. They are evaluated, prioritized, and refined into a system requirements specification for hardware and software development. The specification is also the basis for system testing, which is performed for major milestones and before each release. New versions of each of the different product models are released two to three times per year.

A release can introduce new hardware features, new software features, or both. The software is backward-compatible, and bug fixes can be included in a release. However, the customers perceive the product as physical entity and rarely perform software updates themselves. Problems in the field usually require the support by a service technician. To avoid the high expenses involved in service operations, quality assurance is equally essential for the hardware as well as for the software of the product.

4.3 Case C

Case C is a small software company developing a commercial off-the-shelf software product for document management. The company has specialized on the freight forwarding and logistics industry. The software product was certified under IDW PS 880 for software products relevant to billing and provides audit-proofed digital preservation. The software product has been realized as client/server system with .NET technologies and offers a Web service interface to interoperate with other systems. Overall, there exist about 100 product installations at about 50 customer sites.

The company has a strong focus on software development: Eight out of the 10 employees including the CEO are active software developers. Development follows a formal development process which is oriented toward Scrum. It comprises the roles product owner, enterprise architect, developer, and tester. The CEO itself is the product owner and enterprise architect. Development is organized in sprints which last between 1 and 4 weeks and deliver working software that may be released. In a sprint meeting, the plan including duration as well as the user stories to be implemented is defined. Priorities for user stories as well as uncritical bugs fixed in a sprint are assigned by the product owner. Sprints are developed in a development branch. The main branch contains the actual release and only critical bugs. Daily work is organized by holding a daily scrum meeting and by maintaining a Scrum board.

Differing from Scrum, each user story has to be specified and documented. The specification is reviewed by the enterprise architect. The documentation as well as the source code, which has to follow a company-specific code style guide, is reviewed at least by another team member. So each created artifact is peer-reviewed. Case company C applies continuous integration, i.e., the software product is built and automatically tested after each check-in. All software development and testing tasks are managed using Microsoft Team Foundation Server.

4.4 Case D

Case D is a small company providing recruitment and training services as well as a software platform to support the involved business processes and activities. The focus of the provided recruitment and training services is related to SAP. The company is operating worldwide. The software platform, which is implemented in PHP, is open for persons on the job market, who can provide their profile and details about their qualifications, as well as for companies searching for skilled employees. Overall, the company has around 40 employees of which four employees are working in software development and operation. This organizational structure reflects the company's strong focus on providing recruitment and training services. Software has the (important) role of an enabler to perform these services.

The main duty of the software engineers is currently the configuration and operation of the software platform since the case company is not actively developing the platform anymore. The previous software platform was developed by case company, and future platforms may as well be developed by the company again. Configuration and maintenance tasks as well as requests are either managed informally, via email or telephone, or by written notes on a storyboard, which has recently been replaced by the issue management tool JIRA. Testing lies in the responsibility of each individual developer and is performed informally without peer review or documentation. Additional acceptance testing is optionally performed by the product owner.

4.5 Case E

The Case E company is a solution provider that offers billing and cashless payment systems for restaurants and canteens. The company offers a combination of hardware, software, and consulting services to build custom solutions for their clients. The hardware—mainly developed by the company itself—includes, for example, vending machines, point-of-sale terminals, and ordering clients. The software system runs on the distributed hardware clients and terminals, connects them with a central back office server, and provides interfaces for data exchange with ERP and accounting systems. The company handles all project needs of its clients starting from the initial concept, hardware and software specification and development, to installation at the client's site, as well as ongoing service and support.

The company operates on the international market with a network of service partners in different European countries. From 10 to 20 people are employed at the Austrian headquarter, of which about five are working in software development and testing. A full-time employee is dedicated to system testing.

Software development is organized as a series of consecutive projects. Projects fall in one of the following categories: customer projects, internal development projects, or maintenance projects. Projects are divided into a planning phase, an implementation phase, and a release phase. In the planning phase, a requirements specification is developed, a risk analysis is conducted, the project effort is estimated, and the setup for starting the project is defined. The implementation phase starts with a design step followed by the actual implementation. These steps are accompanied by design and code reviews. At the end of the implementation, the results are handed over to testing, initiating a test and stabilization cycle that ends with the release, and—in case of a customer project—the commissioning of

the solution. The developed software features, enhancements, and changes are merged into the product mainline, and thus, they define the new basis for future projects.

The maintenance of a product mainline and the reuse of the existing software modules are essential for evolving the software system in the face of the diversity introduced by supporting multiple custom solutions. Currently, the product mainline consists of about 60 different software modules that can be combined for building a custom solution. Two times per year a new version of each module is released to synchronize all serviced installations in the field.

Furthermore, software development has to cope with various compliance requirements introduced by technical standards, interfaces to accounting systems, and legal regulations. An example is the directive of the Austrian Federal Ministry of Finance that defines criteria that must be taken into account in order to meet legal compliance of cash registers and point-of-sale systems with respect to recording and retention obligations.

5 Results and discussion

In this section, we answer the research questions based on the studied cases. For each question, first, generalized findings are presented, and, second, the related evidence confirming these findings is described and discussed in context of each case.

