Int J Softw Tools Technol Transfer (2014) 16:609–625
DOI 10.1007/s10009-014-0328-z

RBT

A multiple case study on risk-based testing in industry

Michael Felderer · Rudolf Ramler

Published online: 24 June 2014

Abstract In many development projects, testing has to be conducted under severe pressure due to limited resources and a challenging time schedule. Risk-based testing, which utilizes identified risks of the system for testing purposes, has a high potential to improve testing as it helps to optimize the allocation of resources and provides decision support for management. But for many organizations, the integration of a risk-based approach into established testing activities is a challenging task, and there are several options to do so. In this article, we analyze how risk is defined, assessed, and applied to support and improve testing activities in projects, products, and processes. We investigate these questions empirically by a multiple case study of currently applied risk-based testing activities in industry. The case study is based on three cases from different backgrounds, i.e., a test project in context of the extension of a large 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 telecommunication solutions. By analyzing and comparing these different industrial cases, we draw conclusions on the state of risk-based testing and discuss possible improvements.

Keywords Risk-based testing · Case study research · Multiple case study · Test process improvement · Test management · Risk management · Software testing · Software quality

M. Felderer (✉)
University of Innsbruck, Innsbruck, Austria
e-mail: Michael.Felderer@uibk.ac.at

R. Ramler
Software Competence Center Hagenberg, Hagenberg, Austria

1 Introduction

Risk-based testing is a pragmatic and well-known approach for aligning testing activities with business value and risks. It is based on the intuitive idea to focus testing activities on those areas that trigger the most critical situations for a software system [1]. Its appropriate application may then have several advantages. Risk-based testing is a means for mitigating risks, helps to improve the allocation of resources (budget, time, and persons), identifies critical areas earlier, and provides decision support to the management. For its practical application, guidelines and best practice information is required. So far, research has collected only a small body of empirical evidence on risk-based testing approaches.

The research objective of this article is, therefore to, empirically explore and describe how risk is defined, assessed and applied to support and improve testing activities for industrial projects, products, and processes. The case study methodology is well suited for this kind of research objective, as the objects of study are contemporary phenomena, i.e., risk-based testing activities, which can only be studied in their context and not in isolation [2]. The article presents a multiple case study [3] on currently applied risk-based testing approaches in industry. It is based on three industry cases from different backgrounds to investigate risk-based testing implementations in practice. Case A is a test project in the context of the extension of a large Web-based information system. Case B investigates product testing of a measurement and diagnostic equipment for the electrical power industry. Finally, Case C considers the test process of a system integrator of telecommunication solutions for transportation organizations and mobile network operators.

Although several risk-based testing approaches have been proposed in the literature [4–8], and the recently published international standard series ISO/IEC/IEEE 29119 [9] on

software testing even requires the consideration of risks as an integral part of the test planning process, there are no empirical studies available that analyze the state of risk-based testing in practice on the basis of multiple cases. Therefore, this article contributes a multiple case study which explores different industrial risk-based testing implementations by a cross-case analysis to draw conclusions on the state of risk-based testing in practice.

The remainder of this article is structured as follows. Section 2 provides background and related work on risk-based testing. 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 discusses the analyzed cases, and Sect. 6 presents threats to validity. Finally, Sect. 7 provides a summary and draws conclusions.

2 Background and related work

In this section, we discuss background and related work on risk-based testing. Risk-based testing is a type of software testing that explicitly considers risks of the software product as the guiding factor to solve decision problems in all phases of the test process, i.e., test planning, design, execution and evaluation. It is based on a risk model (Sect. 2.1), defines a test process (Sect. 2.2) and has to be introduced into an existing development and test process (Sect. 2.3). In addition, we present related empirical studies on risk-based testing (Sect. 2.4).

2.1 Risk model

A risk is the chance of injury, damage or loss and is typically determined by the probability of its occurrence and its impact. The standard risk model [10,11] applied in most risk-based testing methodologies is based on the two factors probability (P), determining the likelihood that a failure assigned to a risk occurs, and impact (I), determining the cost or severity of a failure if it occurs in operation. Quantitatively, the risk exposure or risk value R of an arbitrary risk item a, i.e., an artifact to which risks are assigned, is determined based on the probability value P and the impact value I in the following way:

$$R(a) = P(a) \circ I(a)$$

A risk item a is an arbitrary artifact, e.g., a requirement, a component, a security risk or a failure, with an assigned risk exposure value R and, in the context of risk-based testing, assigned tests. The risk exposure R is a value measured at least on an ordinal scale that determines the risk, i.e., an uncertain event or condition that, if it occurs, has a negative effect on the system. The binary operator $\circ$ that connects P and I is typically the multiplication of two numbers or the cross product of two numbers or arbitrary ordered elements (but not restricted to these operations). This standard risk model can be generalized by reducing it to a specific value or refined by considering several (weighted) criteria for P and I. Following for instance the Factor-Criteria-Metrics model [12], these criteria are determined by metrics. Each metric has an assigned scale and calculation procedure which can be performed manually or automatically. In addition, the risk model has a computation model to aggregate metrics to criteria values and further to factor values. Finally, the risk model contains risk levels to which risk exposure values are assigned. A risk level [10] indicates the criticality of risk items and serves the purpose to compare risk items.

Risk models for testing purposes are discussed in several publications. Bach [4] presents a pragmatic approach to risk-based testing grounded on a heuristic software risk analysis. Bach distinguishes inside-out risk analysis starting with details about a situation and identifying associated risk, and outside-in risk analysis starting with a set of potential risks and matching them to the details of the situation. Redmill [6,13] provides a thorough discussion of risk-based testing [13] as well as a proposal for practical application suggesting a single factor risk assessment, either on probability or on impact, or a two-factor risk assessment, in which probability and impact are combined [6]. Felderer et al. [8] propose a risk assessment model defined on the basis of an industrial project. In this model, the risk coefficient is assigned to features and determined by impact, probability, and time factors. Each factor is determined by criteria which are defined by metrics. Metrics are determined manually, semi-automated, or automated. Finally, the practical risk-based testing approach (PRISMA) [7] distinguishes business and technical risks determined by weighted criteria to calculate the overall risk of the risk items.

2.2 Risk-based test process

Risk-based testing involves the identification of product risks, i.e., risks related to specific product properties and qualities, and the use of risk levels to guide the test process [10]. It can only deliver its full potential if a test process is in place and if it is considered appropriately [5]. Risk-based testing is a testing-based approach to risk management. A risk-based test process, therefore, combines a test and risk management process.

A test process contains the core activities test planning and control, test analysis and design, test implementation and execution and test evaluation and reporting [10]. 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 testing activities. In test control, the actual progress is compared against the plan that

often results in concrete measures. During the test analysis and design phase, the general testing objectives defined in the test plan are transformed into tangible test conditions and test cases. Test implementation 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.

A risk management process contains the core activities risk identification, risk analysis, risk treatment, and risk monitoring [14]. In the risk identification phase, the risk items are identified. In the risk analysis phase, the probability and impact of risk items and, hence, the risk exposure is estimated. On the basis of risk exposure values, risk items may be prioritized and assigned to risk levels. This results in a risk classification. In the risk treatment phase, the actions for obtaining a satisfactory situation are determined and implemented. In the risk monitoring phase, the risks are tracked over time and their status is reported. In addition, the effect of the implemented actions is determined. The activities risk identification and risk analysis are often collectively referred to as risk assessment, while the activities risk treatment and risk monitoring are referred to as risk control.

