Information and Software Technology 56 (2014) 1597–1612

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

How are software defects found? The role of implicit defect detection, individual responsibility, documents, and knowledge

Mika V. Mäntylä *, Juha Itkonen

Department of Computer Science and Engineering, Aalto University, Finland

ARTICLE INFO

Article history:
Received 30 March 2013
Received in revised form 11 December 2013
Accepted 11 December 2013
Available online 3 January 2014

Keywords:
Software testing
Defect detection
Activities
Documents
Human factors
Industrial questionnaire study

ABSTRACT

Context: Prior research has focused heavily on explicit defect detection, such as formal testing and reviews. However, in reality, humans find software defects in various activities. Implicit defect detection activities, such as preparing a product demonstration or updating a user manual, are not designed for defect detection, yet through such activities defects are discovered. In addition, the type of documentation, and knowledge used, in defect detection is diverse.
Objective: To understand how defect detection is affected by the perspectives of responsibility, activity, knowledge, and document use. To provide illustrative numbers concerning the multidimensionality of defect detection in an industrial context.
Method: The data were collected with a survey on four software development organizations in three different companies. We designed the survey based on our prior extensive work with these companies.
Results: We found that among our subjects ($n = 105$), implicit defect detection made a higher contribution than explicit defect detection in terms of found defects, 62% vs. 38%. We show that defect detection was performed by subjects in various roles supporting the earlier reports of testing being a cross-cutting activity in software development organizations. We found a low use of test cases (18%), but a high use of other documents in software defect detection, and furthermore, we found that personal knowledge was applied as an oracle in defect detection much more often than documented oracles. Finally, we recognize that contextual factors largely affect the transferability of our results, and we provide elaborate discussion about the most important contextual factors. Furthermore, we must be cautious as the results were obtained with a survey, and come from a small number of organizations.
Conclusions: In this paper, we show the large impact of implicit defect detection activities in four case organizations. Implicit defect detection has a large contribution to defect detection in practice, and can be viewed as an extremely low-cost way of detecting defects. Thus, harnessing and supporting it better may increase quality without increasing costs. For example, if an employee can update the user manual, and simultaneously detect defects from the software, then the defect detection part of this activity can be seen as cost-free. Additionally, further research is needed on how diverse types of useful documentation and knowledge can be utilized in defect detection.

1. Introduction

Finding defects before release is an important and costly software engineering activity that is typically achieved through software testing and reviews. Plenty of academic work on software testing exists, but the connection between the academic work and the realities of software industry have repeatedly been found weak [1,2]. Even though some researchers have studied the actual practice of software testing in the industry [3–5], the diversity of the practice of software testing has not been addressed in academic research. Regarding software review, it has been found that in the industry they often suffer from poor reviewer preparation [6–8] and find lower share of functional defects than evolvability problems [6,9], which suggest that they could be secondary to software testing in detecting functional defects. As the diversity of various defect detection activities that might exist in the industry has not been studied in academic research, we study in this paper the variety of roles, activities, documents, and knowledge used by the people who detect defects in software development organizations.

In our previous case and field observation studies, we have identified the diversity of roles and activities, as well as documentation and knowledge, that are involved in defect detection and testing [10,11]. In this paper, we introduce the concept of implicit

* Corresponding author. Address: P.O. Box 15400, FI-00076 Aalto, Finland. Tel.: +358 505771684.
E-mail addresses: mika.mantyla@aalto.fi (M.V. Mäntylä), juha.itkonen@aalto.fi (J. Itkonen).

0950-5849/$ - see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.12.005

defect detection and study the amount and types of both implicit and explicit defect detection activities performed in four software development organizations. *Implicit defect detection* is an activity where one assesses the quality of the product and detects defects while working toward some other primary goal. The idea that humans can find defects while working towards some other goals has been previously investigated [12]. Additionally, it is utilized in industrial beta testing programs [13] and in internal usage of a company's own software products that is called alpha testing, or dogfooding, i.e., eating your own dog food [14,15]. Prior work has also studied the shares of defects detected in different types of testing and reviews [16,17]. However, to our knowledge, prior work has not studied the relative amount between implicit and explicit defect detection in software development organizations. For example, how many defects are found when testing the software vs. other software development activities that are not primarily done for QA purposes?

This study uses a survey instrument to provide a picture of defect detection activities at an organizational level in four case organizations. In this analysis, we study defect detection activity, the responsibility of individual finding defects, the type of documentation used in defect detection and the oracle information in defect detection. This study extends the earlier observation and case studies [10,11] that have identified the importance of personal knowledge in software testing, by investigating the amount and type of document and knowledge used at the organizational level.

This paper is structured as follows. Next, we present the research methodology and the analytical framework that we used in our analysis. In Section 3, we describe the results of the survey. In Section 4, we discuss our findings and present the related work. Finally, in Section 5, we provide the conclusion of this work.

2. Methodology

We collected data through a survey questionnaire from four software development organizations that we know well due to long-term research collaboration. We distributed the survey of defect detection in the development organizations and aimed it at wide coverage of professionals working in wide variety of roles. The measured variables are the number of found defects, document use, activities performed, personal knowledge, and organizational responsibilities.

Next, we describe the analytical framework in Section 2.1, followed by the definition of the exact research questions in Section 2.2. We continue with a description of the survey instrument and data collection in Section 2.3. Section 2.4 describes the measures and the data analysis procedures in detail. We introduce the case companies and the subjects of the survey in Section 2.5. Finally, Section 2.6 discusses the limitations of this study.

2.1. Analytical framework

The conceptual framework that we use in the analyses of the survey consists of three main dimensions concerning the defect detection phenomenon. The central concepts in our framework are described and motivated by the existing literature: *implicit and explicit defect detection*, *tester and non-tester roles*, and *documentation and knowledge* used in defect detection.

First, we propose dividing defect detection activities into explicit and implicit defect detection (see *Y*-axis in Fig. 1). We define *explicit defect detection* as an activity whose primary goals are to find defects and assess the quality of the product. Both goals of explicit defect detection can be achieved by various testing and review methods. In this paper, the explicit defect detection activities are software testing and software reviews or walkthroughs. We define

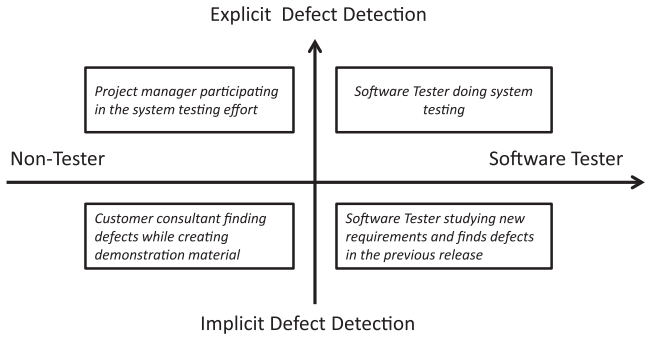

Fig. 1. Defect detection activity quadrants. *X*-axis represents the role dimension and *Y*-axis represents the activity dimension of software defect detection.

implicit defect detection as an activity where one assesses the quality of the product and finds defects while working toward some other primary goal. We argue that almost all people have performed implicit defect detection. Implicit defect detection is very common in our lives as we form opinions and find defects in the things that we use. For example, we have opinions about the quality of our car, our smartphone, or the school we send our children to. Furthermore, we have probably found defects in the things that we use, and we might have even reported these defects back to the responsible organizations. The same kind of implicit defect detection goes on in software product companies, e.g., when a sales person creates a demonstration for the upcoming product release, he/she is performing implicit defect detection as there is a chance that the upcoming release might still have undiscovered defects. Implicit defect detection has also been harnessed by software companies by requiring their employees to use the upcoming alpha versions of the products, called alpha testing or dogfooding [15]. The implicit defect detection performed by external people is called beta testing [13]. The implicit–explicit distinction can be seen as part of experimental designs where subjects have had multiple goals, e.g., perform pension calculation and report data quality defects [12], and create high level test cases and find requirements defects [18]. However, in general, the idea of implicit software defect detection has received limited attention in prior works. We think that large shares of implicit defect detection happen in software development organizations every day, thus, the topic needs to be addressed.

Second, we study the organizational roles of the people performing software defect detection (both implicit and explicit). We divide the defect detection activity based on the roles *tester* and *non-tester* (see *X*-axis in Fig. 1). In our prior case study which was based on defect database data, we found that large shares of defects were found by non-testers [10]. The large contribution of non-testers to defect detection might be more common than previously thought, as further work by us [19], and independent researchers [20], has supported this finding. This paper extends prior works by connecting the roles with different implicit and explicit, defect detection activities as illustrated by the *defect detection activity quadrants* in Fig. 1. In this research, we use a survey instrument to replicate and confirm the results of earlier work that was based on database analysis [10] and interviews [19,20].

Third, we study the documents used and knowledge applied in software defect detection. Earlier work has indicated that documented test cases in manual testing in the software industry are often far from textbook examples and are sporadically used [4,10,21–23]. Furthermore, the benefits of having pre-designed test case documentation in manual testing in terms of defect detection effectiveness are questionable according to experiments comparing test-case-based and exploratory testing [24,25]. Thus, the ques-

tion of what documents are used for defect detection and testing in industry becomes of interest and is studied in this paper.

Finally, we also study the knowledge required to recognize failures in software defect detection. The important effects of knowledge have been recognized in numerous studies [11,22,26–30]. For example, the researchers conclude [26] that, "test design is to a considerable extent based on experience and experience-based testing is an important supplementary approach to requirements-based testing". It is good to notice that documents are, in fact, codified forms of knowledge. The effect of expertise or knowledge to defect detection performance is identified in other activities, such as usability reviews [29] and spreadsheet defect detection [30], in addition to software testing. This paper extends our previous qualitative field observation study [11] with a quantitative survey data.

2.2. Research questions

This work has four research questions that represent different dimensions of defect discovery (see Fig. 2). The research questions were motivated by the previous section.

- *RQ1 Responsibilities:* What are the shares of defects found by testers and non-testers and to what extent do non-testers participate in defect detection?
- *RQ2 Activities:* What are the shares of defects found in explicit and implicit defect detection and what activities contribute to implicit defect detection?
- *RQ3 Documents:* What are the shares of defects found with and without test cases and what other documents are used in defect detection?
- *RQ4 Knowledge:* What are the shares of defects found with different test oracles?