5.1 How is software testing organized? (RQ1)

To answer the first research question, we investigated what roles are involved in testing, what activities they perform, and what artifacts they use as input or produce as output. For each case, we repeated this investigation for each test level, i.e., unit testing, integration testing, system testing, and acceptance testing. The findings are:

- *F-01 Developers have a high degree of self-responsibility for ensuring quality assurance on code level.* It is mainly their decision whether and how unit and integration testing is performed. Personal preferences and technical feasibility are the main influence factors for performing unit and integration testing. Test results or coverage metrics are not specified and reported.
- *F-02 A mandatory system or acceptance test level (higher-level testing) is in place.* This stage serves as a quality gate prior to the release of the software. Its purpose is to verify and to validate the functionality implemented by the developers.
- *F-03 A specific person in the company has the role of the tester at system or acceptance level.* Although closely related to the development team, this person is not a developer himself. In several cases, testing is the main duty of the person dedicated to system and acceptance testing.
- *F-04 Higher-level testing at the system or acceptance level is based on specified test cases and follows established test protocols*. Tool support is sometimes available for managing the test cases and test executions. The approach for developing the tests is different between the cases and is mostly done informal.
- *F-05 Exploratory testing plays an important role.* Exploratory approaches do not rely on specified test cases but are driven from the knowledge and experience of the tester. Such approaches can be observed on all test levels. In some cases, exploratory testing is the main technique used for system or acceptance testing.

Table 3 Findings for how software testing is organized (The symbol + indicates a confirmation of the finding and − a contradiction; blank if the information is missing.)

F-ID	Short description of the finding	A	B	C	D	E
F1-01	Developers have a high degree of self-responsibility for quality assurance on code level.	+	+	−	+	+
F1-02	A mandatory system or acceptance test level (higher-level testing) is in place.	+	+	+	−	+
F1-03	A specific person in the company has the role of the tester at system or acceptance level.	+	+	+	−	+
F1-04	Higher-level testing at the system or acceptance level is based on specified test cases and follows established test protocols.	−	+	+	−	+
F1-05	Exploratory testing plays an important role.	+	+	−	+	
F1-06	Test documentation is of marginal importance.	+	−	−	+	−
F1-07	The overall degree of automation in software testing is generally low.	+	+	−	+	−
F1-08	Additional software quality assurance measures are applied besides testing.	−	+	+	+	+

- *F-06 Test documentation is of marginal importance.* Apart from the cases where test documentation is explicitly prescribed by standards and policies, it plays a subordinate role in testing. For all cases, the minimum result from testing is the reporting of found defects. Other results are not systematically recorded, analyzed, and reported.
- *F-07 The overall degree of automation in software testing is generally low.* If automated tests exist, they are mainly at the level of unit and integration testing. Automation is limited by time and resource constraints and technical restrictions.
- *F-08 Additional software quality assurance measures are applied besides testing.* Testing is an important measure to assure software quality, but it is not the only one. In all cases, quality assurance is understood as holistic principle relevant for all phases and in all activities of the development process. Therefore, in many cases also other QA measures such as reviews or static analysis are applied. In some cases, software testing is embedded as one out of many activities in an overall quality management system.

Table 3 summarizes the findings for how software testing is organized and shows for each of the five cases whether the findings are confirmed or not.

In **Case A**, every developer is responsible for the quality of the software he or she develops. Thus, every developer also defines the type and amount quality assurance that is performed whenever a change or enhancement is made, which also involves the type and amount of testing. Automated unit tests are in place for several of the client-side modules. The actual number of unit tests is mainly depending on the technical feasibility and the module's testability. Test coverage is not measured. It also remains with the developer how frequently the automated unit tests are executed and whether or not they are maintained when the software system is changed.

The changes and enhancements made by the developers are tested in the form of end-to-end integration or system test scenarios, which are executed manually by the individual developers. The test data sets for these tests are also developed and maintained by the developers. There are no automated regression tests at the integration or system level. The reason is the lack of time and resources that would be necessary for setting up and maintaining the automation infrastructure and for managing the testware.

Once development is ready, every feature undergoes a manual test performed by the CTO. This test is best characterized as acceptance test. It not only verifies that changes and enhancements to features have been implemented correctly, but also validates that the correct changes and enhancements have been implemented, i.e., that they make sense from a user's point of view. Sometimes also customer support personnel are involved in this step. Furthermore, together with acceptance testing the user documentation is updated. The complete testing and stabilization phase before a release accumulates up to about 1 month. The tool Bugzilla is used for managing and supporting the involved testing activities (e.g., covering all changes, keeping track of identified issues, re-testing of fixes).

In **Case B**, the company has a dedicated tester assuring the quality of the hardware and the software integrated in the product. While testing is an activity and a role in R&D, it is at the same time an integral part of the company's overall quality management system (ISO 9001:2008 certified) that spans all process phases and relates to the production of the hardware as well as to the development of the software. Quality management makes sure that software quality assurance tasks are an integral part of all software development activities from requirements engineering and design to releasing and maintenance. Furthermore, testing is also tightly connected with other quality assurance activities. It starts with reviewing the requirements specifications and ends with involving service personnel in system testing.

The entire software development process is structured according to the classical "waterfall" development phases and is synchronized with hardware development. In alignment to the main development phases, the corresponding test levels—unit testing, integration testing, and system testing—are distinguished.

Unit testing and integration testing are the duty of the developers. The approaches for unit testing are determined by the environment and technical restrictions applying to the developed software components (e.g., firmware, algorithms, or server and database modules). For example, GoogleTest is used for testing C++ modules as it supports different hardware platforms. In addition, dynamic testing is complemented with static code analysis. For integration testing with hardware prototypes, the same tool set and similar approaches are used as for unit testing.

System testing is performed on the fully integrated product, i.e., the hardware system plus the software system. The basis for system testing is the system requirements specification. It is the input for designing test cases and defining the test specification, which is reviewed together with hardware and software engineers. The tool RTMR is used to manage the test cases as well as the test execution results. Defects are reported and tracked using Mantis bug tracker. The system test is a purely manual test as the system design requires real biometric data as input. The design prevents the use of simulated ("fake") input data and makes circumventing the security mechanisms impossible. There is no exception even for in-house testing. As a consequence, system testing is a time-consuming task that takes from 1 to 3 months before a release.

In **Case C**, testing takes an important role in the Scrum-based software development process and is performed on the unit, system, and acceptance level. In sprint meetings, testing is planned for user stories. For this purpose, there is an attribute "TestStrategy" assigned to user stories defining whether tests for it have to be defined during a sprint. Testing is entirely managed in Team Foundation Server.