Every available risk-based test process integrates testing and risk management activities. In the following, we explain this integration by discussing three published risk-based testing processes, i.e., [5,8], and [7].

Amland [5] defines a risk-based testing approach that is based on the generic risk management process of Karolak [15] comprising the following steps and the corresponding risk management activities: planning (risk identification and risk strategy), identification of risk indicators (part of risk assessment), identification of the cost of a failure (part of risk assessment), identification of critical elements (part of risk assessment), test execution (risk mitigation), and estimation of completion (risk reporting and risk prediction). Amland was one of the first in introducing a systematic risk-based testing approach.

Felderer et al. [8] take the standard test process of the International Software Testing Qualifications Board (ISTQB) [10] as a starting point and integrate risk identification and risk analysis into it. In addition, the activities of the standard test process are extended by risk-specific elements: (1) in the test planning and control phase, the test plan is extended by a risk-based test strategy, (2) in the test analysis and design phase, a concrete test schedule is defined based on the test plan and a concrete risk classification, (3) in the test implementation and execution phase, the execution of the test cases is determined by their priority (determined by the assigned risk exposure value) and the resource limitations, and finally (4) in the test evaluation and reporting phase, the alignment of risks and test results enables residual risk estimations and supports decisions.

The practical risk-based testing approach (PRISMA) [7] defines a process consisting of concrete activities, i.e., initiating, planning, kickoff meeting, extended risk identification, individual preparation, processing individual scores, consensus meeting, and define differentiated risk-based testing approach. The activities are defined in a very concrete way with detailed instructions. Thus, the PRISMA approach is highly specific and not very adaptable. In addition to generic risk-based testing approaches, several model-driven approaches to risk-based testing have been introduced [1,16–18]. As model-driven testing is not applied in the industrial cases discussed in this article, we do not consider these approaches further.

2.3 Introduction of risk-based testing

As a precondition to introduce risk-based testing, a basic test process consisting of the phases test planning and control, test analysis and design, test implementation and execution as well as test evaluation and reporting as described before is required. In practice, the introduction of testing and risk management activities is typically performed stepwise resulting in several incremental stages of integration with growing maturity. In [19], the stages (1) initial risk-based testing, (2) risk-based test results evaluation, (3) risk-based test planning, and (4) optimization of risk-based testing are proposed. Initial risk-based testing comprises the identification of risk items, the definition of a risk analysis procedure, and the design and execution of test cases based on risks in an informal way, i.e., not based on a risk-based test plan, to control the design and execution of test cases. The assigned risk values can, for instance, be used to distribute resources for test design or to prioritize test cases for test execution. In the risk-based test results evaluation stage, test evaluation and the underlying reporting take advantage of the linkage between test and risk information, which provides additional release decision support. Risk-based test results evaluation is valuable to control the test and release quality. Risk-based test planning formally takes risks into account in the test plan, e.g., for selecting appropriate test design techniques or exit criteria. Thus risk-based test planning formalizes the risk-based testing activities of the lower stages. Finally, optimization of risk-based testing continuously evaluates and improves the risk-based test process.

2.4 Related work

Although several risk-based testing approaches have been proposed [4–8], only a few empirical studies on risk-based testing are available.

Yoon and Choi [20] 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. Using this data, the authors apply their test case prioritization approach, which employs risk exposure values as assessed by experts, and compare the results with those of applying Chen's approach [16], which generates risk exposures of individual test cases. Differing from our approach, Yoon and Choi only evaluate test case prioritization based on their risk analysis approach but do not consider the overall risk-based test process in different industrial cases as we do.

Souza et al. [21] 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 [22], which consists of the phases risk identification, risk analysis, test planning, test design, test execution as well as test evaluation and risk control. Differing from our multiple case study, their case study is not performed in an industrial context and does not investigate the overall risk-based test process but only the efficiency and effectiveness of defect detection with risk-based testing.

Finally, Felderer and Ramler [19] 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. Differing from this article, the authors consider only one specific aspect, i.e., the introduction of risk-based testing, in one industrial case, but not across multiple cases and research questions on how risks are defined, assessed, and applied to support testing activities.

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 (Sects. 3.3 to 3.5) to answer them. The research design follows the guidelines for conducting and reporting case study research proposed by Runeson and Höst [2].

3.1 Research questions

For practitioners from industry it is a fundamental necessity to align testing activities with business value and risk. So for them seeking applicable guidelines and best practice information on risk-based testing, it often comes to a surprise that researchers have collected only a small body of empirical evidence on risk-based approaches so far. To support the need for further empirical evidence of practitioners and researchers, the following four research questions are investigated to explore the state of risk-based testing in industry. These questions have been derived from the overall research objective to explore and describe how risk is defined, assessed, and applied to support and improve testing activities in industrial projects.

- (RQ1) What is the notion of risk in software testing? This research question addresses the commonalities and differences of the risk definitions used in practice.
- (RQ2) 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.
- (RQ3) 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.
- (RQ4) 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.

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

3.2 Case selection

Three cases have been selected for investigation. They were selected from the wider set of industrial collaboration projects of the authors. The motivation for selecting these particular cases comes from their different scope and backgrounds: The focus of Case A is primarily on a project for a single software system, Case B is rooted in software product development with a rich history, and Case C has been dealing with a software process improvement initiative to create several product variants (see Fig. 1).

The characteristics of the different cases regarding their scope, engineering domain, previoushistory, and organizational culture in terms of software development approaches and practices are summarized in Table 1. A detailed description and analysis of each case is presented in Sect. 4.

3.3 Data collection procedure

The case study is based on data from the three selected cases described in Sect. 4.

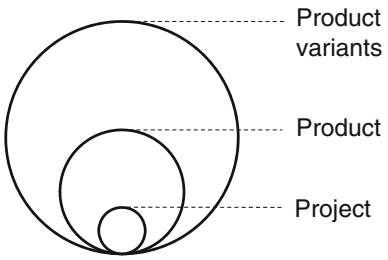

Fig. 1 Scope of the different cases

For answering the research questions, the study relied on multiple sources of evidence. First, project, product, and process documents such as project plans, product documentation, process descriptions, test plans, requirements specifications, design specifications, test specifications, protocols, and risk assessment documentations were analyzed. Second, the data available in project management tools, and repositories was analyzed. Finally, semi-structured interviews were conducted, based on a list of open interview questions derived from the research questions. Interviewed project personnel were test managers, project managers, and process managers. The collected data is further explained in Sect. 4.

3.4 Analysis procedure

The main analysis across the three cases was conducted with qualitative methods. Results were derived by analyzing documents, tools and interview protocols keeping a clear chain of evidence [3], i.e., a reader should be able to follow the derivation of results and conclusions from the collected data. The analysis was started in parallel with the collection of data from documents and tools. On the basis of first insights, additional data was extracted from documents and tools, and interviews were conducted to confirm, refine, or refute findings. Qualitative analysis was combined with a limited quantitative analysis of the number of failures observed in Case A. The analysis was performed by two researchers. The preliminary results from each individual researcher were merged into a common analysis result in a second step. The analysis results are presented in Sect. 5.