Regarding RQ2 and RQ3, this study is exploratory and with regard to RQ1 and RQ4, this study is confirmatory with respect to our prior work on the roles detecting defects [10] and the knowledge use in defect detection [11].

We use the term "defect" to refer to an incorrect behavior of the software system that one reports in the defect management system. This is the term that was used in our survey, and we use it also when describing and discussing our results.

2.3. Survey form and data collection

The data collection was made through an online survey form created with the LimeSurvey program (see Appendix A for the complete survey). The form was administered to the employees of four organizational units (see Section 2.5) through our company contact persons. In total, we had 105 valid answers, giving us a response rate of 38%. The response rate was high for an online survey, but it can be explained through our long term case company collaboration. The survey had six parts that are explained below and are depicted in Fig. 3.

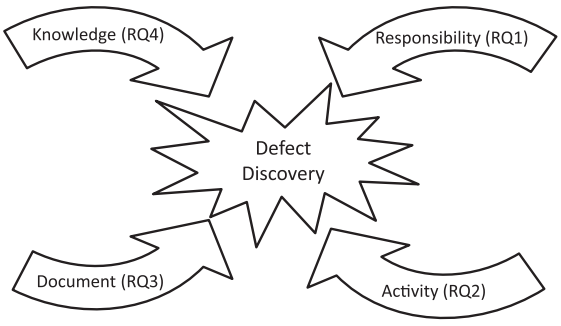

Fig. 2. Dimensions of the research questions.

First, we asked for the respondents' main job responsibilities and the number of defects they had reported during the past 12 months. The results regarding their main job responsibilities can be found in Table 2. The respondents were allowed to select as many job responsibilities as needed. All respondents selected at least one responsibility. Altogether, 188 job responsibilities were given. This meant that, on average, an individual selected 1.64 responsibilities. The frequencies show that 60 (57%) of the respondents selected one responsibility, 25 (24%) selected two, 12 (11%) selected three, six (6%) selected four, and only two respondents (2%) selected five or more responsibilities. In Table 2, the second column indicates how many respondents selected a particular responsibility. The third column shows the share of the particular responsibility. The other columns identify the number of pair combinations, e.g., the intersection between row "software testing" and column "software deployment" has the number 3, which means that there were three individuals who selected both software testing and software deployment as their responsibility. An illustrative example of a person with multi-role responsibility is a usability specialist who participated in feature design and testing, but focused on the usability perspective. Only three respondents (3%) selected "other." We analyzed the description of these three answers and were able to re-classify two of these three responses under our given responsibility classes. Based on this, and the low share of "other" answers, we think that our role list provided good coverage of the responsibilities of the individuals who participated in software defect detection.

Second, we asked how many percentages of defects were found in different activities in the companies. We used our prior experience in working with these companies to construct a meaningful list of activities (see Table 3). Our aim was to cover all activities that could reveal defects. Out of the answers, only 2.7% of the defects were detected in activity "other." We analyzed the textual description of the "other" answers and were able to re-classify them according to our activity list. Thus, we think that our activity list provided a good coverage of the activities where defects were detected.

Third, we asked how many percentages of different material, i.e., documents, were used when the respondents detected defects

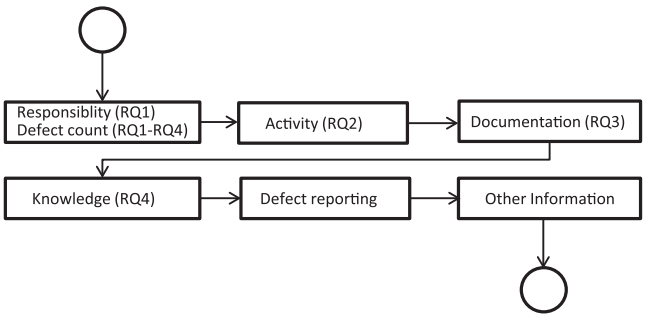

Fig. 3. Flow chart of the survey.

Table 1
Share of defects detected according to the defect detection activity quadrants (see Fig. 1).

	Non-tester (%)	Tester (%)
Explicit defect detection	17	21
Implicit defect detection	48	14

Table 2
Survey: responsibilities of the respondents (RQ1).

Role	n	Percent (%)	Software design and implementation	Software testing	Feature specification and design	Project or product management	Managing and leading people	Customer support	Product packaging	Software architecture	Software deployment	Customer consultation	Product sales	Other
Software design and implementation	43	41.0	–	11	8	7	6	1	0	9	4	1	0	0
Software testing	26	24.8	11	–	8	4	2	1	1	2	3	1	0	0
Feature specification and design	21	20.0	8	8	–	7	7	1	0	2	2	1	2	0
Project or product management	19	18.1	7	4	7	–	6	1	0	3	1	1	1	0
Managing and leading people	18	17.1	6	2	7	6	–	2	1	1	4	0	1	0
Customer support	13	12.4	1	1	1	1	2	–	2	0	2	1	0	0
Product packaging	12	11.4	0	1	0	0	1	2	–	0	1	0	0	0
Software architecture	10	9.5	9	2	2	3	1	0	0	–	1	0	0	0
Software deployment	10	9.5	4	3	2	1	4	2	1	1	–	2	0	0
Customer consultation	5	4.8	1	1	1	1	0	1	0	0	2	–	2	0
Product sales	3	2.9	0	0	2	1	1	0	0	0	0	2	–	0
Other	1	1.0	0	0	0	0	0	0	0	0	0	0	0	–

Table 3
Survey: list of the defect detection activities in the survey (RQ2).

- Preparing for product presentation or training
- Giving product presentation or training
- Specifying or designing features
- Technical software design
- Implementing the software (programming)
- Testing the software
- Participating in product review or walkthrough
- Deploying or installing the software
- Creating or updating product documentation
- Providing helpdesk service
- Internal usage of the software product
- Other, please specify below

Table 4
Survey: list of material that could be used in defect detection (RQ3).

- Test cases
- Release notes
- Product manual
- Product presentation or training material
- Product requirement or feature specification
- Technical product specification
- Message or report, e.g., e-mail, ticket of customer request, defect report
- I am not using any written material
- Other, please specify below

(see Table 4). Out of the answers, 4.2% of the detected defects were found with material "other." With the analysis of textual answers, we were able to re-classify many "other" answers, leaving us with 1.3% of the defects detected with true "other" answers. The true "other" answers were, "asking information from colleagues," and, "comparing the functionality with previous version." Thus, we think that the material list had a decent coverage of the materials used in these companies for defect detection, but in future studies, the survey should include the true "other" answers listed above.

Fourth, we asked how many percentages of defects were recognized based on different knowledge types (see Table 5). This list is based on observing knowledge use during exploratory software testing sessions [11]. For this question, the share of defect detected with "other" knowledge type was 0.9%, but we were able to re-classify all these answers according to our given knowledge list. Again, we think that our list of knowledge sources used for defect detection had good coverage over all possible knowledge sources.

Table 5
Survey: list of knowledge that could be used in defect detection (RQ4).

- Obvious errors, e.g. text too big for the text field, software crash
- Understanding how the software system should work (system logic)
- In-depth understanding of the application domain and its rules
- Customers and the real usage scenarios
- Documentation indicating the correct result
- Other, please specify below

Finally, we also had questions on defect reporting. However, as the focus of this article is the defect discovery, those results are not analyzed here. In the final page, the respondent had the possibility to give feedback in an open text field and leave his or her email address if interested in hearing about the survey results.

2.4. Measures and data analysis

The data was analyzed with the Microsoft Excel spreadsheet tool. We normalized all responses twice. First, we normalized all the percentage responses to 100%. This means that if the sum of the responses to a single question, from a single respondent, was more, or less, than 100%, the responses were normalized to 100% total. We had to do this, as our survey software did not have an applicable way to force 100%.[1] This normalization was also performed for the question regarding the documents used in defect detection in order to make it more convenient for the reader to understand the shares of the document use, even though it is quite possible to use multiple documents when detecting a single defect. Furthermore, if multiple documents are used when detecting defects, the contribution of each of those documents is likely to be smaller than if only one document was used. A second normalization was performed based on the number of defects reported by the respondents. This means that the answers from those reporting many defects have a larger weight in the results.

All research questions are operationalized with the *number of defects found* measure. This means that the number of defects that can be attributed to a particular responsibility, activity, document, or knowledge. For example, if a respondent has found 50 defects and indicates 50% defects are found with test cases and 50% are found with requirements documents, then the number of defects found with either of these documents is 25.

[1] The LimeSurvey has a way to force 100% sum for set of fields, but the feature did not do this automatically. Our pre-testing with survey suggested that it might decrease response rate as the feature put extra burden for the respondents.

2.5. Companies and respondents

We surveyed the three companies we had worked with in our previous case study of roles participating in defect detection [10]. The labeling of the companies A, B, and C is the same as in [10]. The only difference is that for company B, we had two departments participating. The departments are marked as B1 and B2, whereas in our previous work we only had data from department B1. The description of the cases (organization units) and companies of our study is listed in Table 6, wherein we describe the contextual variables that are typically reported and the ones that we assume to have an effect on the observed results. The affecting context variables are discussed in more detail in Section 4.5.

Our sampling strategy aimed at getting as high a coverage as possible on the individuals reporting defects in the case organizations. We requested a list of employees' email addresses through our company contact persons. In detail, the request said (translated from Finnish to English): "As respondents, we would like to have the widest possible range of people working in different roles (developers, testers, product managers, consultants, customer support, etc.) who find and report defects in your products. We do not want to focus only on testers as we wish to achieve a more complete picture of how defects are detected and reported in different activities and roles." The company contact personnel responded with a list of email addresses, which were then inputted into a LimeSurvey program that took care of sending invitations and reminders to the participants. Our response rate varied between 33% and 65% in the case organizations.

The distribution of respondents between our cases is unfortunately not equal (see Table 6). Case A had the fewest responses, with only 8 responses, and case B2 had the most responses, with a total of 51 responses. However, although case A had the lowest number of respondents (8), it also had the highest number of defects reported per respondent on average (61), and while case B2 had the highest number of respondents (51), it also had the lowest average of defects found per respondent (36). This means that for cases where a lower number of responses were received, the responses came from more active individuals in terms of defect detection. Based on our prior analysis of the company defect databases, the distribution of defect reporting in these companies is power-law distributed— roughly 65% of defects were found by 20% of the individuals reporting defects. Thus, getting responses from the more active defect reporters was important when having a smaller sample.