On the unit level, tests are created with the MS Test Framework and their automated execution is triggered when the software is built, i.e., after each check-in. Unit testing is not measured by coverage metrics but reviewed by another developer. Thus, developers

have a limited degree of self-responsibility for conducting quality assurance measures at the code level.

On the system level, user story tests defined in test scenarios are manually executed for each release. A dedicated person is responsible for the complete execution and documentation of the specified test scenarios. Additional exploratory testing does not play an important role. Detected bugs are classified as critical or uncritical. Critical bugs are immediately fixed on the main branch, whereas uncritical bugs are fixed in the development branch during the next sprint.

Finally, on the acceptance test level, release candidates are deployed internally as the created software is used by the company itself for document management. The team members then perform a beta test of the deployed software for 1 week before it is delivered to customers. Critical bugs detected in this phase may be resolved in the main branch or—if this is risky—in a separate bug fix release.

Furthermore, reviews are used as additional quality assurance measure. Thus, in addition to testing on the unit, system, and acceptance level, all artifacts (e.g., source code, test cases, documentation, and bug fixes) are also subject to reviews.

In **Case D**, every developer is responsible for the quality of the software he or she develops. Thus, every developer also defines the type and amount of quality assurance that is performed whenever a change is made. In addition to these unit-level tests, the product owner performs acceptance tests on demand. Both, unit- and acceptance-level tests, are performed exploratory based on the experience of the tester. Neither unit-level testing nor acceptance-level testing is automated, documented, or following specific test end or exit criteria. Bugs are reported either via a contact form on the software platform or informally via email or telephone, and they are resolved on demand. Besides exploratory testing, a separate quality management measure is dedicated to the monitoring and evaluation of delivered services. It regularly collects customer feedback for the recruitment and training services including the software platform as a basis for continuous improvement measures.

Software testing in **Case E** is organized in two parallel tracks: first, as part of the software development projects and, second, on the product main line in form of regression testing. In the development projects, testing is only one of many quality assurance measures. Other measures are, for example, risk analysis, design reviews, code reviews, and progress reports. Code and design reviews are the main quality assurance measure in place for at the implementation level. In addition, developers perform unit and integration testing. However, it is up to the individual developer whether and to what extent unit and integration testing is actually performed.

At the system test level, a dedicated tester defines a project-specific test plan and derives the test cases from the software requirements specification. The test environment for system testing (i.e., the hardware and network configuration, the test data, the included software modules, and their configuration) is set up to match the customer's actual environment. The specified test cases are executed manually, and, later on, a selection of these test cases is implemented for regression testing with the test automation tool Ranorex. In the development projects, system testing is performed in short test and stabilization cycles with direct interaction between the tester and the developers. In-house system testing as part of the development project is followed by commissioning and a beta test phase at the customer site.

The automated regression tests are taken over into the product main line together with the newly implemented features, enhancements, and changes. These tests are system tests that exercise the software system via the user interface. The setup of the test environment follows a reference system covering the most typical configurations in the field. The test

cases are periodically executed by the build system. Defects are reported and tracked with the Bugzilla issue tracker.

5.2 What is the role of risk in software testing? (RQ2)

To answer this research question, we investigated the five cases under the light of the four research questions (*RQ2a* to *RQ2d*) that also guided the previous study on risk-based testing (Felderer and Ramler 2014b) we used for comparison.

The findings w.r.t. the research question **RQ2a** *How do risks support the activities of the software test process?* are:

- *F2-01 Risk considerations focus mainly on business risks or project risks.* Following the distinction of product (technical) risks, project (development) risks, and business risks, it can be observed that risk considerations are mainly associated with issues that threaten the economic success of a project or product. Even though risks are discussed in context of the actual product, they usually focus on critical product functionality triggering business risks.
- *F2-02 The members of development have a high awareness of business risks.* Implications on the company's business success are usually well known by the key members of the development team, regardless of their role (e.g., developer, tester, or product manager).
- *F2-03 The term risk remains undefined, and risk-related considerations capture mainly the impact side.* Commonly, risk is defined by the factors impact and probability. In the context of the studied companies, however, the main question is about the impact—"What are the cost or severity of a failure if it occurs in operation?". The likelihood that a failure occurs is much less or not at all discussed.
- *F2-04 Risk is an implicit concept and relies on subjective perception.* The studied companies are generally aware of the risks they take, but these risks are usually not made explicit. Risks are not systematically identified or analyzed. As a result, risks remain in the state of subjective opinions.

The findings w.r.t. **RQ2b** *How do risks support the activities of the software test process?* are:

- *F2-05 Risk is considered in all testing activities, even when testing does not explicitly follow a risk-based approach.* The analyzed cases include examples where risk is used to design new test cases, to classify test cases according to a general "priority," to select test cases for regression testing, to prioritize test cases for test execution, and to define the test end criterion.
- *F2-06 The understanding of risks is used to focus testing on "risky" areas.* The knowledge of risks is used in testing in two ways: first, as a guideline to adjust the focus of testing to "risky" areas where most critical problems are located and, second, to further extend the scope of testing toward such "risky" areas.
- *F2-07 The understanding of risks is used to adjust the amount of testing, the overall test effort, or the established test budgets.* Risk-oriented testing is understood as an approach to adjust (i.e., to shrink or extend) the amount of testing, the overall test effort, or the test budget to the perceived business criticality. The adjustment may result in a reduced amount of testing to keep the quality assurance "overhead" low, but also in additional testing in order to address all risks.

The finding w.r.t. **RQ2c** *How are risks organized from a technical perspective?* is:

- *F2-08 Risks are only implicit, and thus, no representation and tool support exist.* As observed earlier, risks are usually not made explicit. Thus, risks are not represented as tangible entities, and furthermore, there is no representation in existing tools.