3.5 Validity procedure

Based on the guidelines of Runeson and Höst [2], 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 and tools was triangulated by interviews and the findings were discussed 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 three studied cases selected 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 describes a software development project completed by a company providing IT services. In the studied project, a Web-based application has been developed, which is an integral part of a large information system. The project was one out of a series of related development projects conducted over the last years to establish and extend the overall information system. The goal of the project investigated in our study

Table 1 Characteristics of the three cases

	Case A	Case B	Case C
Scope	*Project* Case relates to a single development project out of a series of related projects	*Product* Case relates to product development over several consecutive versions	*Product variants* Case relates to the software process for developing and testing several product variants
Domain	IT services	Electrical engineering	Telecommunication
History	Less than one year for the analyzed project; Several years of project development	Product development history of more than 15 years	Established process for development and integration; Risk-based testing recently introduced
Culture	Structured, iterative development process; Specified requirements and acceptance criteria	Established overall engineering process; Agile organization of development projects	Development and testing follows V-model; Traceability of requirements, features and components

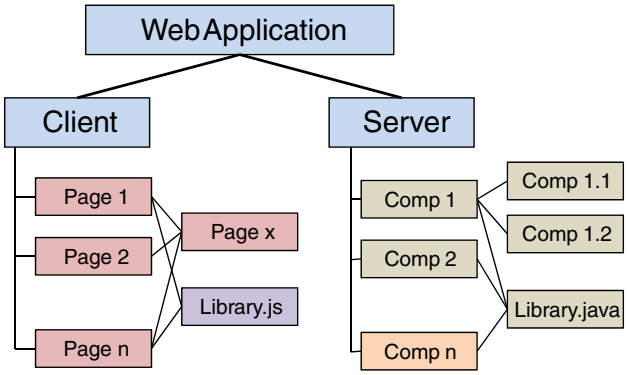

Fig. 2 Application structure and technologies

was the integration of third-party information services and to provide a Web-based user interface to access these services. The project followed an established process and applied a set of best practices that had shown their usefulness in the previous projects.

In total, ten people were involved in the project in different roles, such as developer, tester, architect, or project manager. The overall duration of the project was nine months. This timespan was structured in two iterations, each further divided in a development and a stabilization phase. In both iterations, a defined set of features was implemented in a series of short, time-boxed sprints. The feature-freeze milestone marked the completion of the implementation and, in consequence, the transition from the development phase to the stabilization phase.

In the development phase, the specified requirements assigned to the iteration were implemented. The requirements were organized according to the use cases of the application, which were also reflected in the page structure of the Web-based user interface. The requirements specification also detailed the graphical layout of the Web pages. The implementation involved a variety of different technologies and programming languages spanning from Java and Java Server Pages on the server-side to JavaScript running in the browser on the client-side (Fig. 2).

Testing activities in the development phase concentrated on unit and integration tests, which were the responsibility of each individual developer. In parallel to the development activities, system tests were derived from the requirements and specified in a test management tool. In the development phase, system testing was performed only informally, to provide early feedback. Thorough system and regression testing based on the specified test cases was the central activity in the stabilization phase, together with the fixing of the detected issues. At the end of the stabilization phase, the application containing the new features was released and handed over to a separate service and maintenance organization.

The everyday work in Case A was characterized by the joint endeavor to successfully complete the project in time and on budget. The project management maintained a plan describing how to meet this goal and a list of risks, which threatened reaching the goal. Even though testing was considered a time-consuming and costly activity, it was also appreciated as one of the most important measure for risk mitigation by making sure that the developed application fulfilled the specified functional and quality requirements.

4.2 Case B

The company providing Case B produces measurement and diagnostic equipment for the electrical power industry. Besides various measurement devices and tools, the product spectrum includes a large software suite with a comprehensive set of features for electrical hardware diagnostics, e.g., for checking protective relays.

The software suite controls the different measurement devices, executes measurement and diagnostic programs, manages and tracks results, and creates various kinds of statistics and reports. The software runs on standard PC hardware and is compatible with all recent versions of Microsoft Windows. Furthermore, with customers in over 140 countries, the software is fully internationalized and supports 14 languages.

With a development history of over 15 years, the software system has evolved to a total size of more than 2.5 million lines of code. A range of different technologies and programming languages (C/C++, C#, Visual Basic, FORTRAN as well as popular scripting languages) have been used to implement the system. The software system's architecture distinguishes a set of 25 dedicated modules that realize the measurement and diagnostic functionality. They share several components for handling data persistence, report generation, etc., as well as a hardware abstraction layer including the device drivers to communicate with the different tools and measurement devices.

The complexity of the software system has increased from version to version. The major complexity drivers were (1) additional modules introducing further inter-module dependencies, (2) extensive support for internationalization including different languages and character sets, (3) backward compatibility with previous versions for importing measurement data in versions containing new or extended functionality, (4) transparent support for different revisions of hardware devices, and (5) compatibility with a wide range of operating systems, service packs and updates from Microsoft Windows 95 to Windows 8 on 32-bit and 64-bit systems.

Software development follows an iterative, incremental process. The product manager outlines and specifies new functionality such as a new software module, support for a new hardware device, etc., based on customer requests and technological advancements. The Scrum development approach is used to drive the development of new function-

ality as increments to the existing software system. On a regular basis, several increments are combined, integrated, and released as major versions of the software product. Maintenance and support are ongoing activities that are conducted in parallel to development and result in service releases, which are also provided for supported previous versions.

Testers are involved from the very beginning, when new functionality is specified. Thus, they can start to design test cases in parallel to development. Whenever an increment is completed, the new functionality is tested on the basis of the corresponding test cases, augmented with some additional exploratory testing. The combination of all increments is tested together with the existing functionality in a full-system regression test before a new version is released. The basis for this test is the whole set of existing test cases. They are maintained with the help of the test management tool SilkCentral [23]. Critical functionality is again cross-checked by additional exploratory testing.

The changes made for service releases concern only a fraction of the overall system. For service releases, a change-based regression approach is considered sufficient in most cases. Thus, the changes are tested in combination with the affected modules. Instead of a full-system regression test before releasing, only a set of standard tests is used to cover the most important use cases.

The majority of the tests for the software system are executed manually. The highly dynamic behavior of the diagnostic modules including the user interfaces with limited testability call for a primarily manual testing approach. Automated testing is mainly used for the hardware abstraction layer. These tests can be easily automated but cover only a small part of the overall system. Running and maintaining the available automated test suites is, nonetheless, a major cost factor. Full-test automation is therefore not an option, even when the technical issues of UI test automation could be resolved.

The required time for full-system regression testing as well as the involved effort and costs restrict the frequent release of new versions. For service releases, the introduction of change-based regression testing in combination with exploratory testing and a set of standard tests has considerably reduced the overall effort. Therefore a change-based approach is also considered for major releases in future.

4.3 Case C

The Case C company is a system integrator of telecommunication solutions in the fields public transportation, railways as well as mobile network operators. The portfolio of the company comprises software-based solutions for IP-based telecommunication infrastructure that supports all services and traffic types. Among others, solutions for billing, network traffic analysis, call management, or asset management are provided. For specific customers or markets, these solutions are implemented as product variants. The components implementing these functionalities are integrated and adapted to a broad range of different system contexts and complex environments where different architectures, technologies, and programming languages (C/C++, Java, and the proprietary programming language Protel) are used.