2.6. Limitations

The main limitations of our results come from the small set of organizations and the limited number of respondents studied. Thus, we cannot claim representativeness and we must be cautions in making statements about generalizability, as we do not know what the shares of, e.g., implicit defect detection, would be in a larger set of companies. However, the small set of companies is also a strength, as we know these companies well from the prior studies, e.g., [10,31–33], which enabled us to construct a meaningful survey and especially helped us interpret the open text answers. Our goal was not a statistical generalizability, but instead, a more restricted survey in a known context, where the interpretations

Table 6
Organizational units of our case study and number of respondents.

Companies	A	B		C
Cases (organizational unit)	A	B1	B2	C
Personnel	>110 Employees	>80 in the studied division (>300 in the whole company)	240 in the studied division (>300 in the whole company)	>70 in the studied divisions (>100 in the whole company)
Customers	>200	>80	>1000	>300
Company age	>10 years	>20 years	>20 years	>20 years
Studied product	Single product	Two products for engineering in different fields	Single product for engineering design	Single product for engineering design
	Business software of specific industry	The products share a common technological core architecture	COTS type of software (i.e., not heavily integrated or customized)	Product has a separate core that is also used for another product
	Integrated directly into the customers' other business systems	Integrated directly into the customers' other systems		COTS type of software (i.e., not heavily integrated or customized)
	Many customization opportunities			
Release process	Internal monthly mainline release	External main release two times a year	External main release once a year	External main release once a year
	Majority of software development done in customer branch and later imported back to mainline	External minor release four times a year	Majority of software development conducted in main branch	External minor release once a year
	Projects encouraged to frequently update to the latest mainline release	Majority of software development conducted in main branch		Majority of software development done in main branch
Type of GUI	WIMP	WIMP + complex 3D modeling	WIMP + complex 3D modeling	WIMP + complex 3D modeling
Main model of organizing testing	Mainline testing by testing team	Team effort	Team effort + Supported by testing team	Team effort
	Projects responsible customer specific testing			
Separate testing organization	Separate testing team	None	Separate testing team	None
Response% – *N* responses – Sum of defects – Avg. number of defects	38% – 8 – 488 – 61	43% – 36 – 1541 – 43	33% – 51 – 1844 – 36	65% – 11 – 599 – 54

Table 7
Share of defects detected.

Research questions	Coarse-grained categorizations	Total (%)	Cases			
			A (%)	B1 (%)	B2 (%)	C (%)
RQ1: Responsibility	Tester	35	54	46	23	28
	Non-tester	65	46	54	77	72
RQ2: Activity	Explicit defect detection	38	61	35	34	42
	Implicit defect detection	62	39	65	66	58
RQ3: Documents	Test case	18	37	19	12	19
	Other document	66	63	66	65	73
	No document	16	0	15	22	8
RQ4: Knowledge	General & System knowledge	55	43	54	61	54
	Domain knowledge	37	48	39	33	39
	Document	7	9	7	7	7

of the results would be more reliable. One way to assess generalizability is through discussion of what are the most important context variables affecting the results, and we do this in Section 4.5. We think that this discussion of the context opens up new avenues for future studies. Next, we discuss the limitations in more detail.

Our results are affected by the survey sample. We asked, and received, a list of defect reporters from the case organization's contact person (see Section 2.5). Thus, the base sample should be representative of the defect reporters in the case organizations. However, our true sample is formed from the individuals who responded to the survey. Thus, it is likely that our results have a small variation from what the situation really is within the companies. Importantly, the conclusions of this study are not affected by this, as our conclusions would be the same even if we would base them on any single case (A, B1, B2, or C), as seen in Table 7. It is highly unlikely that all companies would have had such a highly skewed set of respondents, causing our overall conclusions to be different.

Our results are based on the respondents' recollection of their past behavior and their personal understanding about their role and main responsibilities, and these can be seen as a limitation. Thus, the measures we provide are illustrative only. Probably the respondents were able to refer the actual defect database regarding the total number of reported defects and other details to support their responses. However, it is likely that the respondents could not give fully accurate answers to the questions requiring recollection, e.g., during which activities, and based on what documentation, the defects were found.

The survey instrument itself may represent a threat to validity. For example, only two activities out of 12 represented explicit defect detection, and 9 activities represented implicit defect detection, and the other option in addition. Setting up response options this way may attract more responses from uncertain subjects to implicit testing activities, as they represent 75% of the items. However, we believe that the activity list represents the true variety or activities that take place in software development organizations. We wanted to have high level activities, e.g., testing and programming, and splitting testing between unit, integration, and system testing would have over-represented the testing activity in comparison to other activities. Thus, it might have biased the results in favor of explicit testing. Overall, it is difficult to say what kind of question setup should have been used to get the most realistic view of the true distributions. However, we think that our high-level conclusions would be the same regardless of the questions set up.

The time used for different activities was not collected in our survey and was also not available in the company time reporting systems, as they were used to track the billing of the work rather than the activity of the work. Thus, in the future studies, estimates of the spent effort should also be collected. However, based on the long collaborations with the companies, we estimate that implicit defect detection has a higher volume than explicit defect detection activities.

Overall, the limitations of this work mean that our data describes the defect detection phenomena, and how the different roles and activities contribute to the defect detection, but our data cannot be used to claim exact percentages of defects found by people working in certain roles in organizations. However, there was no better way of collecting such data. In our previous work [10], we learned that the defect databases in the companies rarely contained information that could be reliably used to answer our research questions. For example, some people would sometimes indicate the defect detection activity, but overall the activity field was too unreliable for analysis. Similarly, our previous work only used one role for each person, when in fact many people in the companies had multiple roles and responsibilities. Thus, the survey was created to fix these shortcomings of our prior work [10].

3. Results

In this section, we present the results of the study by looking at our four research questions addressing responsibilities (RQ1), activities (RQ2), documents (RQ3), and knowledge (RQ4) of defect detection in Sections 3.1, 3.2, 3.3, and 3.4, respectively. Table 7 presents a coarse-grained summary of the results for all the research questions, while Sections 3.1–3.4 provide a more detailed analysis of each of the research questions.

In Table 7, we see that in our survey data, individuals with tester responsibility detected 35% of the defects varying from 23% to 54% between the cases. When looking at the activities where the defects are found, we see similar shares with explicit defect detection activities finding 38% of the defects and the share ranging from 34% to 61% between companies. However, this does not mean that testers only find defects through explicit defect detection, as we describe in Section 3.2. Regarding the document use, we found that the test cases were used for detecting 18% of defects, while the role of other documents in defect detection was 66%. The role of other documents than test cases in defect detection was also stable ranging from 63% to 73%. Finally, when looking at the oracle information used for recognizing defects, the general & system knowledge was the most frequently used, with a 55% share, and domain knowledge also had a high share with 37%. Furthermore, documents rarely acted as test oracles in our cases, with only 7% share of the detected defects. For Table 7, the survey item options are classified as follows:

- *RQ1 Responsibility:* Share of Testers' defects come from all individuals who have selected "Software Testing" as their responsibility, regardless of what other responsibilities they have selected. Non-testers are everyone else who have not selected tester responsibility (see Table 2 and Table 8 for details).

- *RQ2 Activity:* Explicit defect detection is the share of defects from the item options, "Testing the software" or "Participating in product review or walkthrough." All other activities are classified as implicit defect detection (see Table 3 and Table 9 for details).
- *RQ3 Documents:* Test case is the share of defects found with the item option "Test Cases." "No document" is the share of defects found with the item option, "I am not using any written material." "Other document" includes all other documents used for defect detection (see Table 4 and Table 11 for details).
- *RQ4 Knowledge:* General & System knowledge is the share of defects found with the item options "obvious errors, e.g., text too big for the text field, software crash," and, "understanding how the software system should work." Domain knowledge is the share of defects found with the item options, "in-depth understanding of the application domain and its rules," or, "customers and the real usage scenarios." Finally, Document contains the share of the item option, "documentation indicating the correct result."

3.1. Responsibilities

Table 2 shows the distribution of work responsibilities of the respondents. The table also shows that each individual could have more than one responsibility. In Table 8, we see that software developers had the highest defect counts with 39% total share and software testers had the second largest share with 35%. When we analyze the average number of defects found per individual, we found that customer support found the largest share, 61.0 defects per individual respondent, followed by software testers, with an average of 60.3 defects. Software developers represented the largest group of respondents, thus, their total defect share was the largest even when their average number of defects (40.1) was below the overall average (42.4). Altogether, several roles contributed to the reporting of defects. The importance of roles varied, depending on whether we measured sum of contributions, or average contribution per individual. Software testers were ranked second highest, both in terms of average defects per individual and total share. To summarize, the testers' contributions were 35% of the defects, whereas non-testers' contributions were 65% (see Fig. 1 and Table 1).

3.2. Activities

The list of activities, during which defects were revealed, is shown in Table 9. Software testing was the leading activity for software defect detection from three viewpoints: First, the largest number of individuals, 81 (77%), had performed software testing activity to find defects; Second, software testing found the most defects in total, with a share of roughly one-third; and third, the

Table 8
Defect shares for work responsibilities.

Responsibility	*N*	Total defects	Total share (%)	Average defects
Software testing (=testers)	26	1567	34.8	60.3
Product sales	3	122	2.7	40.7
Customer consultation	5	102	2.3	20.4
Project or product management	19	834	18.5	43.9
Feature specification and design	21	1091	24.3	52.0
Software architecture	10	373	8.3	37.3
Software design and implementation	43	1725	38.4	40.1
Software deployment	10	337	7.5	33.7
Product packaging	10	234	5.2	23.4
Customer support	13	793	17.6	61.0
Managing and leading people	18	607	13.5	33.7
Other, please specify below	1	1	0.0	1.0
All[a]	105[a]	4497[a]	–[a]	42.8

Calculation of the results in Table 8.
N = number of respondents selecting the responsibility.
Total defects = sum of respondent's total defects that have selected the responsibility.
Total share = total defects for the responsibility/all reported defects (4497).
Average defects = total defects for the responsibility/number of respondents reported the responsibility.
[a] This cell is not the sum since a respondent could select more than one role.

Table 9
Defect shares for work activities.