The findings concerning **RQ2d** *What are the potential benefits of a risk-based testing approach?* are:

- *F2-09 Make testing more effective: detecting additional defects in testing, fewer defects slip through to the field.* A central benefit of using risk information in testing is to increase the range of testing with additional risk-based test cases and, thus, to increase the chance of detecting additional defects. As a result, risk-based testing helps to make testing more effective. It is expected that fewer defects will slip through to the field.
- *F2-10 Make testing more efficient: selection of tests based on risks lead to a reduction of cost and time for testing.* Testing is usually limited by time and resource constraints. Thus, the perceived risks are used to improve testing efficiency by using the available time and resources to cover at least the most critical parts.
- *F-11 Risk is used as rationale and motivation for the application of QA measures.* Risks in terms of the impact of defects for the user and customer and, in consequence, for the company's business success are the main rationale for applying QA measures. The high risk awareness of all development members explains and motivates decisions to apply specific QA measures.

Table 4 summarizes the findings on the role of risk in software testing and shows for each of the five cases whether the findings are confirmed or not.

In **Case A**, risks are associated with problems that have a direct impact on the user or customer. An example for such a problem would be a defect that blocks mission-critical processes of the customer, for example offer or billing processes. As a consequence, such a defect would pose a risk for the company's business success. The company is operating in a niche market where all players are well known and news spread quickly. The link between risks and business success is also evident in another example given for what is perceived as risk: "the development of a feature that is not usable or not needed by the customer."

Defects are not risks per se. It is the impact that is seen as risk. From the company's perspective, thus, it is the potential damage of a defect. Yet as long as the service and support team quickly responds and resolves the defect, its impact—and thus the perceived risk—is low. Although the migration of the product to a new technology platform was a risky endeavor for the company, they have not lost any customers because of the defects introduced in the course of the migration.

Risk orientation, however, is also seen as an attitude for reducing the overall effort and costs. For example, the members of development share the common goal to keep the effort for service and support low. All members have a good understanding of the effect of their work (and the risks involved) on the company's overall results. This is reflected in the conclusion of a developer: "At the end of the year, the financial statement will show the quality we achieved."

Despite the general risk orientation, risks are not identified, analyzed, and documented. An attempt to introduce risk management has been abandoned due to a lack of time and resource. So the knowledge about the risks is based on personal experience and subjective opinions that are shared between the members of the company. When specific software modules, enhancements, or changes are considered particularly "risky," this information is passed from the developer to the tester. At the acceptance test level, an exploratory testing

Table 4 Findings on the role of risk in software testing (+ indicates the confirmation and − the contradiction of the finding by the studied case)

F-ID	Short description of the finding	A	B	C	D	E
F2-01	Risk considerations focus mainly on business risks or project risks	+	+	+	+	−
F2-02	The members of development have a high awareness of business risks	+	+	+	+	+
F2-03	The term risk remains undefined and risk-related considerations capture mainly the impact side	+	−	+	+	−
F2-04	Risk is an implicit concept and relies on subjective perception	+	−	+	+	−
F2-05	Risk is considered in all testing activities, even when not following an explicit risk-based testing approach	+	+	+	+	+
F2-06	The understanding of risks is used to focus testing on "risky" areas	+	+	+	+	+
F2-07	The understanding of risks is used to adjust the amount of testing, the overall test effort, or the established test budgets.	+	−	+	+	+
F2-08	Risks are only implicit and, thus, no representation and tool support exists	+	+	−	+	−
F2-09	Make testing more effective: detecting additional defects in testing, fewer defects slip through to the field	+	+	+	+	+
F2-10	Make testing more efficient: selection of tests based on risks lead to a reduction in cost and time for testing	+	−	+	+	+
F2-11	Risk is used as rationale and motivation for the application of QA measures	+	+	+	−	+

approach is applied. The risk information is used to adjust the amount of testing and to focus the available time on those parts of the system with the highest risk.

The product of the **Case B** company is characterized by its high reliability and robustness. These characteristics are considered typical for a hardware product that has an expected lifespan of more than 10 years. As a consequence, similar characteristics have to be shown by the embedded software system. Important metrics are the failure rates of admitted and denied accesses (1 per 1,000,000 for access and 1 per 1000 for denial) as well as the identification performance (<3 s per request). Defects that affect these metrics are considered highly critical as they have a direct impact on the end-user experience.

Another risk in offering a hardware product lies in production errors, which increase the "need to get things done right the first time." Since customers rarely perform software updates themselves, costs for service and support can be dramatic if all products of a line are affected by a defect. Hence, a quality management system is in place to govern hardware development, software development, and the productions process. It includes the use of process and product metrics evaluated by product management as well as subsequent risk management practices. Concerning software development, however, the focus of risk management is mainly on project risks affecting the development schedule rather than on product risks directly relevant for testing.

The basis for deriving test cases in system testing is the reviewed system requirements specification. Nevertheless, the tester's experience and knowledge about product risks play an important role in designing test cases and in test execution, which is a purely manual

process. Although risk information is considered in testing, it is currently not used to shrink test cycles or to reduce the amount of testing in order to save time and costs. On the contrary, risks are seen as rationale and motivation for more rigorous testing.

In **Case C**, risks are mainly related to critical product functionality like auditing acceptability and billing. As the company's overall success is strongly connected with the software product, critical product risks immediately trigger business risks. The key members of the development team are aware of the relationship between product and business risks. User stories and bugs are prioritized by the product owner, implicitly taking the related business criticality into account. Yet the notion of risk is not explicitly defined. It relies on an individual perception and subjective estimation, and it remains subjective during software development and testing.