The company follows a structured development and test process on the basis of a clearly defined generic system and test model shown in Fig. 4. In this model, -so called features are the central concept to plan and control implementation and testing. A feature has a concise and complete description of its functionality, along with non-functional aspects like performance or security. Features are on the one hand assigned to requirements and on the other hand to components. A requirement describes a certain functional or non-functional property of the system and is implemented by a set of features. A component is an installable artifact that provides the functionality of several features. Components are defined hierarchically in a tree. The root component represents the system and the leaves are units. As features are the tested artifacts, test cases are assigned to them. Differing from components, testable objects are executable units composed of one or more component and a test environment. Testable objects are assigned to the system or a component and have an attached test plan. A test plan contains test cases grouped either by features or components. Each test case contains a description, test steps, and expected results. Development and testing follow the V-model. First, a customer solution manager collects the requirements in a user requirements specification. Then, the features are defined and the system architecture is derived by a technical solution manager. The features are assigned to requirements in the technical requirements specification. The system design is then further refined to concrete components with assigned units, which are implemented and tested by a developer. As soon as feature definitions are available, test planning is started. First, testable objects and a test plan, which is based on formerly determined requirements acceptance criteria, are defined. It contains test cases grouped either by features or components and has a test-end criterion. In the test design phase, executable test cases are defined by testers according to the test plan. Test cases are also adopted from existing components or features. New or changed test cases are reviewed and corrected if necessary. After the respective testable object (including its test environment) is available, the test cases are executed. Each test run contains a test result for each of its executed test cases. Depending on the test results, a problem ticket is created. As soon as the test-end criterion is reached, a test report is provided.

Testing and development are managed with the project management tool in-Step [24]. Most test cases are executed manually. Test automation support is available only

for specific network protocols and analysis reports. A parallel change management process organizes the activities concerning the handling of requests for change (RfC). It consists of the three linearly connected activities (1) collecting and inspecting RfC, (2) approving or declining RfC and (3) performing change and periodic RfC monitoring. Change management comprises also testing activities, i.e., executing test cases for the affected feature or component.

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 is described and discussed in context of each case.

Following conventions are applied in this section: Text translated from recorded interviews or paraphrasing statements of case participants is shown in italics; comments and interpretations added by the authors are shown in square brackets.

5.1 What is the notion of risk in software testing? (RQ1)

To answer the first research question, we investigated how "risk" is understood in the different cases. The findings are:

- *F-01* The concept of risk depends on context and scope, but there is an agreement that product risks are relevant for testing, as testing is understood as measure (i.e., a protective action) to reduce the exposure of the product to risks of this type.
- *F-02* There is a general agreement that the risk as understood in the different cases follows the common definition of risk as $R = P \circ I$, where P is its probability and I its impact. However, the definition and relation of P and I may remain informal and implicit. Both factors are usually based on (subjective) estimates made in a project or product development context.
- *F-03* The degree of formality of risk as a concept depends on the application scope. It increases when the scope widens from project to product and, furthermore, to process.

In all cases, risk management is an established activity at the project or organizational level. At this level, project as well as product risks are addressed. In software testing, the focus is on risk related to specific product properties and qualities. In all cases, we found that testing activities are understood as means to mitigate the identified product risks.

Case A Testing is used to detect quality problems before they "escape" to the field. In this way, testing reduces the involved risk. Testing is not the only measure to reduce risks. It is highly resource-intensive. So, also other quality assurance measures are applied whenever possible, for example, reviews are used for specifications and documentation. However, when it comes to executable software, testing is usually the only practicable measure available.

Even though the observed practices in testing are to some degree based on risk, Case A and Case B do not rely on a formal definition of risk. The interviewed participants agree with the definition of risk from ISTQB's standard glossary of terms used in software testing, which defines risk as "a factor that could result in future negative consequences; usually expressed as impact and likelihood" [10]. However, this high-level definition is typically operationalized in specific contexts, usually by enumerating concrete examples for actual product risks.

Examples for risks are correctness of the main usage scenarios, correct display of content (e.g., images), robustness in case of connection problems, and integration of potentially unreliable third-party services. The main source for relevant risks is the risk management conducted at project level. Risk management maintains a list of evaluated risks. The test manager is involved in identifying and evaluating these risks. Evaluation includes estimating probability and impact.

In *Case B*, test management does not rely on a formal risk management procedure. The understanding of risk is mainly derived from a thorough understanding of the critical characteristics of the product, and a profound expertise in the application domain. This understanding has been established over the many years of product development and testing.

Quality is related to the trust of the user in the software and its results. The highest risks are problems that threaten the reliability and accuracy of measurement results, their correct visualization, or their correct interpretation. Such problems are even more critical than crashes, even when they may lead to the loss of unsaved measurement data.

This particular understanding of risk in Case B is determined by what users and customers value most. It mainly relates to the impact, the part of the risk expressing the consequences if a problem occurs. Over time, it became "common knowledge" within the product team, as user and customer views have been found to remain stable for years.

Changes are considered the main source of problems. In contrast to the stable impact, the risk probability increases with every change made in a release. Various types of changes are considered relevant for testing.

Changes of existing functionality may lead to incorrect behavior of this functionality, adding new functionality to the software product may lead to unwanted side effects on existing unchanged functionality, and changes in the environment such as the underlying operating system may have a negative impact on the product's stability.

In *Case C*, at process level, a generic risk definition is used. Risk is defined on the basis of the Factor-Criteria-Metrics model of McCall [12]. The definition considers the factors probability P and impact I as well as the additional factor time T. Probability reflects the technical view, impact the business view, and time the system evolution. The factor values are determined by weighted criteria.

The probability factor is composed of the weighted technical criteria code complexity, data complexity, functional complexity, visibility, and third-party software. The impact factor is composed of the user and business-oriented weighted criteria usage, availability, importance, and performance. The time factor is composed of the criteria bug tracking, change history, new technologies, and project progress. Each factor is determined by a specific metric. Metrics can be determined manually or automatically.

The definition of risk in Case C is more formal than the definition in the Cases A and B. Probability, impact and time are explicitly defined on the basis of several criteria. For each criterion, metrics are defined to determine the value in an objective way. The metrics can be calculated manually by a suitable stakeholder or even automatically. For instance, the importance is measured manually on a five-point Likert scale and the code complexity is measured automatically by the McCabe complexity [25].

Due to the traceability between requirements, components, and units via features, the probability criteria are measured for units, the impact factors for requirements, and the weights are assigned to components (from which only a few exist). Time criteria are directly assigned to features, for which the risk exposure values are calculated (see Fig. 4).

The risk exposure value of a feature can be calculated via the formula, $R = (P \cdot T) \times I$, where P denotes the aggregated probability factor, I the aggregated impact factor, and T the aggregated time factor which reduces the value of P over time. Based on the probability factor (possibly reduced by the time factor) and the impact value, the risk level low, medium. or high is assigned to a feature. This risk information is used to support test planning, design, and execution.

The resulting risk model is described in detail as part of the answer of the next research questions.