Activity	*N*	Total defects	Total share (%)	Average defects
Testing the software (=explicit defect detection)	81	1537	34.5	19.0
Product review or walkthrough (=explicit defect detection)	25	164	3.7	6.6
Preparing for product presentation or training	33	182	4.1	5.5
Giving product presentation or training	22	102	2.3	4.6
Specifying or designing features	44	331	7.4	7.5
Technical software design	29	165	3.7	5.7
Implementing the software (programming)	50	776	17.4	15.5
Deploying or installing the software	30	115	2.6	3.8
Creating or updating product documentation	22	104	2.3	4.7
Providing helpdesk service	34	546	12.3	16.1
Internal usage of the software product	56	434	9.7	7.8
All activities (sum)	426	4457[a]	100	10.5

Calculation of the results in Table 9.
N = number of respondents selecting the activity.
Total defects = sum of respondents' share of defects for the activity.
Total share = total defects for the activity/all reported defects (4457).
Average defects = total defects for the activity/number of respondents reported the activity.
[a] This number deviates between tables because not all respondents responded to all questions.

average number of defects per individual was the highest for software testing. Although software testing is the most prominent activity in software defect detection, the share of one-third means that it is not a dominant activity, as other activities are responsible for finding two-thirds of the defects. Other prominent activities are software implementation with 17%, helpdesk with 12%, and internal usage with 10% share of the defects found. Review use in the companies appears to be sporadic, and the share of defects found with review is low at only 3.7%. Connecting the results of this section to the *Y*-axis in Fig. 1, which illustrates the distinction between explicit and implicit defect detection, allows us to state that explicit defect detection found 38% of the defects, while implicit defect detection found 62% of the defects.

Table 10 shows the cross-tabulation of the activities and responsibilities. The table shows how responsibilities and activities are interconnected in terms of defect detection. For example, if we pick software testing responsibility (row) and testing-the-software activity (column), we find the number 55%. This means that people who have software testing as one of their responsibilities found 55% of the total count of the defects while testing the software. When we look at the activities alongside the respondents' responsibility, we see that all roles actually find defects through several activities. For example, customer support people found 52% of defects when providing help desk service, and customer consultation found 31% of defects when preparing for and 22% when giving product presentations or trainings. Additionally, we can see that all roles also contribute to the activity of testing the software. This highlights the cross-cutting role of software testing activity.

3.3. Documents

We investigated the role that documentation played when detecting defects. The data on document use in defect detection is presented in Table 11. The most frequently used document in defect detection depends on the viewpoint: the number of individuals using the document, the total number of defects detected using the document, or an average number of defects detected per individual using the document. First, product requirements or specifications are used by the highest number of individuals, 66 (64%). Second, the total number of detected defects is the highest using message or report type of document, 969, followed by test cases and requirements, 804 and 711 defects detected, respectively. The large number of defects found with message or report type of documentation can partly be explained by the wide scope of documents that can be interpreted for inclusion in this group. Third, if we look into the average number of defects detected per individual using a document, we see that not having any written material has the highest average with 18.6 defects, followed by the message or report type of document and test cases—15.9 and 13.9 defects found on average, respectively (note: other material actually has the highest average, but since there are only two respondents, it is ignored). The survey results in Table 11 challenge the dominant role of test cases in software defect detection and highlights several sources and varieties of documented knowledge.

There were large differences in the use of documentation regarding different responsibilities in Table 12. Test cases were most frequently used by product packaging (30%), software testers (29%), and by software implementation (18%). Test cases were rarely used by product sales (8%), customer consultation (3%), and customer support responsibilities (7%). People with these responsibilities relied more on documented release notes and the product manual when detecting defects. This is not surprising as those people often interact with customers and, thus, work with and are interested in the documentation that is visible to the customer. Release notes (10%) and product manuals (18%) were also frequently used by people with product packaging responsibilities, who also work with such documents. Product presentation and training materials had the highest shares among sales people (18%), while other responsibilities had a negligible share. Requirement specifications were most frequently used by software deployment (22%), software architecture (22%), and project and product manager responsibilities (19%). Product requirements were seldom used by customer consultants (6%) and managers (9%).

Technical product specification was most frequently used by people working on software deployment who also used a lot of requirements documentation. It seems that people responsible for deployments found defects by studying the requirements from a technical viewpoint. They are the people responsible for the system setup on the customer site, and they want to make sure that the product installation runs technically smoothly when they install the product to the customer site.

Customer support people used messages or reports frequently (38%). They often get such messages from customers in the form of, for example, complaints, questions, or a defect report. In addition, people with management and leadership responsibilities often found defects while using such messages (38%). Finding

Table 10
Cross-tabulation of the defect shares for work responsibilities and activities.

Responsibilities	Activities (%)										
	Preparing for product presentation or training	Giving product presentation or training	Specifying or designing features	Technical software design	Implementing the software (programming)	Testing the software	Product review or walkthrough	Deploying or installing the software	Creating or updating product documentation	Providing helpdesk service	Internal usage of the software product
Product sales	30.6	19.4	6.8	0.0	0.0	**29.3**	**0.0**	2.5	0.0	4.7	6.7
Customer consultation	30.6	21.8	0.6	0.0	1.6	**14.7**	**3.5**	9.4	0.0	9.6	8.2
Project or product management	9.4	5.0	9.4	2.5	7.7	**43.2**	**3.4**	2.0	0.8	4.5	12.0
Feature specification and design	6.9	4.3	13.1	2.5	17.3	**31.6**	**5.7**	0.5	1.2	3.7	13.1
Software architecture	6.3	6.5	7.0	11.3	34.5	**20.9**	**5.0**	0.2	0.0	4.6	3.8
Software design and implementation	3.4	2.6	7.5	7.0	38.1	**25.2**	**3.2**	1.4	0.6	3.0	8.0
Software testing	**2.4**	**1.1**	**7.0**	**2.6**	**14.7**	55.2	**5.6**	**1.2**	**1.4**	**2.2**	**6.7**
Software deployment	0.5	2.0	5.2	7.7	32.6	**16.3**	**3.7**	9.9	0.0	9.2	12.9
Product packaging	0.0	2.1	1.3	1.3	6.4	**23.3**	**0.0**	5.3	21.9	30.7	7.7
Customer support	1.2	1.8	3.0	1.2	7.4	**14.9**	**0.5**	3.7	2.9	51.5	11.9
Managing and leading people	4.5	2.8	7.4	3.3	14.8	**34.1**	**4.3**	4.9	0.0	8.6	15.4
Total Share	4.1	2.3	7.4	3.7	17.4	34.5	3.7	2.6	2.3	12.3	9.7

Calculation of the results in Table 10.
The percentages are counted as the total share column in Table 9. Table 10 extends Table 9 by providing activity shares for each responsibility. Each percentage shows the share of defects found in each activity, e.g., the total share of the testing activity is 34.5%, but for respondents selecting testing responsibility, it accounts for 55.2%. The rows add up to 100%. The percentage is accounted for by taking each responsibility and computing what percent of their defects can be accounted to a particular activity.
Columns with Explicit defect detection activity are marked with bold. Rows with main responsibility on defect detection are marked with bold.

Table 11
Defect shares for document usage.

Document	*N*	Total defects	Total share (%)	Average defects
Test cases	58	804	18.2	13.9
Release notes	31	206	4.7	6.7
Product manual	54	507	11.4	9.4
Product presentation or training material	29	138	3.1	4.7
Product requirement or feature specifications	66	711	16.1	10.8
Technical product specification	42	351	7.9	8.4
Message or report, e.g., e-mail, ticket of customer request, defect report	61	969	21.9	15.9
I am not using any written material	37	685	15.5	18.5
Other, please specify below	2	56	1.3	27.8
All documents	380	4427	100	11.7

Calculation of the results in Table 11.
N = number of respondents selecting the document.
Total defects = sum of respondents' share of defects for the document.
Total share = total defects for the document/all reported defects (4427).
Average defects = total defects for the document/number of respondents reported the document.

Table 12
Cross-tabulation of the defect shares for work responsibilities and used documents.

Responsibility	Test cases (%)	Release notes (%)	Product manual (%)	Product presentation or training material (%)	Product requirement or feature specifications (%)	Technical product specification (%)	Message or report, e.g., e-mail, ticket of customer request, defect report (%)	I am not using any written material (%)	Other, please specify below (%)
Product sales	7.9	1.6	13.9	17.5	13.2	0.0	6.3	39.5	0.0
Customer consultation	3.1	6.9	16.6	4.3	5.9	2.4	18.3	42.5	0.0
Project or product management	11.3	4.7	8.6	4.6	19.2	8.2	32.0	11.3	0.0
Feature specification and design	11.5	0.6	9.6	3.0	19.1	4.8	28.6	22.4	0.5
Software architecture	6.5	1.9	0.7	1.0	21.6	13.0	24.4	30.9	0.0
Software design and implementation	18.4	3.6	9.3	2.4	16.0	9.9	17.8	19.7	2.9
Software testing	28.6	1.1	12.1	3.1	20.4	9.9	11.0	13.8	0.0
Software deployment	5.1	4.4	5.2	0.8	21.2	22.5	16.5	24.3	0.0
Product packaging	24.0	6.1	28.8	2.9	5.6	0.7	26.3	5.7	0.0
Customer support	6.7	9.9	19.1	4.6	11.0	5.0	37.5	6.3	0.0
Managing and leading people	15.1	2.7	3.4	3.6	8.8	4.4	36.1	26.0	0.0
Other, please specify below	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0
All	18.2	4.7	11.4	3.1	16.1	7.9	21.9	15.5	1.3

Calculation of the results in Table 12.
The percentages are counted as the total share column in Table 11. Table 12 extends Table 11 by providing document shares for each responsibility. Each percentage shows the share of defects found using each document, e.g., the total share of the test cases is 18.2%, but for respondents selecting testing responsibility it accounts for 28.6%. The rows add up to 100%.
The percentage is counted by taking each responsibility and computing what percent of their defects can be accounted to a particular document.

defects without any type of documentation was most frequent in product sales (40%), customer consultation (38%), software architecture (30%), and managing and leading (25%). We think that people with these responsibilities possess strong personal knowledge of the product and prefer relying on their experience, rather than written documentation, when detecting defects.