Risk is also implicitly considered for test planning, design, implementation, and evaluation. Testing for each user stories is planned during sprint meetings. There are three options in testing—called "test strategies"—which take the type but also the risk of a user story into account: Unit tests are created and included in the automatic build process for continuous execution; test scenarios are created and executed for each new release; manual testing by reviewers if due to small changes, other forms of testing are not performed. Also for test design, risks are implicitly taken into account by defining more comprehensive test scenarios in case "risky" areas are affected. Furthermore, critical bugs observed during beta testing are considered a high risk for the release, and they are therefore not fixed in the main branch but in a separate bug fix release. Thus, risk orientation helps to focus testing on risky functionality, to detect the most critical defects first, and to avoid critical bugs in operation. Finally, risks are also recognized as a measure to motivate additional quality assurance measures like test-driven development.

The overall test and development process including all related artifacts is managed with Microsoft's Team Foundation Server. Although this would easily allow implementing risk-based testing support, this possibility is not taken in practice. For user stories, there even is an attribute "risk" with values low, medium, and high, but this attribute is neither used in practice nor are risks made explicit. Nevertheless, the implicit risk orientation of testing helps to guarantee that fewer defects slip through to the field while keeping the overall number of tests and, thus, the time needed for testing reasonable.

In **Case D**, the software platform is an enabler for providing business services. The main risks are not considered directly related to the software platform but to business and operations. With regard to the business, loosing major customers is considered the highest risk. With regard to operations, server downtimes and data privacy issues are considered critical. The most important parts of the software platform are its usability and the functionality relate to payment. These parts have to be tested carefully to avoid problems due to misconfigurations. The members of the development team are aware of potential risks induced by the software platform, although these risks remain implicit and are based on subjective perception. Overall, risks are mainly associated with their impact.

Both testing by developers and the product owner are performed exploratory. Risks are implicitly taken into account in test planning, design, execution, and evaluation. Risk information is used to extend testing toward risky areas and control the overall testing effort. Furthermore, it is assumed that the implicit knowledge about risks also helps to identify critical defects.

Since risks remain implicit, tool support is neither needed nor available. Even bugs are mostly fixed on demand, and they are not explicitly classified based on their criticality. On the management level, business risks are systematically managed, but this information is not made available for software testing.

As a solution provider, the **Case E** company understands the needs and concerns of their customers and, moreover, the customers of their customers, i.e., the clients of a restaurant using the cashless payment system. Billing issues or service delays can be disastrous, especially when happening at peak times in a restaurant. Therefore, the company is highly concerned about the impact of possible defects as they pose a direct threat to the company's business success. Legal regulations and compliance requirements are another source of risks with a potentially huge impact. Several modules are subject to costly and time-consuming certifications, which have to be repeated in case of modifications due to enhancements or bug fixes.

To proactively address the various risks involved in solution development, the company has established a process for development projects that specifies several quality assurance measures. Among them are risk management, design and code reviews, unit testing, and system testing. The actual set of quality assurance measures is defined in the project's planning phase based on a risk assessment.

The central tool is a risk assessment sheet used for enumerating product risks. Typical examples for recorded risks are failure scenarios blocking essential functions of the system. For each risk, the affected parts of the system, probability, and impact estimates, as well as contingency measures are specified. Testing and other QA practices are typical example for such measures. If testing is proposed as contingency measure, the affected system parts, functions, or configurations are specified that should be included in the test plan for extended testing.

Risk information—although less formalized—is also part of other QA practices and made available for tasks on the product mainline, where all enhancements and changes from the projects are integrated. For example, the knowledge about risks is used for selecting test scenarios for automated regression testing and for defining the relevant configurations required to set up the test environment.

5.3 What are the differences between risk-based testing in SME and large enterprises? (RQ3)

To answer this research question, we compared the findings from research question RQ2 to the findings from our previous study on risk-based testing in large enterprises (Felderer and Ramler 2014b). The findings derived from this comparison (see Table 5) are:

- *F3-01 Risk considerations in SME are at the level of business risks, while testing in large enterprises is focused on product risks.* Risk perception in SME is dominated by business risks and project risks. Developers and testers have a high awareness of the business risks, and they act accordingly. In contrast, testing in large enterprises is understood as measure (i.e., a protective action) that addresses primarily product risks.
- *F3-02 In SME, the term risk is usually not defined and risk-related considerations involve mainly the impact side, whereas large enterprises have a more complete understanding of risk.* For most SME, the term risk is undefined and remains a generic, implicit concept. In these cases, the term risk is not present other than in habitual language use where it relates to the impact side only. Large enterprises tend to have a more complete understanding of risk based on the common risk definition ($R = P \times I$) including both components, probability P and impact I. This finding conforms with a result of a systematic literature review on software process improvement in SME by Pino et al. (2008) stating that measurement of process improvement is performed through informal and non-objective processes, based on the

Table 5 Comparison of findings on risk-based testing in SME versus large enterprises