5.2 How do risks support the activities of the software test process? (RQ2)

To answer this research question, we investigated how the risks are used in software testing in the different cases. The findings are:

- *F-04* Risk is taken into consideration in virtually all testing activities, even when testing is not explicitly following 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.
- *F-05* Although traceability between test cases and risk items would enable justified release quality statements as well as the estimation of residual risks on principle, risk information from testing has not been found to play an explicit role in making release decisions.
- *F-06* Risk information is used in testing in two ways: (1) As a suggestion to extend the scope of testing towards "risky" areas where critical problems can be found, and (2) as a guideline to optimally adjust the focus of testing to "risky" areas where most critical problems are located.
- *F-07* Risk-based testing is not understood as an approach to reduce the amount of testing, the overall test effort, or the established test budgets. On the contrary, additional test cases are usually designed as a consequence of addressing all risks, and additional test cases may be included in a regression test suite to adequately cover all risks.

The *Case A* organization defines a standard process for software development projects including activities such as risk management and test management. The analyzed project follows this process. While the process describes the activities in an abstract way, the project team develops a concrete, innovative solution for a specific problem and a specific context. This project differs from previous ones in many ways and, thus, has to cope with new, unique risks. Risk management and the related testing activities are therefore driven by the goal to successfully implement the specific solution fulfilling all (functional and quality) requirements.

Test cases are mainly derived from specified requirements. The tree structure used to organize the test cases in the test management tool resembles the structure of the requirements specification. The requirements are specified as use cases (organized according to the functional structure of the Web application). The test cases derived from these use cases are located in the corresponding sections in the test management system.

The list of risks maintained by project management is used as an additional source inspiring the design of further test cases. High-level risks are represented as separate sections in the test structure. Test cases mitigating these risks are placed in the corresponding sections. For example, the section "Connection Tests" contains test cases dedicated to the risk of possible connection problems. Occasionally, risks affect one or more specific requirements. Thus, additional test cases addressing these risks may also enhance the test sets related to these requirements.

Specified test cases are classified according to their priority. The prioritization schema reflects a risk-based testing

approach. However, the concrete priorities of the test cases are defined by a simple procedure that does not take specific risks into account.

The priority is defined by the custom attribute "Priority" in the test management tool. Four levels of priority are distinguished; they range from "high" to "low". The priorities depend on the corresponding requirements. The rule is that test cases for the main scenarios of a use case are given high priorities, test cases for alternative scenarios are assigned medium priorities and, finally, test cases related to exceptional or very specific scenarios are of low priority.

In *Case B*, the main effort in software testing is caused by extensive regression test phases, which last up to three months. A huge set of regression test cases has been established that captures the context-specific essence of risk, distilled from a long history of testing the software product. Risk plays a major role in selecting test cases for regression testing for a specific release. As described in answering RQ1, the two risk factors impact and probability are clearly distinguished in Case B. Impact expresses the knowledge about the user and customer view, and probability represents the risk due to changes. The two factors affect the testing activities in different ways. The factor impact is used to determine the static "initial" priority of a regression test case. The factor probability incorporates the volatile aspects of an individual change relevant for selecting regression test cases.

A total of several thousand test cases have evolved over the years of ongoing development and testing. These test cases are stored in the test management system, where they are organized with respect to the different modules [i.e., the functional entities] of the software product. Test case design is based on the specified requirements and the testers' expert knowledge. At design time, test cases are assigned a priority corresponding to the tested functionality, ranging from "must" to "nice-to-have". Since the user and customer view on what is important remains the same over the releases, the test case priorities are rarely readjusted.

When compiling an execution list (i.e., regression test suite) for a new release, all test cases with the priority "must" are included. Including these "must" test cases is mandatory. The test cases with priority "must" cover only the critical parts of a module. Depending on a specific change, several other parts may also be affected and need to be tested. The testers add further test cases to the execution list based on the description of the change, a discussion with the developers responsible for the change, and their experience from testing software system for many years.

The risk probability represents the likelihood of an unintended side effect caused by the change. Additional test cases (with lower initial/static priority) are assigned to the regression test suite depending on this risk probability. The probability is estimated by the tester often implicitly and on the fly, when compiling the regression test suite. "Coverage of the change" is among the most important criteria for estimating the probability. Custom tool support has been introduced for collecting and combining test coverage and change data to aid testers in selecting a minimal set of regression tests.

The described procedures relate to the activities in the established regression test process. This process has evolved over many releases and is suitable to cover the "commonly known risks" in Case B. However, a specific release may also include specific risks that need to be addressed in context of this particular release. The use of additional exploratory testing allows testers to "learn" about these specific risks and to address them individually. As a consequence for future releases, the tester may decide to add new test cases derived from exploratory testing to set of regression tests specified in the test management system.

In *Case C*, risks are explicitly considered in the test planning, test design, and test execution phase. In the test planning phase, first testable objects, which are executable units composed of one or more components and a test environment, are defined. Then a test plan is created on the basis of formally defined requirements acceptance criteria, features with attached risk exposure values and components.

The test plan may have a test-end criterion which considers the risk levels of features, e.g., all features with high risk have to be tested, features with medium risk are optional candidates to be tested to reach the required test coverage, and features with low risk are only tested if all others have been tested and resources are available.

The definition of the test-end criterion in Case C is not optimizing, i.e., it does not minimize the test effort utilizing risk information, but only adjusts the focus of testing according to the risk level.

In the test design phase, executable test cases are defined by testers according to the test plan and get assigned the risk exposure value of the feature they are designed for. Already existing test cases, which are applicable are selected from similar previous components or features. New or changed test cases are reviewed and corrected if necessary. After the respective testable object (including its test environment) is available for the test, the test cases are executed ordered by their risk level and risk exposure values (inherited from the assigned features). Each test run contains a test result for each of its executed test cases. Depending on the test results, a problem ticket is created. As soon as the, test-end criterion is reached, a test report is created. But the test report itself does not explicitly take risk information into account.

The process manager of Case C originally intended to take risk information explicitly into account in test reports. This would support reliable release quality statements as well as the estimation of residual risks based on the risk reduction with every test cycle (until a risk level is reached where it is acceptable to release the software application). But it turned

out that the customer solution manager who decides on the release of an application does not require risk information for his decision but only takes the importance of features for the customer as well as the severity of open defects of important feature implementations into account.

5.3 How are risks organized from a technical perspective? (RQ3)

This research question relates to the technical aspects underlying the elicitation, management, and usage of risk information. To answer this question, we investigated (1) how risk is modeled and (2) how this model is implemented in the tool infrastructure. The findings are:

- *F-08* Only high-level risks at the level of project management are modeled as individual entities. In testing, risks are calculated from risk information associated to testable entities for identifying critical areas of the software system. The testing effort is concentrated on these areas.
- *F-09* Testing relies on the organizations' standard tool infrastructure. This infrastructure usually includes conventional test management tools for managing specified test cases and test execution results. Tools for risk management, commonly used at the level of project management, are not applied in software testing.
- *F-10* Risk-based testing is implemented with the help of the available tools or as extension to the existing tool infrastructure. Extensions are developed individually, when the applied standard tools do not provide the necessary support for risk-based testing.
- *F-11* Tool support specific for risk-based testing is mainly considered as an aid to automate the laborious tasks of collecting and analyzing measurement data. This data is used to derive risk information or as a basis for risk estimation.