3.4. Knowledge as oracle

Our previous work had shown that the tester's personal knowledge could be a significant factor when recognizing software defects [11,22,34]. We investigated knowledge usage as a defect detection oracle with previously created knowledge classification. We compared the personal knowledge use with codified knowledge, i.e., documentation indicating the correct results. Our knowledge classification had three levels: (1) generic correctness (in Table 13, as, "obvious errors…"); (2) system knowledge (means knowledge of the features and technical details of the tested software system—in Table 13 as, "understanding how the software system should work…"); (3) domain knowledge (in Table 13 as, "customer and real usage scenarios," and, "in-depth understanding of the domain rules"). We also acknowledge that defects can be found when comparing the output with, "documentation indicating the correct result" (see Table 13).

Table 13 shows the type of knowledge used for detecting defects. Most frequently, defects were detected with generic correctness, i.e., obvious errors requiring the lowest level of knowledge. It was used by 96 (93%) of the respondents and the total share of defects detected by generic knowledge (28%) was also the highest. The second frequent knowledge type was system knowledge used by 88 (85%) of the respondents and it also had the second highest share of defects. Additionally, it had the highest average of defects found (13.9) among our respondents. Customers and real usage scenarios was the third frequently used knowledge type, 77 (75%) of the respondents. It was followed by an in-depth understanding of a domain knowledge type that was used by 61 (59%) of the respondents. Finally, using documentation that provided the correct result was only used by 48 (47%) respondents and the average number of defects detected with that method was only 6.7, compared to the overall average of 12.1.

Table 14 shows how the knowledge was distributed to different responsibilities. All responsibilities had considerable shares of generic knowledge and system knowledge types. Respondents with customer consultation and customer support responsibilities had the lowest share of obvious defects—only 9% and 19% shares,

Table 13
Defect shares for knowledge usage.

Knowledge type	*N*	Total defects	Total share (%)	Average defects
Obvious errors, e.g., text too big for the text field, software crash	96	1268	28.4	13.2
Understanding how the software system should work (system logic)	88	1222	27.3	13.9
In-depth understanding of the application domain and its rules	61	706	15.8	11.6
Customers and the real usage scenarios	77	953	21.3	12.4
Documentation indicating the correct result	48	323	7.2	6.7
All	370	4472		12.1

Calculation of the results in Table 13.
N = number of respondents selecting the knowledge type.
Total defects = sum of respondents' share of defects for the knowledge type.
Total share = total defects for the knowledge type/all reported defects (4472).
Average defects = total defects for the knowledge type/number of respondents reported the knowledge type.

Table 14
Cross-tabulation of the defect shares for work responsibilities and knowledge types.

	n	Total defects found	Obvious errors, e.g. text too big for the text field, software crash (%)	Understanding how the software system should work (system logic) (%)	In-depth understanding of the application domain and its rules (%)	Customers and the real usage scenarios (%)	Documentation indicating the correct result (%)	Other, please specify below (%)
Product sales	3	122	34.5	28.2	11.6	18.9	6.8	0.0
Customer consultation	5	102	9.3	36.2	16.2	35.7	2.5	0.0
Project or product management	19	824	27.9	30.1	18.6	19.0	4.4	0.0
Feature specification and design	21	1091	29.7	24.8	15.3	25.2	5.1	0.0
Software architecture	9	358	21.1	46.6	21.8	8.8	1.7	0.0
Software design and implementation	42	1710	28.5	33.9	19.6	13.0	5.0	0.0
Software testing	28	1567	33.1	21.6	16.9	19.6	8.7	0.0
Software deployment	10	337	26.1	36.9	15.6	19.4	1.9	0.0
Product packaging	13	234	24.4	26.5	8.3	18.7	22.1	0.0
Customer support	14	793	18.9	29.4	6.4	31.2	14.0	0.0
Managing and leading people	18	607	34.4	24.3	15.5	23.3	2.6	0.0
Other, please specify below	1	1	58.3	41.7	0.0	0.0	0.0	0.0
All	103	4472	28.4	27.3	15.8	21.3	7.2	0.0

respectively. The system knowledge was the most prominent knowledge type for software architects (47%) and least prominent for software testers (21%). Knowledge of customers and real usage scenarios was most frequently used by people with close customer contact, i.e., customer support and consultants who had 36% and 31% shares, respectively and it was least frequently used by software architects (9%) and developers (13%). On the other hand, in-depth understanding of the domain was used most frequently by software architects (22%) and developers (20%), whereas being infrequently used by customer support (6%) and product packaging (8%). Documentation was used to recognize small share of defects (7%) overall. However, it was more frequently used by the people working with product packing or customer support responsibilities, who detected 22% and 14% of the defects based on documentation, respectively.

Overall, the differences between responsibilities were what one would intuitively expect. However, the diversity in the types of knowledge across responsibilities was smaller than we initially anticipated. For example, for in-depth domain knowledge, 8 out of the 11 responsibilities are between 15% and 22%.

4. Discussion and related work

We structure the discussion according to the four research questions, presented in Section 2.2, and discuss our findings in relation to earlier research. Finally, in the last subsection we discuss the effects of the context of this study and the generalizability of our results.

4.1. RQ1 Responsibilities

The first research question was: "What are the shares of defects found by testers and non-testers and to what extent non-testers participate in defect detection?"

We found that the share of defects detected by the testers in this study was roughly 35% and for the non-testers the share was 65% (see Table 7). In this study, the testers included all respondents that had selected "Software Testing" as their responsibility, while non-testers are all others. This result, acquired through the survey, confirms our prior findings [10] on the large contribution of non-testers in defect detection. In [10], the share of defects found by testers was only 10% in three of the four cases studied in this paper based on the defect database data, whereas in this work, we found a share of 35%. The explanation for the large difference is individuals having multiple roles in the companies. In our prior work, we used the single role that we found in the organizational charts of the companies for each employee. In reality, the individuals had multiple responsibilities, e.g., a person responsible for product documentation could also have the responsibility for software testing. In this paper, the respondents were allowed to select multiple responsibilities. Therefore, this paper represents the upper boundary of testers in the cases.

In other works, the amount of testing work done by non-testers has been investigated. Recently, Mahmud et al. [20] found that at IBM, testing is also done by non-testers. They developed a test automation tool that was particularly helpful for the non-testers, as it worked with a capture replay approach while producing high-level test code that could be edited, even with individuals with limited technical skills. Thus, they present a tool approach on how non-testers can be made more productive. In a case study of software testing in the automotive industry, it was found that dedicated human resources for software testing were often lacking, leaving testing as everyone's and, thus, no-one's, job [19]. Rooksby et al. [23] point out that testing is cooperative work by qualitatively describing testing activities at four software development sites where testing was done by programmers, proxy customers, and testers. However, that paper is more focused on the other social dimensions in general rather than roles and no quantitative data of the defect detection is given. In this paper, we can see the large share of non-tester contributions to defect detection activity in Table 10.

To summarize, the main conclusions of this and prior works [10,19,20,23] are the same: *defect detection is a cross-cutting activity involving multiple roles*, either due to resource constraints [19] or by design [10].

4.2. RQ2 Activities

The second research question was: "What are the shares of defects found in explicit and implicit defect detection and what activities are part of implicit defect detection?"

In Section 1, we defined explicit software defect detection as an activity whose primary goal is to find defects and assess the quality of the product, i.e., software testing and reviews. We defined implicit defect detection as an activity when one assesses the quality of the product and finds defects while working toward some other primary goal. Our results indicated that implicit defect detection activities revealed more defects than explicit ones. This may be due to the higher number of hours used in implicit defect detection. Table 7 shows that explicit defect detection (software testing or reviews) found 38% of the defects, while implicit defect detection (all other activities) found 62% of the defects.

When looking inside explicit defect detection, we find that testing found almost ten times more defects as compared with reviews. Our knowledge from our long collaboration with companies indicates that they performed reviews, but the practice was sporadic and done perhaps more for knowledge distribution benefits rather than defect detection. Similar findings can be found from an international industrial survey with 226 respondents [8] that indicated that reviews are unsystematically applied in the industry. Our findings support those results. A case study illustrating why reviews might fail in industry [35] suggests that reviews require extensive enforcing and are, thus, dependent on the enthusiasm of the review champion, which may fluctuate over time. However, we do not know whether there is a connection between the low defect detection share of reviews and the high defect detection share of implicit defect detection activities. Nevertheless, our findings mean that implicit defect detection is a highly important quality assurance practice in the studied companies. Yet, it seems that very little prior work of the implicit vs. explicit distinction of defect detection exists. Next, we discuss the relevant prior work and Table 15 provides a summary of the existing knowledge.

Robillard and Francois-Brosseau [36] highlight the importance of explicit defect detection (testing activity only in their case) in a small industrial study. They found that when software developers were required to make testing activities explicit, it resulted in big improvements in product quality. The idea was to make developers aware of the goal of the testing, instead of seeing it as an obstacle preventing them to complete a task. On the other hand, having more explicit defect detection activities would increase the total effort, as compared with implicit defect detection as part of another task, e.g., designing new features or preparing for product demonstrations.

We were able to find two works that have, in fact, studied implicit defect detection in the areas of software inspections [18] and data quality checking [12]. Fogelström and Gorschek [18] propose a reading technique called test-case-driven inspection that is based on perspective-based reading [37]. In test-case-driven inspection, the testers inspect requirements while creating high-level test cases based on the requirements. Although original authors make no such claim, we think that the test-case-driven inspection can be interpreted as an implicit inspection when creating the test cases is the primary goal. The authors also show, with a controlled experiment, that test-case-driven inspection finds the same number of defects in total, but finds more major defects than checklist-based reading.

Klein et al. [12] studied the effect of defect detection goal settings and incentives from data when the subjects worked on pension calculation tasks. The study shows that if the defect detection goal was vaguely described in the instructions, the subjects detected only 7% of the defects. When the defect detection goal was stated clearly, the subjects detected 31% of the defects. Finally, when a monetary incentive was added for finding the defects, then the subjects found 56% of the defects. Their results suggest that making the defect detection goal explicit can improve the defect rate in the case of implicit defect detection activity and increasing the incentives further improves the defect detection results. Thus, making the defect detection goals explicit for the implicit defect detection activities can be a cost-effective approach for improving the defect detection results in software development organizations.

Alpha testing or dogfooding [15] and beta testing are defect detection activities where several people use a new version of the product before it is released to the market. These activities utilize implicit defect detection at an individual level to find defects, but at the company level, such activities are in fact explicit defect detection. For example, companies have beta-testing programs and strategies [13], highlighting the explicit nature of these activities. At the individual level, the defect detection is implicit as the individuals are using the product for primary goals other than defect detection and quality evaluation. Despite the apparent wide usage of alpha and beta testing and a large num-

Table 15
Existing knowledge of implicit defect detection activities.