F3-ID	Findings from SME	Rel.	Findings from large enterprises
F3-01	F2-01: Risk considerations focus mainly on business risks or project risks	#	F-1: Testing is a measure to reduce risk exposure, the focus is on product risks, the risk concept is dependent on context and scope
	F2-02: The members of development have a high awareness of business risks		–
F3-02	F2-03: The term risk remains undefined and risk-related considerations capture mainly the impact side	<	F-2: Common definition of risk $R = P \times I$ applies, definition and relation of P and I may remain informal and implicit, usually based on (subjective) estimates
F3-03	F2-04: Risk is an implicit concept and relies on subjective perception	<	F-3: Degree of formality of risk depends on the application scope, formality increases with wider scope and abstraction level
	F2-05: Risk is considered in all testing activities, even when not following an explicit risk-based testing approach	=	F-4: Risk is considered in all testing activities, even when not following an explicit risk-based testing approach
F3-06	–		F-5: Risk information from testing is not explicitly considered in making release decisions
F3-05	F2-06: The understanding of risks is used to focus testing on "risky" areas	<	F-6: Risk information is used to extend testing toward "risky" areas, risk information is used as a guideline to focus testing on "risky" areas
F3-04	F2-07: The understanding of risks is used to adjust the amount of testing, the overall test effort, or the established test budgets.	#	F-7: Risk-based testing is not used to reduce the amount of testing, the overall test effort, or the established test budgets
F3-03	F2-08: Risks are only implicit, and thus, no representation and tool support exists	<	F-8: Risks at the level of project management are individual entities, risks in testing are calculated from risk information associated with testable entities
	–		F-9: Risk-based testing relies on standard test tool infrastructure, risk management tools are not applied in software testing
	–		F-10: Implementation of risk-based testing uses an extensions to standard test tools
	–		F-11: Tool support for risk-based testing is an aid for collecting and analyzing measurement data from which to derive risk information
F3-05	F2-09: Make testing more effective: detecting additional defects in testing, fewer defects slip through to the field	=	F-12: Make testing more effective: detecting additional defects in testing, fewer defects slip through to the field
F3-04	F2-10: Make testing more efficient: selection of tests based on risks lead to a reduction in cost and time for testing	<	F-13: Make testing more effective: prioritization for detecting most critical defects first, reduces overall stabilization costs and time
F3-06	F2-11: Risk is used as rationale and motivation for the application of QA measures	<	F-14: Risk information used for informed decision-making and new insights to triangulate and refine decisions

employees' perception and not through formal measurement processes. Furthermore, finding F3-02 also conforms with the role of risk management in the minimal test practice framework of Karlström et al. (2005), where the identification of problem areas should avoid or reduce the impact of issues within these areas. But differing from Karlström et al. (2005), where risk management is considered only for enterprises with 30 and more employees in development, risk is according to our finding considered in all SME independent of their size.

- *F3-03 The low degree of formality of risk in SME inhibits risk representation and tool support, which exists in large enterprises.* In the studied SME cases, the risks remain usually subjective opinions that are not made explicit by representations or with the support of tools. In contrast, large enterprises link risk with available information such as measurement data and some use risk models. To include this information for risk-based testing, they rely on existing tools for test management or extensions to the company's standard tool infrastructure.
- *F3-04 Other than in large enterprises, in SME the perceived risks are used for increasing efficiency, i.e., the amount of testing is adjusted to reduce cost and time.* Market success and competition are often the driving forces for SME. The business environment implies severe time and resource constraints. Thus, as Martin et al. (2007) put it, there are "good organizational reasons for bad software testing." In this context, the understanding of risks is used to make testing more efficient, i.e., to adjust the amount of testing to optimally use the available time and resources for covering at least the most critical parts. Time and resource constraints also apply to large enterprises, especially when investigating the testing activities in individual projects. However, no cases have been observed where risk-based testing has been understood as an approach to reduce the amount of testing.
- *F3-05 In SME and in large enterprises, the risk information is used for increasing effectiveness, e.g., by focusing on risky areas to find more defects.* Both—SME and large enterprises—aim to make testing more effective by using risk information. Based on identified or perceived risks, they increase the range of testing with additional risk-based test cases, and thus, they increase the chance of finding defects before they slip through to the field. Nevertheless, large enterprises use risk information in more ways: first, for extending the scope of testing in order to cover risky areas in greater depth with additional tests and, second, for prioritizing testing activities to primarily target risky areas and to detect critical defects earlier. We have not observed this "advanced" aim in SME, mainly due to their priority to make testing most efficient by reducing the involved cost and time as indicated in the previous finding.
- *F3-06 Risk-based testing in SME is an integral part of QA, inseparable from other practices, while it has the status of a distinct practice in large enterprises.* The overall amount of testing, the dedicated resources, and the time spent for testing in SME are much smaller than in large enterprises. Testing in SME is an activity that is tightly integrated in the wider range of established QA practices. Thus, risk orientation is a general matter associated with activities from the company level down to QA and testing. In contrast, there are clearly defined boundaries for testing in large enterprises, which also define the scope of risk-based testing. The link between risk-based testing and other risk-based activities (in QA, project management, etc.) is established by exchanging risk information.

6 Threats to validity

In this section, we present the threats to validity of our results and the applied countermeasures. Referring to Runeson et al. (2012), we discuss threats to the construct validity, reliability, internal validity, and external validity of our multiple case study along with countermeasures taken to reduce the threats.

6.1 Construct validity

Construct validity reflects to what extent the phenomenon under study really represents what the researchers have in mind and what is investigated according to the research questions. Of specific interest to construct validity are the definitions of terms and concepts in the case context versus the research context. We defined all relevant terms and concepts of risk-based testing in Sect. 2. The research questions are formulated based on these terms and are in alignment with the previous study on risk-based testing in large enterprises (Felderer and Ramler 2014b), which is used to investigate differences and commonalities of risk-based testing in SME and large enterprises. The notation of each case and therefore also interview questions are based on the terms and concepts of the case. These terms and concepts are linked to the general terms and concepts of risk-based testing.

6.2 Reliability

Reliability focuses on whether the data are collected and the analysis is conducted in a way that it can be repeated by other researchers with the same results. This is a threat in any study using qualitative and quantitative data. The observations are of course filtered through the perception and knowledge profile of the researchers. Counteractions to these threats are that two researchers are involved in the study, the data collection and analysis procedures are well documented, and the data extracted from documents and tools is triangulated with interviews and some quantitative data.

6.3 Internal validity

Internal validity is of concern when causal relations are examined. When the researcher is investigating whether one factor affects an investigated factor, there is a risk that the investigated factor is also affected by a third factor. In our case, only few quantitative data are available, the quantitative analysis is not interpreted in isolation, and it is not even feasible to infer statistical analysis, due to the incompleteness of the data. The analyses about casual relationships are instead based on qualitative analysis. Feeding back the analysis results to interviewees is another action taken to reduce internal validity threats.