In *Case A*, a diverse list of risks is maintained in Excel by the project manager. Risks are mostly coarse grained. They include a unique ID, a brief description, and a criticality score based on the factors probability and impact. The estimates are subjective ratings by experts using a generic five-level scale. The criticality score is determined by multiplying the two estimates. If a relation of risks and requirements, non-functional aspects or parts of the software system exists, it is indicated in the risk description. Traceability remains informal and is not managed at the level of test cases. Test cases derived from risks are usually organized in separate sections ("packages") of the hierarchically test structure in the test management tool. In that way, test cases can be traced back to the risks from which they originate. It remains a subjective decision whether the criticality scores of the risks are taken into consideration when defining the priorities of the test cases. Using risks in testing is a one-way information flow. Risks are not specifically considered in evaluating and reporting test results.

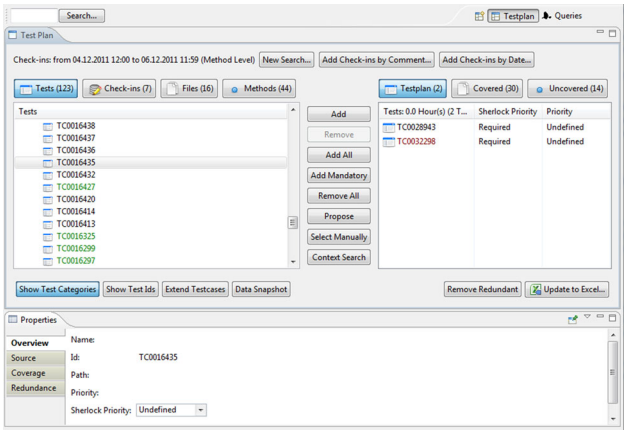

Fig. 3 Tool support for selecting regression test cases

In *Case B*, a lot of tacit knowledge is involved in risk-based testing. It manifests itself mainly in two points of the test process. First, in the priorities assigned to test cases in the test management system SilkCentral and, second, in the change-based approach used to select regression tests.

For the second point, additional tool support has been developed as an extension to the test management system (Fig. 3). The tool collects coverage information from test execution to compute a coverage footprint for every test case. Coverage information is recorded in terms of covered files and classes, covered methods as well as covered lines. In addition, dependencies between files and methods are analyzed. The coverage footprint is extended by including dependent files and methods. Change information is retrieved from Microsoft Team Foundation Server, which records the changed lines of all affected files for every enhancement or bug fix. Tests that cover changed code or dependent code are selected for regression testing. The dynamically computed regression test sets are suggestions offered to the testers. The testers are free to incorporate, reduce, or extend the suggested sets when compiling the actual regression test suite for a release.

Thus, in Case B, risks are not specified as individual entities. Risk information is encoded in the form of attributes of test cases or it is provided as additional information linking test cases to areas of the software system.

In *Case C*, risk-based testing is applied to test features. Therefore, risks are also assigned to features which are the risk items in Case C. Explicit high-level risks are not defined. Each feature describes one specific capability of a system or component. Thus, risk-based testing is applied on the system and component level. Figure 4 shows the core artifacts, their

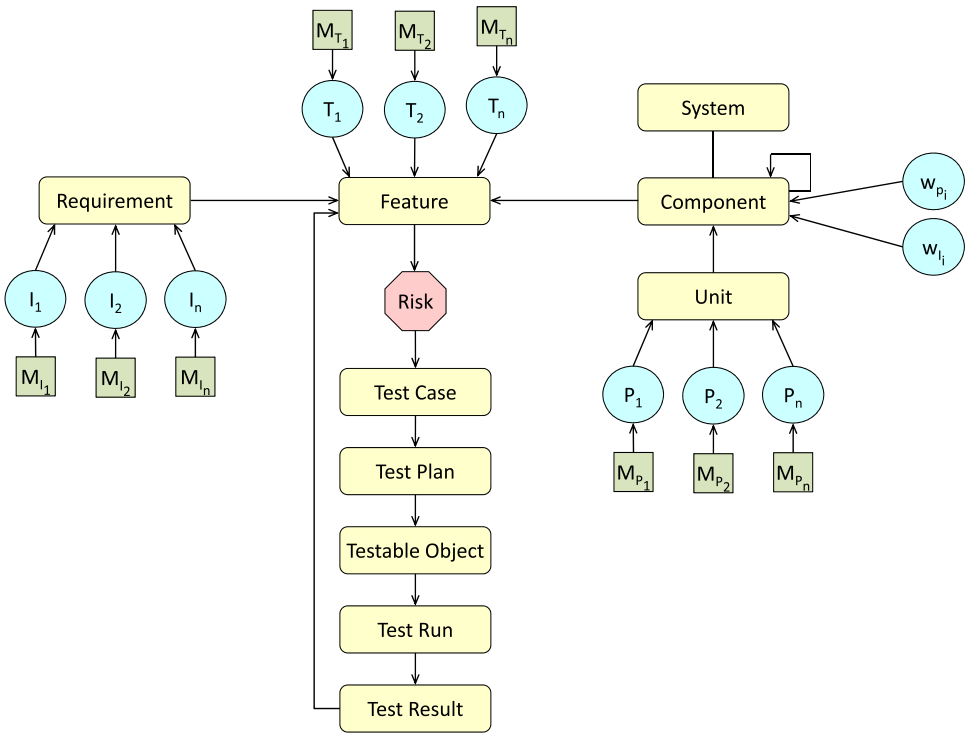

Fig. 4 System, test, and assessment model of Case C

relationship and the risk assessment model of the test process applied in Case C.

The distributed computation of risk exposure values on the basis of factors assigned to features, units, and requirements is illustrated in Fig. 4. As already discussed for RQ1 in Sect. 5.1, the computation is based on the Factor-Criteria-Metrics model. Probability criteria (P_i) are assigned to units, impact criteria (I_i) to requirements, time criteria (T_i) to features and weights (w_i) to components. Each criterion is measured by an automatically or manually determined metrics. Automatically determined metrics can be measured by specific tools like static analysis tools for source code quality management. For manual determination of metrics the distributed model has the advantage that various stakeholders with specific knowledge can be integrated into the risk analysis process. The probability criteria are determined by software architects, the impact criteria by customer solution providers, i.e., a product and requirements engineer, and the time criteria as well as the weights by test managers.