Knowledge	Source
Humans are able to find a good share of defects when performing other tasks	This paper and [12,18]
Making defect detection and testing more explicit, even within the implicit activity, increases the share of defects detected	[12,36]
Implicit defect detection through various activities is a highly important quality assurance method in certain industrial contexts	This paper
Alpha testing or dogfooding and beta testing represent perhaps the most institutionalized and widely spread implicit defect detection activities	[13–15]
The defect detection done when additionally performing other tasks is more effective in finding major defects, than defect detection activity alone	[18]

ber of articles that mention such testing, we are only aware of one research article that has a primary focus on this practice [13]. As that article is soon 20 years old, this is an important avenue for future research.

To summarize, implicit defect detection activities seem to be commonplace in the industry. However, it seems that prior work on the topic is scattered and systematic literature reviews of this topic are likely to be difficult when common vocabulary is missing. The existing research [12,18] shows that humans can detect defects while simultaneously performing other tasks like calculating pensions or creating test cases. The primary task in implicit defect detection does not necessarily impair the defect detection performance in comparison to an explicit defect detection task, but can result in a similar performance [18]. However, neither of those studies [12,18] reports the quality level of the main tasks (test cases produced or pension calculation performed). Perhaps the increased focus on the defect detection goal decreases the quality of the output of the second task. Finally, implicit defect detection can be improved by making the defect detection goal more prominent and by offering rewards for finding defects [12].

We conclude that, as the contribution of implicit defect detection activities is high, it can be beneficial to explicate the defect detection goals of the implicit defect detection tasks

We also highlight that the *explicit–implicit dimension of defect detection activity is actually a continuum* where the focus on the defect detection task can vary from highly explicit to very implicit (see *Y*-axis in Fig. 1).

4.3. RQ3 Documents

The third research question was: "What are the shares of defects found with and without test cases and what other document are used in software defect detection?"

In this study, test cases were the second most frequently used document for defect detection, with 18% share, whereas the most frequently used document type was a message or report (22%) (see Table 11 and Table 7). This means that when studying defect detection activities and especially manual testing, researchers should also consider other documents and written information sources in addition to test cases. Testing with the help of product requirements (16%), product manual (11%), and testing completely without documents (16%) should also be considered as testing methods and studied in software engineering research. Briand [38] points out, in the context of software of test automation, that testing needs to be done based on technical documents such as state charts and sequence diagrams that act as input for the test case creation process. The companies participating in this research relied heavily on manual testing, despite having automated testing in place as well. We think that in manual testing, knowledgeable individuals are able to perform relevant testing based on their knowledge and with the help of diverse types of documentation, such as natural language requirements, manuals, or reports.

We are not aware of any prior work that provides detailed data of documents used in industrial software defect detection. However, from industrial studies of software testing, we can find snippets of results on test case usage in manual testing. Ahonen et al. [21] report in a case study that the studied companies had poor test case design management. Engström and Runeson [39] found that designed test cases had design redundancy and execution redundancy. Itkonen and Rautianen [22] report that companies with rich GUI products thought that it was impossible to write test cases for all possible combinations, which was one reason they favored an exploratory type of testing. In summary, these papers indicate that in industrial practice the test cases often do not match the examples presented in the textbooks. Itkonen and Rautiainen [22] also indicate that often manuals are used as a basis for exploratory software testing. This matches our results as the product manual was one document often used for software testing.

Our work relates to the connection between requirements and testing that has gained some attention in recent years [40–43]. In those works, the majority of attention has been given to model-based testing, formal approaches, and traceability between requirements and testing. Barmi et al. conclude that there is still a significant gap between requirements and testing in research [41]. In this paper, we confirm the important connection between requirements and testing activities by showing that product requirements are an often-used document (3rd most frequently) when detecting defects. This emphasizes the need for more research on the use of requirement documentation in testing.

Defect detection and testing in particular without using documents (the 4th most frequent response) has been studied under the names of exploratory software testing and ad hoc testing (see [11] for more references). However, none of the past works have indicated the share of defect detection activities that are performed without documents. Our initial assumption was that the share of document-less defect detection would have been higher since we knew that the companies did not use test cases very often. Thus, it appears that the lack of test case usage is actually compensated by the use of other documents. In our prior work [24,25] we have compared testing with test cases and without test cases (exploratory testing with the user manual as source documentation). The results of this paper reveal the diverse set of documentation used in defect detection, which suggests for more research on how different development documentation can be utilized in testing, and what types of documentation would be most beneficial for testing.

The use of messages, reports, or tickets in software defect detection relates to communication between people, e.g., customer support gets messages or defect reports from customers and reveal defects when working on such requests. Communication and collaboration in software engineering has been studied in recent years, focusing on distributed software development [44]. Grechanik et al. [45] presents challenges and a research agenda for using distributed test organization in software development. However, we are not aware of any studies that would have focused on communication usage when detecting defects.

The responses to the "other document" question also revealed that the information source can be another person that is consulted about the application, instead of actual documents. Previous versions of the software provide another information source that is used when detecting defects. These sources should be further investigated in the future. For example, communication studies could be used to identify the potential application experts. Expertise mining has already been presented in a software implementation area where expertise regarding the use of software functions has been mined from a software repository [46]. However, the mining approach might not work for finding high-level product expertise, as the software repositories may not contain such information.

As a summary, the document used in testing, as well as defect detection activities in general, is highly diverse and the weight of test case documentation seems to be over-emphasized in the literature. *Our results highlight the importance of researching the effective use of various information sources in defect detection activities. Especially the use of requirements documentation and communication mechanisms other than documents to support testing and defect detection are important research areas.*

4.4. RQ4 Knowledge

The fourth research question was: "What are the shares of defects found with different test oracles?" We found that the source of a software-testing oracle, i.e., knowledge about whether the

software works correctly or not, was general knowledge (28%), system knowledge (27%), customer knowledge (21%), or domain knowledge (16%). We found that in only 7% of the defects, the oracle was a document indicating the correct result. This finding contradicts the traditional wisdom that prescribed expected results are a crucial part of the test documentation. If we compare this to the results regarding the document usage in the previous research question, we see that 84% of defect detection activities involved some kind of documentation, but only 7% involved a documented oracle. This might seem conflicting, but we think that it indicates that documents have an important role in detecting defects, but do not often explicate the correct result in practice. For example:

- A user manual indicates how a certain feature works, but it cannot possibly indicate all of the ways the feature can fail.
- Some failures are related to how the entire system functions, e.g., if a feature X works in a certain way, then feature Y should also work in the same way.
- Some failures only become apparent when one understands the users' work and goals, as presented in more detail in our previous work [11].

Thus, documents provide valuable input to defect detection, but they are not solely comprehensive and do not contain all of the answers and instructions. These results extend our earlier study, where we studied the knowledge types used in software testing from video-recorded test sessions [11]. In that research, we described an extensive use of personal knowledge for defect detection and found that 20% of the defects were so called *windfall* defects, i.e., revealed in features that were not the primary target of the testing session in question. For these type of defects, the testers cannot have any prepared documentation on the expected test results, but they still reveal defects based on their knowledge and experience. This is one phenomenon that explains the low contribution of documented oracles in manual testing. Similarly, in related work it has been found that software test teams also find defects in components that they are not testing [47,48].

The results of this paper emphasize the role of personal knowledge in defect detection and extend our earlier field study [11] through a larger set of survey data. *These results reveled the high reliance on personal knowledge as a test oracle, even though a variety of documentation was used to support defect detection in general.* Future research should find ways of utilizing personal knowledge in defect detection, study how to support building relevant knowledge in testing organizations, and study how to utilize the knowledge outside of testing organizations in defect detection.

4.5. Effects of the context and generalizability of the results

Importance of the context has been widely discussed in software engineering research, as such contextualization increases understanding about the generalizability of the results. Whereas past work highlights the importance of specifying as much as possible of the context [49–51], more recent work by Dybå et al. [52] points out that such an approach leads to a context space that has more combinations than there are atoms in the universe. Thus, Dybå et al. [52] suggest that authors focus, instead, on the context that can improve the development of theories and explain the phenomena, its constraints and opportunities. Following the suggestions of Dybå et al., we next discuss three context variables that we think can explain the results, and we also provide analytical reasoning as to why they are likely to do so. The three important context variables in our cases were: a rich graphical user interface (GUI); a small number of technically different solutions offered over time; and a relatively low separation of the testing organization.

We need to point out that the authors' understanding of these companies is much deeper than the survey results presented in this paper, as we have studied these organizations in our prior works as well [10,31–33]. This discussion of context also acts as an initial proposal for theory explaining the software defect detection and testing in software organizations (see Table 16 for a summary). Naturally, many future works are needed with a much larger set of companies to see if our reasoning, based on our understanding of the case companies, actually leads to an empirically supported theory.

4.5.1. Human as a user—GUI or no GUI

The first important context variable is the role of the humans as users: Is the software meant to be used by humans directly through a GUI or is the software used indirectly as part of a larger system? An example of direct use would be a calendar application, or an application that is used to draw blue prints for engineering products, such as airplanes. Examples where humans are indirectly using the software could include ABS-brakes or telecom switches.

We think that direct human access through a GUI has a big impact on how software testing is done and what the share of implicit defect detection and contribution of non-testers is. This factor affects our results concerning the responsibilities and activities (RQ1 and RQ2). When software has a GUI, it gives primary access to the software to all people who work with the product. Most of the people in the organization must also be able to use the product. For example, sales and customer consultants must understand how to use the application in order to sell it and in order to train the users of the software. When people in various roles can use the product, it also means that they can test it, i.e., they can find defects and assess the quality.

When there is no direct human access to the software, this all changes. For example, sales people cannot advertise the features of the product with GUI demonstrations, but they can highlight the performance, standard compliance, reliability, and the costs of a telecom switch, for example. When no GUI is present, people in various roles still have an understanding about the product, but since they cannot really use the product, they cannot really test it, either. Thus, we think that the amount of explicit defect detection and the involvement of testers are likely to increase when humans have no, or only limited, access to the features of the product.