6.4 External validity

External validity is concerned with to what extent it is possible to generalize the findings and to what extent the findings are of interest to other people outside the investigated cases. As the cases have been carefully selected, the presented results are based on a broad basis on how (risk-based) testing is performed in SME, which makes the results relevant for other contexts as well. Each case of course has its own specifics, and in that sense, there is no general case. However, some key characteristics of the cases may be general, and for

other cases with similar contexts, the results may be used as a reference. In order to allow external comparison, we have presented the context of each case as clearly as possible, given the confidentiality constraints we have. Replicated studies may further strengthen the external validity of our study.

7 Conclusion

In this article, we explored the risk orientation in software testing of SME and its differences to large enterprises by a multiple case study. Five SME have been selected as cases for this study. Case A develops a commercial off-the-shelf software product for enterprise resource planning, Case B develops and manufactures biometric access systems, Case C develops a commercial off-the-shelf software product for document management, Case D provides recruitment and training services as well as a software platform to support these, and finally, Case E is a solution provider offering billing and cashless payment systems for restaurants and canteens.

The main analysis across the five cases was conducted with qualitative methods. Results were gained by analyzing documents, tools, as well as interview records and workshop results, keeping a clear chain of evidence. Based on the research objective to explore the risk orientation in software testing of SME and its differences to large enterprises, three research questions were defined and the findings were derived by cross-case analysis. In addition, the results from a previous multiple case study on risk-based testing in large enterprises (Felderer and Ramler 2014b) were taken as a reference for comparing the state of the art in SME and large enterprises.

With regard to *how software testing is organized in SME*, the main findings are that developers have a high degree of self-responsibility for ensuring quality assurance, mandatory system and acceptance-level testing—in some cases based on specified test cases—is in place. Exploratory testing plays an important role, the overall degree of automation is generally low due to resource and time limitations, and a range of additional software quality assurance measures are applied in combination with testing.

Regarding the *role of risk in software testing of SME*, we found that risk considerations focus mainly on business risks, the concept of risk is implicit and relies on subjective perception, and risk is considered in all testing activities and used to make testing more effective (by focusing on risky areas to detect additional defects) and more efficient (by reducing testing time and cost). Finally, risk is used as rationale and motivation for the application of QA measures.

The main *difference of risk-based testing in SME compared to large enterprises* is that the concept of risk is mainly focused on the level of business risks and usually undefined and informal, which inhibits risk representation and tool support. Furthermore, in SME the perceived risks are used for increasing efficiency, i.e., the amount of testing is adjusted to reduce cost and time. Finally, risk-based testing in SME is an integral part of QA inseparable from other practices, while it has the status of a distinct practice in large enterprises.

In future, we plan to develop guidelines and to derive best practices for risk-based testing in SME, which will incorporate the findings from this multiple case study. We also aim to develop an approach for estimating risk probabilities based on defect data to increase the awareness for risk probability in a pragmatic way, suitable for the SME context. Furthermore, our plans include adapting and extending the stage model for introducing risk-based testing (Felderer and Ramler 2014a). Our findings on how risks support the testing activities provide

important insights for making such a model useful for SMEs. Finally, we plan to conduct additional case studies to investigate the long-term effect of risk-based testing and to further increase the empirical evidence in this research area.

Acknowledgments This work has been supported by the research project Smart Testing funded by the Austrian Research Promotion Agency (FFG), the COMET Competence Center program of the Austrian Research Promotion Agency (FFG), and the project QE LaB—Living Models for Open Systems (www.qe-lab.at) funded by the Austrian Federal Ministry of Economics (Bundesministerium für Wirtschaft und Arbeit).

References

Ahern, D., Clouse, A., & Turner, R. (2008). *CMMI distilled: A practical introduction to integrated process improvement*. Boston: Addison-Wesley Professional.

Amland, S. (2000). Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software, 53*(3), 287–295.

Azar, J., Smith, R. K., & Cordes, D. (2007). Value-oriented requirements prioritization in a small development organization. *IEEE Software, 24*(1), 32–37.

Bath, G., & van Veenendaal, E. (2014). *Improving the test process*. Rocky Nook: Massachusetts.

Dorling, A. (1993). SPICE: Software process improvement and capability determination. *Software Quality Journal, 2*(4), 209–224.

Erdogan, G., Li, Y., Runde, R. K., Seehusen, F., & Stølen, K. (2014). Approaches for the combined use of risk analysis and testing: A systematic literature review. *International Journal on Software Tools for Technology Transfer, 16*(5), 627–642.

Felderer, M., Haisjackl, C., Breu, R., & Motz, J. (2012). Integrating manual and automatic risk assessment for risk-based testing. Software quality. In *Process automation in software development. 4th international conference SWQD* (pp. 159–180).

Felderer, M., & Ramler, R. (2013). Experiences and challenges of introducing risk-based testing in an industrial project. In *Software quality. Increasing value in software and systems development. 5th international conference SWQD* (pp. 10–29).

Felderer, M., & Ramler, R. (2014a). Integrating risk-based testing in industrial test processes. *Software Quality Journal, 22*(3), 543–575.

Felderer, M., & Ramler, R. (2014b). A multiple case study on risk-based testing in industry. *International Journal on Software Tools for Technology Transfer, 16*(5), 609–625.

Felderer, M., & Schieferdecker, I. (2014). A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer, 16*(5), 559–568.

Felderer, M., Wendland, M-F., & Schieferdecker, I. (2014). Risk-based testing. In *Leveraging applications of formal methods, verification and validation. Specialized techniques and applications* (pp. 274–276). Berlin, Heidelberg: Springer.