The risk-based test process of Case C with all its artifacts and process steps is supported in the project management tool in-Step [24]. In Case C, in-Step is the established tool to implement the development and test process. For risk-based testing it has been extended to support risk analysis. Figure 5

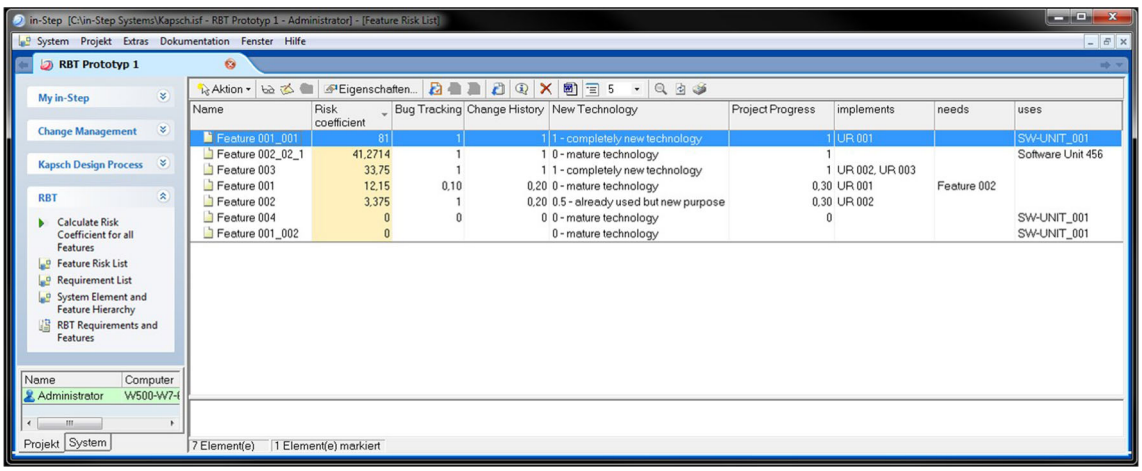

Fig. 5 Integration of risk assessment in project management tool

shows a screenshot of the specifically developed risk analysis view of in-Step where the measures for criteria are entered and processed to calculate risk exposure values. In addition, for the automatic measurement of source code metrics the source code quality management tool Sonar [26] is applied.

5.4 What are the benefits of a risk-based testing approach? (RQ4)

To answer this research question, we investigated the achieved or expected positive effects of following risk-based approaches in software testing. The findings are:

- *F-12* 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.
- *F-13* In addition to detecting more defects, risk information is used to target the most critical defects from the very beginning. By setting priorities based on risk information, once again, risk-based testing helps to make testing more effective. It increases the chance to find critical defects in the early iterations of testing and, in consequence, it also reduces the overall costs and time required for stabilization.
- *F-14* Test managers and testers consider the incorporation of risk as beneficial for informed decision-making. The insights derived from the risk information are used to triangulate and refine initial decisions. The analyzed cases include examples where risk information supports decisions on resource allocation, defining regression test suites, identifying candidates for test automation, as well as decisions on when to end testing.

In *Case A*, testing is primarily oriented towards the specified requirements. The requirements are the reference when defining the goal of the project and when ultimately deciding whether or not this goal has been met. Testing measures the progress in terms of implemented requirements and provides feedback about remaining gaps as well as incomplete or defective implementations. By deriving additional test cases from risks, the scope of testing is increased towards potential defects that threaten reaching the project's goal.

Testing the requirements is not enough. Critical defects may be missed. Thus, additional tests are designed (with the aim to increase the scope of testing) to find all relevant defects (including those not obvious from analyzing the requirements specification). Typical examples are defects specific for the design and implementation of the solution and defects related to the applied technologies. Furthermore, the requirements are specified in a positive form, describing only what should happen. Testing also has to make sure that undesired, negative situations, that should not happen, will not happen.

Risk-based testing addresses these "negative situations". The test cases derived from risks go beyond techniques for testing exceptions and failure scenarios of use cases and specified requirements. Risks are related to complementary sources of failures, and the observed risk-based testing approach is related to techniques like error guessing and using defect taxonomies [27].

Furthermore, a risk-based approach includes giving priority to important requirements and high-risk system parts. Therefore, testing will be better prepared to handle the inevitable time and budget constraints. Prioritization in risk-based testing is also assumed to provide early detection of critical defects. The expected benefit is a reduced overall duration of the stabilization phase.

For *Case B*, the advantage of risk-based testing is the optimal usage of the available resources for regression testing. The consideration of impact (user and customer perception) and probability (recently changed parts) in selecting regression tests allows concentrating testing activities on those parts of the software system that have a high likelihood of being defective without jeopardizing quality due to missing critical defects in other parts. A full regression test can, therefore, be avoided when releasing bug fix and maintenance versions. As a benefit, bug fixes can be provided promptly and small updates can be released more frequently than the major versions, which usually include regression testing for several months.

Quality involves a timely response to bug reports and feature requests. The ability to quickly fix a defect is essential for problems found by customers in the field. Time to market is generally a critical factor in product development. The risk-based approach is also considered for testing major versions. When focusing on changes first, the bugs are found earlier and the often long and time-consuming fix and re-test cycles can be reduced. (For every fix of a defect, the previously executed tests have to be run again.) So on the long run, the product can be released earlier.

The availability risk information is also considered a benefit. The implementation of the risk-based testing approach provided access to a wealth of change impact and coverage data for the testers. Previously, the impact of changes was often a subjective estimate based on expert opinion and tacit knowledge. With the tool support for change-based testing, also testers less familiar with implementation details of a particular part of the software system are able to make sound decisions when selecting regression tests. Furthermore, testers use this information also for decision on where to invest in test automation.

In *Case C*, risk exposure values and risk levels are so far only used to prioritize and select test cases for execution and as test-end criterion in a basic risk-based test plan. But risk

information is not used to control test design itself, e.g., by varying the test depth dependent on the risk level. In the test design phase, executable test cases are defined by testers according to the test plan and get assigned the risk exposure value of the feature they are designed for. Already existing test cases which are applicable are selected from similar previous components or features. After the respective testable object is available for the test, the test cases are executed ordered by their risk level and risk exposure values.

The achieved benefits of this risk-based testing approach in Case C are mainly decision support on resource allocation. The time resources for testing are limited as solutions have to be provided at fixed dates. Therefore, test cases have to be prioritized for execution based on their risk level to mitigate highest risks in the limited test window. A test-end criterion which considers the risk levels of features terminates testing. A typical test-end criterion defines that all features with high risk have to be tested, features with medium risk are optional candidates to be tested to reach the required test coverage, and features with low risk are only tested if all others have been tested and resources are available.

As additional benefit, the automation supported determination of failure probability of features makes testing more effective and helps to find more defects in important features (which are prioritized for testing) before deployment. Each test run contains a test result for each of its executed test cases and is traceable to a feature. Depending on the test results, a problem ticket is created. As soon as the test-end criterion is reached, a test report is created.

6 Threats to validity

In this section, we present the threats to validity of our results and the applied countermeasures. Referring to Runeson and Höst [2], 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 is the definitions of terms and concepts in the case context vs. 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. 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 is 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 (considering a project, a product, and a process), 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. To allow external comparison, we have presented the context of each case as clearly as possible, given the confidentiality constraints we have. On the risk side, there are so many variation factors in every context, that we may have focused on other than the key ones. Only replicated studies may help assessing the external validity of our study.