4.5.2. Small number of technically different solutions leads to knowledge accumulation and shared interest

The second context variable is the number of technically different solutions the unit offers over time. We think that if an organizational unit focuses only on one, or a few, solutions over a longer period of time, this leads to *accumulated domain knowledge* and *shared interest* in improving the solution. In all our organizational units, only one or two technical solutions was offered, and we think this explains our results regarding the responsibilities (RQ1), document use (RQ3), and knowledge use (RQ4). We purposefully use the term solution as we think that it does not matter whether the software solution is a COTS type of software product, or a long-lived bespoke software system, e.g., a governmental pension calculation system[2]—in both cases, the people working in the organization would accumulate knowledge of the solution, and fur-

[2] Our first impression was that COTS vs. bespoke would be an explaining factor in our results. Then we talked about the knowledge accumulation with a practitioner that worked in a company that provided a single tailor-made solution for calculating pensions to a large governmental organization, and he mentioned that they had similar findings. This discussion led to the refinement of this context factor.

Table 16
Most important contextual factors.

Context variable	RQs affected	Explanation
GUI	RQ1, RQ2	GUI allows humans without technical skills to use the software and find defects during several activities of software development
Small number of solutions	RQ1, RQ3, RQ4	Offering a small number of solutions creates shared interest to the solutions and allows the knowledge accumulation to occur. This makes it easier and more valuable for the company to have several roles participate in explicit and implicit defect detection
Separate test organization	RQ1, RQ2, RQ3	A separate testing organization would increase the amount of testers, explicit defect detection, and formal test documentation

thermore, would share an interest to the solution the company is offering.

The knowledge accumulation of the solution by the personnel explains our results, because staff members with a long history with the product can find relevant defects by using their knowledge of the product, its usage scenarios, users, and history. In an opposite case, where a software company would offer several types of solutions and short term projects, there would be much less knowledge accumulation as the software, its customers, and project teams would change between projects and products. Thus, we think that the knowledge accumulation has an effect on the amount of personal knowledge used in detecting defects. Consequently, also the need for documentation is affected, as highly knowledgeable personnel would need less-detailed documentation. We think this can explain why test cases were seldom used as knowledgeable staff could find defects without such detailed instruction, only using other types of documents.

The shared interest of the solution the company is building also explains our results. Although the internal competition of resources, even in our cases, can make people favor their own project, there is still a shared interest in the solutions since the staff understand that the company's success is ultimately tied to a success of the core solution. This can explain why different roles participate in the testing effort, the small share of testers, and explicit testing. In an opposite case, where a company has several solutions that are not related to each other, the interest to detect and report defects over project and product boundaries might be lower. This would be especially true when a person would be required to work with a solution one is not familiar with. This way the number of solutions over time affects the shared interest and accumulated knowledge of the software.

Manager of organization B2 (the largest one of our cases) pointed out that the size of their organization and software has led to a situation where knowledge accumulation still happens but different parts of the organization learn about different things. Thus, knowledge of certain topics varies among organizational units. The manager pointed out that this unbalanced knowledge distribution has to be taken into account when planning which individuals should participate in testing.

4.5.3. Organization of testing

Finally, the way testing is organized undoubtedly affects our results. Kit and Finzi [53] presents seven possible ways to organize testing starting from lightweight, "testing is each developer's responsibility" to heavyweight "centralized test organization with technology center." Kit highlights that separate testing organization or team provides testers with test processes, standards, policies, tools, training, and measures. In other words, the separation of testing increases testing knowledge among testers. On the other hand, having a separate organization for testing breaks the communication and flow of knowledge from development to testing, and this is even amplified more if the testing is globally distributed [45]. Our cases highly valued testers' domain knowledge rather than their testing knowledge [10]. Our cases (B1 and C) were partly hesitant in having a separate testing organization or team as it might have reduced the flow of domain knowledge, and they wanted to see testing as a team effort in order to make sure that quality is every employee's responsibility. Case B2 had started to create a separate testing to increase their baseline quality and to have better management of the testing activities. As case B2 was the largest organization, they had the highest need for such a separate testing organization, yet they still highlighted the idea that quality is everyone's responsibility. Thus, the company applied team effort of testing and supported it with separate testing team. Case A had had a separate testing team for years for the testing of their internal monthly release, but even in case A, each project also performed testing on the parts they had created on top of the common core. This is visible in the results as case A had a higher share of explicit testing and a higher number of defects found by testers (see Table 7).

A separate testing organization would reduce the number of roles searching and reporting defects, thus affecting our results regarding responsibilities (RQ1). A separate testing organization would also increase the defects found by explicit defect detection, affecting our results regarding activities (RQ2) due to two reasons. First, there would be more explicit defect detection; and, second, people in other roles might care less about the product quality and make less efforts to find and report defects, i.e., relying on the safety net of the testing organization. This is also supported by Grechanik et al. [45], who states that when a separate testing organization is present, developers typically do not perform unit testing, which leads to many shallow defects that waste the test organizations' resources and prevent finding more relevant or subtle defects. Furthermore, a separate testing organization would most likely use more test-case-based approaches to defect detection, affecting our results regarding documentation use (RQ3).

There are factors explaining why some companies have a separate testing organization and others, such as many of the ones we studied, have purposefully decided not to have one. Those reasons further complicate the list of relevant context factors; however, two of them must be mentioned. First, we think that if the required quality is very high, then it is more likely that there is a separate testing organization or team. For example, from a case study of the telecom industry [16], we can find that system integration testing needed over 300 h (almost two person months) of effort to find a single defect on average. For telecom, such a high level of quality and associated costs might be needed, but for the organizations we studied, it would be much too expensive. The second factor explaining the existence of a separate testing organization might be the number of people working in the company or the number of people working for a single solution. In a larger company, it would be easier to argue why separate testing organization would be needed.

4.6. Future works

This section presents four avenues for future work. First, one should study the effects of contextual factors presented in Section 4.5 and in summarized Table 16 in varying contexts to validate our hypotheses of the context factors and their relationship to soft-

ware defect detection. This could be done by submitting the survey questions of this work with additional questions on the context variables to a large number of companies. Then a statistical relationship could be established, and we could see whether the context variables can explain the results and what is the strength of the relationship, e.g., is GUI a more prominent factor than a separate testing organization in explaining the number of responsibilities finding defects.

Second, we think more future studies should be devoted toward implicit defect detection, as past work has mainly focused on explicit defect detection. Such work can have many forms, ranging from controlled experiments to case and other real world observation studies. Experiments could be used to study how is defect detection affected if there are other goals in addition to defect detection, e.g., how is manual testing affected if one has to create training material for a certain feature while searching defects. Case studies could be used to understand implicit defect detection in the software industry. As an example of implicit defect detection, Microsoft has used informal defect detection by the internal use of their products, known as dogfooding or "eating your own dog food," for 20 years [15], and beta testing has been a common industry practice for decades [13]. Still, the academic studies of the implicit defect detection are extremely rare, and the limited knowledge we could find in this area was presented in Table 15. Future research should also conduct a systematic literature review of this topic, although the lack of common terminology is likely to make it challenging.

Third, documentation use during manual defect detection should be studied in more detail. Based on this study, we only know that several documents are used when detecting defects from the software (see Table 11). However, we have no detailed knowledge on how the documents are used to support defect detection. Perhaps there would be ways to improve, for example, requirement documents so that they would be better directly suitable for defect detection.

Fourth, improvements in tools or training would benefit non-testers and implicit defect detection in general. A recently developed tool is found to be particularly useful for non-testers [20]. Additionally, all systems that automatically, or semi-automatically, collect crash[3] and error data can be viewed as tools that help implicit defect detection. Prior works have presented successes in analyzing such data [54]. Nevertheless, we think there is still plenty of work to be done to harness the full potential of non-testers and implicit defect detection.

5. Conclusions

This paper makes four main contributions. First, we classify software defect detection either explicit (software testing and review activities) or implicit defect detection. In both explicit and implicit defect detection, defects are detected and software quality is evaluated. However, only in explicit defect detection are the defect detection and software quality evaluation the primary goals of the activity. In explicit defect detection, we found that testing revealed roughly ten times more defects than reviews. In implicit defect detection, the defect detection and quality evaluation happen implicitly while performing an activity with another goal, e.g., preparing for a product demonstration. We studied explicit and implicit defect detection by asking for the activities wherein the personnel of four software development organizations detected defects. We found that implicit defect detection revealed a surprisingly large share of the defects (over 60%). Most notable activities finding defects outside of explicit software defect detection were programming, helpdesk, and internal usage of the product. As the contribution of implicit detect detection can be high due to large volume of such activities in comparison to explicit defect detection activities, we suggest that implicit defect detection activities make a good target for efficient improvements in overall defect detection. The existing knowledge on implicit defect detection is summarized in Table 15.

Second, we confirm previous findings [10,19,20,23] that defect detection and testing is a cross-cutting activity involving multiple roles in the companies we studied. Participation in explicit software defect detection activity comes from a wide spectrum of roles and the contribution of non-testers is important. Combining the role and activity dimensions of software defect detection is illustrated as the defect detection activity quadrants in Fig. 1, and the illustrative defect detection shares for each quadrant in the surveyed companies are presented in Table 1. We think that the defect detection activity quadrants can help in understanding the multifaceted phenomenon of software defect detection.

Third, we studied the role of documentation and knowledge in software defect detection and found that the share of defects detected with test cases was low (less than 20%). However, the lack of test case usage seemed to lead in using other types of documents and information sources in defect detection, such as product requirements, product manual, and information from defect database and customer request trackers. Defect detection completely without documents had a 16% share. Thus, the future studies of manual software testing need to also focus on the quality and usage of other documents than test cases. Finally, we asked about the oracle used when detecting defects and found that the tester's knowledge was applied very frequently as a test oracle while using a document as an oracle had a 7% share, indicating that documents are used as oracles much more rarely than for supporting defect detection in other ways.

Fourth, we put forward and discuss three context factors: GUI, number of solutions offered, and organization of testing, which we think explains our results. We think that our results can be generalized to other companies or organizational units given that their product has a rich GUI, they offer only a few technical solutions with a long lifecycle to their customers, and they do not have a substantial, separate testing organization (see Section 4.5 and Table 16). For example, if the software does not have a user interface, e.g., ABS-brakes, then we think that a far greater share of defects would be detected with explicit defect detection than in our case companies. Future studies with a larger set of companies are needed to establish whether these context variables can explain the roles participating in defect detection, shares of defects found by implicit defect detection activities, the amount and types of test documentation used, and the type of knowledge applied when recognizing defects.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.infsof.2013.12.005.