García, I., Pacheco, C., Mendoza, E., Calvo Manzano, J. A., Cuevas, G., & San Feliu, T. (2012). Managing the software process with a software process improvement tool in a small enterprise. *Journal of Software: Evolution and Process, 24*(5), 481–491.

Gerrard, P., & Thompson, N. (2002). *Risk based e-business testing*. Norwood: Artech House Inc.

Gleirscher, M., Golubitskiy, D., Irlbeck, M., & Wagner, S. (2014). Introduction of static quality analysis in small-and medium-sized software enterprises: Experiences from technology transfer. *Software Quality Journal, 22*(3), 499–542.

Harrold, M. J. (2000). Testing: A roadmap. In *Proceedings of the conference on the future of software engineering* (pp. 61–72).

ISO/IEC. (2011). Software engineering—Lifecycle profiles for Very Small Entities (VSEs). Available online at http://www.iso.org/iso/catalogue_detail?csnumber=51150. Accessed on July 15, 2015.

ISO/IEC/IEEE. (2013). ISO/IEC/IEEE 29119 Software testing. Draft available online at http://www.softwaretestingstandard.org/. Accessed on July 15, 2015.

ISTQB. (2012). *Standard glossary of terms used in software testing. Version 2.2*. Brussels: International Software Testing Qualifications Board.

Karlström, D., Runeson, P., & Norden, S. (2005). A minimal test practice framework for emerging software organizations. *Software Testing, Verification and Reliability, 15*(3), 145–166.

Karolak, D. W. (1995). *Software engineering risk management*. Hoboken: Wiley-IEEE Computer Society Press.

Kautz, K. (1999). Making sense of measurement for small organizations. *IEEE Software, 16*, 14–20.

Koomen, T., & Pol, M. (1999). *Test process improvement: A practical step-by-step guide to structured testing*. Boston: Addison-Wesley Professional.
Koomen, T., van der Aalst, L., Broekman, B., & Vroon, M. (2006). *TMap next, for result-driven testing*. 's-Hertogenbosch: UTN Publishers.
Martin, K., & Hoffman, B. (2007). An open source approach to developing software in a small organization. *IEEE Software, 24*(1), 46–53.
Martin, D., Rooksby, J., Rouncefield, M., & Sommerville, I. (2007). 'Good' organisational reasons for 'Bad' software testing: An ethnographic study of testing in a small software company. In *29th international conference on software engineering (ICSE '07)*.
Mc Caffery, F., Taylor, P. S., & Coleman, G. (2007). Adept: A unified assessment method for small software companies. *IEEE Software, 24*(1), 24–31.
Mishra, D., & Mishra, A. (2008). Software process improvement methodologies for small and medium enterprises. In *9th international conference on product-focused software process improvement (PROFES 2008)*.
Pino, F. J., García, F., & Piattini, M. (2008). Software process improvement in small and medium software enterprises: A systematic review. *Software Quality Journal, 16*(2), 237–261.
Pino, F. J., Pardo, C., García, F., & Piattini, M. (2010). Assessment methodology for software process improvement in small organizations. *Information and Software Technology, 52*(10), 1044–1061.
Redmill, F. (2004). Exploring risk-based testing and its implications. *Software Testing Verification and Reliability, 14*(1), 3–15.
Redmill, F. (2005). Theory and practice of risk-based testing: Research Articles. *Software Testing Verification and Reliability, 15*(1), 3–20.
Richardson, I., & von Wangenheim, C. G. (2007). Why are small software organizations different. *IEEE Software, 24*(1), 18–22.
Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012). *Case study research in software engineering: Guidelines and examples*. Hoboken: Wiley.
Sanchez-Gordon, M. L., O'Connor, R. V., & Colomo-Palacios, R. (2015). Evaluating VSEs viewpoint and sentiment towards the ISO/IEC 29110 standard: A two country grounded theory study. In *15th international conference on software process improvement and capability determination (SPICE 2015)*.
Souza, E., Gusmão, C., & Venâncio, J. (2010). Risk-based testing: A case study. In *seventh international conference on information technology: New generations (ITNG)* (pp. 1032–1037).
Souza, E., Gusmão, C., Venâncio, J., & Melo, R. (2009). Measurement and control for risk-based test cases and activities. In *10th Latin American Test Workshop (LATW'09)* (pp. 1–6).
Steiner, M., Blaschke, M., Philipp, M., & Schweigert, T. (2012). Make test process assessment similar to software process assessment—The test SPICE approach. *Journal of Software: Evolution and Process, 24*(5), 471–480.
van Veenendaal, E., Goslin, A., Olsen, K., O'Hara, F., Miller, M., Thompson, G., & Wells, B. (2008). *Test Maturity Model integration (TMMi) Version 1.0.*. TMMi Foundation: Princeton.
Yin, R. K. (2014). *Case study research: Design and methods*. Thousand Oaks: Sage Publications.
Yoon, H., & Choi, B. (2011). A test case prioritization based on degree of risk exposure and its empirical evaluation. *International Journal of Software Engineering and Knowledge Engineering, 21*(02), 191–209.

Michael Felderer is a senior researcher and project manager within the Quality Engineering research group at the Institute of Computer Science at the University of Innsbruck, Austria. He holds a PhD in Computer Science. His research interests are software testing and software quality in general, requirements engineering, empirical software engineering, software processes, and improving industry–academia collaboration. He works in close collaboration with industry and transfers his research results into practice as a consultant and speaker on industrial conferences.

Rudolf Ramler is a researcher at the Software Competence Center Hagenberg, Austria. His research interests include software testing, quality management, and requirements engineering. He has led research projects as well as tool development for test automation, test management, and quality monitoring. Rudolf works as a consultant in industry projects and is a lecturer with the Upper Austria University of Applied Sciences at Hagenberg and the Vienna University of Technology. He studied Business Informatics and holds a MSc (2001) from the Johannes Kepler University of Linz, Austria.