7 Summary and conclusion

In this article, we presented a multiple case study on currently applied risk-based testing approaches in industry. It is based on three industry cases from different backgrounds. The first case investigates testing in context of a develop-

Table 2 Summary of findings structured according to research questions RQ1 to RQ4 as well as Cases A, B, and C ("+" = confirming observation, "−" = controverting observation, empty = no observation)

Research question	Finding		A	B	C
RQ1: What is the notion of risk in software testing?	F-01	Testing is a measure to reduce risk exposure, the focus is on product risks, the risk concept is dependent on context and scope	+	+	+
	F-02	Common definition of risk R = P I applies, definition and relation of P and I may remain informal and implicit, usually based on (subjective) estimates	+	+	+
	F-03	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?	F-04	Risk is considered in all testing activities, even when not following an explicit risk-based testing approach	+	+	+
	F-05	Risk information from testing is not explicitly considered in making release decisions	+	+	−
	F-06	Risk information is used to extend testing towards "risky" areas, risk information is used as a guideline to focus testing on "risky" areas	+	+	+
	F-07	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?	F-08	Risks at the level of project management are individual entities, risks in testing are calculated from risk information associated to testable entities	+		+
	F-09	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		+	+
RQ4: What are the benefits of a risk-based testing approach?	F-12	Make testing more effective: detecting additional defects in testing, fewer defects slip through to the field	+	+	+
	F-13	Make testing more effective: prioritization for detecting most critical defects first, reduces overall stabilization costs and time	+	+	+
	F-14	Risk information used for informed decision-making and new insights to triangulate and refine decisions		+	+

ment project extending a large information system, the second one analyzes testing of a software product for measurement and diagnostic in the electrical power industry, and the third one considers the test process of a system integrator of telecommunication solutions. The main analysis across the three cases was conducted with qualitative methods. Results were gained by analyzing documents, tools, and interview protocols keeping a clear chain of evidence. Based on the research objective to explore how risk is defined, assessed, and applied in industry to support and improve testing activities, four research questions were defined and findings were derived by cross-case analysis. A summary of the findings is shown in Table 2.

The findings shown in Table 2 have been derived from the synthesis of the observations we made, when investigating the three different cases. Findings with confirming observations are marked by a "+" in the column of the corresponding case; contradicting observations are marked by a "−" sign. An empty cell indicates that no observations related to the described finding were made. Each finding has been confirmed by observations from at least two different cases. The findings are grouped according to the four research questions we studied in this article.

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 of 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 to 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.

Based on the findings derived by cross analysis of the three cases in context of the research questions, suggestions can be provided how to enhance usage of risks for testing purposes in each of the three cases. In Case A, the next step to enhance risk-based testing could be an established risk assessment procedure providing objective and reliable results based on automatically collected measurement data and expert estimates. Case B could benefit from an explicit risk-based test plan guiding decisions which regression tests to select and where to invest in test automation. Finally, risk-based testing in Case C could be improved using risk information to control test design.

In future, we plan to extend our suggestions on how to enhance risk-based testing in each of the three cases to general guidelines for the stepwise introduction of risk-based testing into existing test processes. These guidelines may be based on a stage model for introducing risk-based testing. Our findings on how risks support the testing activities provide important insights for developing such a model. The mitigation of risks is a strong driver for introducing risk-based testing. Risk information may first be considered in the test planning, design, and execution phases, and afterwards in the test evaluation and reporting phases. Optionally, release quality statements and residual risk estimations may be considered. We plan conducting additional case studies to investigate the introduction of risk-based testing and to further increase empirical evidence of risk-based practices in industry.

Acknowledgments This work has been supported by the COMET Competence Center program of the Austrian Research Promotion Agency (FFG), the project QE LaB Living Models for Open Systems (http://www.qe-lab.at) funded by the Austrian Federal Ministry of Economics (Bundesministerium für Wirtschaft und Arbeit), the project MOBSTECO funded by the Austrian Science Fund (FWF) as well as the competence network Softnet Austria (http://www.soft-net.at) funded by the Austrian Federal Ministry of Economics (Bundesministerium für Wirtschaft und Arbeit), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH (SFG), and the city of Viennas Center for Innovation and Technology (ZIT).

References

1. Wendland, M.F., Kranz, M., Schieferdecker, I.: A systematic approach to risk-based testing using risk-annotated requirements models. In: ICSEA 2012. The Seventh International Conference on Software Engineering Advances, pp. 636–642 (2012)
2. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131–164 (2009)
3. Yin, R.K.: Case study research: design and methods, vol. 5. Sage (2009)
4. Bach, J.: Heuristic risk-based testing. Softw. Test. Qual. Eng. Mag. **11**, 99 (1999)
5. Amland, S.: Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. J. Syst. Softw. **53**(3), 287–295 (2000)
6. Redmill, F.: Theory and practice of risk-based testing. Softw. Test. Verif. Reliab. **15**(1), 3–20 (2005)
7. van Veenendaal, E.: Practical risk-based testing: the PRISMA approach. UTN, Cambridge (2012)
8. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. Software Quality. Process Automation in Software Development, pp. 159–180 (2012)
9. ISO: ISO/IEC/IEEE 29119 Software Testing (2013). Available at http://www.softwaretestingstandard.org/. Accessed May 6 2014
10. van Veenendaal, E. (ed.): Standard glossary of terms used in software testing, version 2.2. Technical report, International Software Testing Qualifications Board, Glossary Working Party (2012)
11. ISO: ISO/IEC/IEEE 24765:2010 System and software engineering—Vocabulary (2010)
12. Cavano, J., McCall, J.: A framework for the measurement of software quality. ACM SIGMETRICS Perform. Eval. Rev. **7**(3–4), 133–139 (1978)
13. Redmill, F.: Exploring risk-based testing and its implications. Softw. Test. Verif. Reliab. **14**(1), 3–15 (2004)
14. Standards Australia/New Zealand: Risk Management AS/NZS **4360**, 2004 (2004)
15. Karolak, D., Karolak, N.: Software Engineering Risk Management: A Just-in-Time Approach. IEEE Computer Society Press, Los Alamitos (1995)
16. Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research. IBM Press (2002)
17. Stallbaum, H., Metzger, A.: Employing requirements metrics for automating early risk assessment. In: Proceedings of MeReP07, Palma de Mallorca, Spain, pp. 1–12 (2007)
18. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software Test, pp. 67–70. ACM Press, New York (2008)
19. Felderer, M., Ramler, R.: Experiences and challenges of introducing risk-based testing in an industrial project. In: Software Quality. Increasing Value in Software and Systems Development, pp. 10–29. Springer, Berlin (2013)

20. Yoon, H., Choi, B.: A test case prioritization based on degree of risk exposure and its empirical study. Int. J. Softw. Eng. Knowl. Eng. **21**(02), 191–209 (2011)
21. Souza, E., Gusmão, C., Venâncio, J.: Risk-based testing: a case study. In: IEEE 2010 Seventh International Conference on Information Technology: New Generations (ITNG), pp. 1032–1037 (2010)
22. Souza, E., Gusmao, C., Alves, K., Venancio, J., Melo, R.: Measurement and control for risk-based test cases and activities. In: 10th Latin American Test Workshop, pp. 1–6. IEEE Press, New York (2009)
23. Borland: SilkCentral (2013). Available at http://www.borland.com/products/silkcentral/. Accessed November 30 2013
24. Microtool: in-Step (2013). Available at http://www.microtool.de/inStep. Accessed November 30 2013
25. McCabe, T.: A complexity measure. IEEE Trans. Softw. Eng. 308–320 (1976)
26. SonarSource: Sonar (2013). Available at http://www.sonarsource.org/. Accessed November 30 2013
27. Felderer, M., Beer, A.: Using defect taxonomies to improve the maturity of the system test process: results from an industrial case study. In: Software Quality. Increasing Value in Software and Systems Development, LNBIP 133, pp. 125–146. Springer, Berlin (2013)