References

[1] L. Briand, Embracing the engineering side of software engineering, Softw. IEEE 29 (4) (2012). pp. 96-96.
[2] R.L. Glass, R. Collard, A. Bertolino, J. Bach, C. Kaner, Software testing and industry needs, Softw. IEEE 23 (4) (2006) 55–57.
[3] D. Martin, J. Rooksby, M. Rouncefield, I. Sommerville, 'Good' organisational reasons for 'Bad' software testing: an ethnographic study of testing in a small software company, in: Proceedings of the 29th International Conference on Software Engineering, 2007, pp. 602–611.

[3] For example, Windows Error Reporting http://msdn.microsoft.com/en-us/library/windows/hardware/gg487440.aspx and Google Breakpad http://code.google.com/p/google-breakpad/.

[4] C. Andersson, P. Runeson, Verification and validation in industry – a qualitative survey on the state of practice, in: Proceedings of the 2002 International Symposium on Empirical Software Engineering, 2002, pp. 37–47.

[5] M. Greiler, A. van Deursen, M. Storey, Test confessions: a study of testing practices for plug-in systems, in: 34th International Conference on Software Engineering (ICSE), 2012, pp. 244–254.

[6] M.V. Mäntylä, C. Lassenius, What types of defects are really discovered in code reviews?, IEEE Trans Software Eng. 35 (3) (2009) 430–448.

[7] O. Laitenberger, M. Leszak, D. Stoll, K. El Emam, Quantitative modeling of software reviews in an industrial setting, in: Proceedings of the Sixth International Software Metrics Symposium, 1999, pp. 312–322.

[8] M. Ciolkowski, O. Laitenberger, S. Biffl, Software reviews, the state of the practice, Softw. IEEE 20 (6) (2003) 46–51.

[9] H. Siy, L. Votta, Does the modern code inspection have value?, Int Conf Softw Mainten (2001) 281–289

[10] M.V. Mäntylä, J. Itkonen, J. Iivonen, Who tested my software? Testing as an organizationally cross-cutting activity, Software Qual. J. 20 (1) (2012) 145–172.

[11] J. Itkonen, M.V. Mäntylä, C. Lassenius, The role of the tester's knowledge in exploratory software testing, IEEE Trans. Software Eng. 39 (3) (2013) 707–724.

[12] B.D. Klein, D.L. Goodhue, G.B. Davis, Can humans detect errors in data? Impact of base rates, incentives, and goals, MIS Quart. (1997) 169–194.

[13] R.J. Dolan, J.M. Matthews, Maximizing the utility of customer product testing: beta test design and management, J. Prod. Innovation Manage. 10 (4) (1993) 318–330.

[14] W. Harrison, Eating your own dog food, Softw. IEEE 23 (3) (2006) 5–7.

[15] M.A. Cusumano, R.W. Selby, Microsoft Secrets, The Free Press, USA, 1995.

[16] T. Berling, T. Thelin, An industrial case study of the verification and validation activities, in: Software Metrics Symposium, 2003, Proceedings. Ninth International, 2003, pp. 226–238.

[17] C. Jones, Software defect-removal efficiency, Computer 29 (4) (1996) 94–95.

[18] N.D. Fogelström, T. Gorschek, Test-case driven versus checklist-based inspections of software requirements – an experimental evaluation, in Workshop em Engenharia de Requisitos (WER 07), 2007, pp. 116–126.

[19] A. Kasoju, K. Petersen, M.V. Mäntylä, Analyzing an automotive testing process with evidence-based software engineering, Inf. Softw. Technol. 55 (7) (2013) 1237–1259.

[20] J. Mahmud, A. Cypher, E. Haber, T. Lau, Design and industrial evaluation of a tool supporting semi-automated website testing, Softw. Test. Verif. Reliab. 24 (1) (2014) 61–82.

[21] J.J. Ahonen, T. Junttila, M. Sakkinen, Impacts of the organizational model on testing: three industrial cases, Empirical Softw. Eng. 9 (4) (2004) 275–296.

[22] J. Itkonen, K. Rautiainen, Exploratory testing: a multiple case study, in: Proceedings of the International Symposium on Empirical Software Engineering, 2005, pp. 84–93.

[23] J. Rooksby, M. Rouncefield, I. Sommerville, Testing in the wild: the social and organisational dimensions of real world practice, Computer Support. Cooperat. Work (CSCW) 18 (5) (2009) 559–580.

[24] J. Itkonen, M.V. Mäntylä, Are test cases really needed in manual software testing? – Replicated comparison between exploratory and test-case-based testing, Empirical Softw. Eng., 2013. http://dx.doi.org/10.1007/s10664-013-9266-8.

[25] J. Itkonen, M.V. Mäntylä, C. Lassenius, Defect detection efficiency: test case based vs. exploratory testing, Empirical Software Engineering and Measurement, 2007, ESEM 2007, First International Symposium on, 2007, pp. 61–70.

[26] A. Beer, R. Ramler, The role of experience in software testing practice, in: Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications, 2008, pp. 258–265.

[27] P. Poon, T. Tse, S. Tang, F. Kuo, Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing, Software Qual. J. 19 (1) (2011) 141–163.

[28] R. Merkel, T. Kanij, Does the individual matter in software testing?, Swinburne University of Technology, Centre for Software Analysis and Testing, Technical, Report, vol. 1, 2010.

[29] A. Folstad, B.C.D. Anda, D.I.K. Sjoberg, The usability inspection performance of work-domain experts: an empirical study, Interact. Comput. 22 (2) (2010) 75–87.

[30] D.F. Galletta, D. Abraham, M. El Louadi, W. Lekse, Y.A. Pollalis, J.L. Sampler, An empirical study of spreadsheet error-finding performance, Account., Manage. Inform. Technol. 3 (2) (1993) 79–95.

[31] M.V. Mäntylä, J. Vanhanen, Software deployment activities and challenges – a case study of four software product companies, in: 15th European Conference on Software Maintenance and Reengineering (CSMR), 2011, pp. 131–140.

[32] T.O.A. Lehtinen, M.V. Mäntylä, J. Vanhanen, Development and evaluation of a lightweight root cause analysis method (ARCA method) – field studies at four software companies, Inf. Softw. Technol. 53 (10) (2011) 1045–1061.

[33] J. Vanhanen, M.V. Mäntylä, J. Itkonen, Lightweight elicitation and analysis of software product quality goals – a multiple industrial case study, in: Proceedings of the third International Workshop on Software Product Management (IWSPM), 2009, pp. 42–52.

[34] J. Itkonen, M.V. Mäntylä, C. Lassenius, How do testers do it? An exploratory study on manual testing practices, in: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 494–497.

[35] M. Komssi, M. Kauppinen, M. Pyhäjärvi, J. Talvio, T. Männistö, Persuading software development teams to document inspections: success factors and challenges in practice, 2010 18th IEEE International Requirements Engineering Conference, 2010, pp. 283–288.

[36] P.N. Robillard, T. Francois-Brosseau, Saying, I am testing, is enough to improve the product: an empirical study, in: Proceedings of the International Multi-Conference on Computing in the Global Information Technology (ICCGI'07), 2007, pp. 5.

[37] V.R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, M.V. Zelkowitz, The empirical investigation of perspective-based reading, Empirical Softw. Eng. 1 (2) (1996) 133–164.

[38] L.C. Briand, A critical analysis of empirical research in software testing, Empirical Software Engineering and Measurement, 2007 (ESEM 2007), First International Symposium on, 2007, pp. 1–8.

[39] E. Engström, P. Runeson, Test overlay in an emerging software product line–an industrial case study, Inf. Softw. Technol. 55 (3) (2013) 581–594.

[40] E.J. Uusitalo, M. Komssi, M. Kauppinen, A.M. Davis, Linking requirements and testing in practice, in: 16th IEEE International Conference, Requirements Engineering (RE'08), 2008, pp. 265–270.

[41] Z.A. Barmi, A.H. Ebrahimi, R. Feldt, Alignment of requirements specification and testing: a systematic mapping study, in: Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2011, pp. 476–485.

[42] G. Sabaliauskaite, A. Loconsole, E. Engström, M. Unterkalmsteiner, B. Regnell, P. Runeson, T. Gorschek, R. Feldt, Challenges in aligning requirements engineering and verification in a large-scale industrial context, Requirements Eng.: Found. Softw. Qual. (2010) 128–142.

[43] H. Post, C. Sinz, F. Merz, T. Gorges, T. Kropf, Linking functional requirements and software verification, in: 17th IEEE International Conference on Requirements, Engineering Conference (RE'09), 2009, pp. 295–302.

[44] F. Lanubile, C. Ebert, R. Prikladnicki, A. Vizcaíno, Collaboration tools for global software engineering, Softw. IEEE 27 (2) (2010) 52–55.

[45] M. Grechanik, J.A. Jones, A. Orso, A. van der Hoek, Bridging gaps between developers and testers in globally-distributed software development, in: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, 2010, pp. 149–154.

[46] D. Ma, D. Schuler, T. Zimmermann, J. Sillito, Expert recommendation with usage expertise, in: International Conference on Software Maintenance (ICSM), 2009, pp. 535–538.

[47] P. Jalote, R. Munshi, T. Probsting, The When–Who–How analysis of defects for improving the quality control process, J. Syst. Softw. 80 (4) (2007) 584–589.

[48] C. Andersson, P. Runeson, Investigating Test Teams' Defect Detection in Function test, in: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM), 2007, pp. 458–460.

[49] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734.

[50] P. Clarke, R.V. O'Connor, The situational factors that affect the software development process: towards a comprehensive reference framework, Inf. Softw. Technol. 54 (4) (2012) 433–447.

[51] K. Petersen, C. Wohlin, Context in industrial software engineering research, in: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 401–404.

[52] T. Dybå, D.I.K. Sjøberg, D.S. Cruzes, What works for whom, where, when, and why?: On the role of context in empirical software engineering, in: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2012, pp. 19–28.

[53] E. Kit, S. Finzi, Software Testing in the Real World: Improving the Process, ACM Press/Addison-Wesley Publishing Co., 1995.

[54] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, G. Hunt, Debugging in the (very) large: ten years of implementation and experience, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 103–116.