UNIVERSITY OF HELSINKI

https://helda.helsinki.fi

Analyzing the Relationships between Inspections and Testing to Provide a Software Testing Focus

Elberzhager, Frank; Münch, Jürgen; Assmann, Danilo

2014
Elsevier B.V.
http://hdl.handle.net/10138/153167
Elberzhager, F, Münch, J & Assmann, D 2014, 'Analyzing the Relationships between Inspections and Testing to Provide a Software Testing Focus', Information and Software Technology, vol. 56, no. 7, pp. 793–806. https://doi.org/10.1016/j.infsof.2014.02.007

Downloaded from Helda, University of Helsinki institutional repository. https://helda.helsinki.fi
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.
Please cite the original version.

This is an author-generated version.

The final publication is available at
http://dx.doi.org/10.1016/j.infsof.2014.02.007

Bibliographic information:

Frank Elberzhager, Jürgen Münch, Danilo Assmann. Analyzing the Relationships between Inspections and Testing to Provide a Software Testing Focus, Information and Software Technology (IST), vol. 56, no. 7, pp. 793–806, 2014.

Copyright © 2014 Elsevier B.V.

The published article can be found here: http://dx.doi.org/10.1016/j.infsof.2014.02.007

Analyzing the Relationships between Inspections and Testing to Provide a Software Testing Focus

Frank Elberzhager

Fraunhofer Institute for Experimental Software Engineering (IESE)

Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

Email: frank.elberzhager@iese.fraunhofer.de

Jürgen Münch

University of Helsinki

P.O. Box 68 (Gustaf Hällströmin katu 2b), 00014 Helsinki, Finland

Email: juergen.muench@cs.helsinki.fi

Danilo Assmann

Vector Informatik GmbH

Ingersheimer Straße 24, 70499 Stuttgart, Germany

Email: danilo.assmann@vector.com

Context: Quality assurance effort, especially testing effort, is frequently a major cost factor during software development. Consequently, one major goal is often to reduce testing effort. One promising way to improve the effectiveness and efficiency of software quality assurance is the use of data from early defect detection activities to provide a software testing focus. Studies indicate that using a combination of early defect data and other product data to focus testing activities outperforms the use of other product data only. One of the key challenges is that the use of data from early defect detection activities (such as inspections) to focus testing requires a thorough understanding of the relationships between these early defect detection activities and testing. An aggravating factor is that these relationships are highly context-specific and need to be evaluated for concrete environments.

Objective: The underlying goal of this paper is to help companies get a better understanding of these relationships for their own environment, and to provide them with a methodology for finding relationships in their own environments.

Method: This article compares three different strategies for evaluating assumed relationships between inspections and testing. We compare a confidence counter, different quality classes, and the F-measure including precision and recall.

Results: One result of this case-study-based comparison is that evaluations based on the aggregated F-measures are more suitable for industry environments than evaluations based on a confidence counter. Moreover, they provide more detailed insights about the validity of the relationships.

Conclusion: We have confirmed that inspection results are suitable data for controlling testing activities. Evaluated knowledge about relationships between inspections and testing can be used in the integrated inspection and testing approach In^{2}Test to focus testing activities. Product data can be used in addition. However, the assumptions have to be evaluated in each new context.

Keywords: Inspection, testing, integration, precision, recall, F-measure, case study

1. Introduction

Verification and validation are an indispensable part of modern software development. The effort and costs for performing such quality assurance are often rather high; testing, in particular, can consume 50% or more of the overall effort in certain environments [1, 2, 3] Still, the resulting quality of the software is often poor, i.e., many defects are found by customers after the release of the software. This can lead to high rework costs or even entail risks for human beings in safety-critical environments.

Different approaches for optimizing quality assurance exist [4]. One of these approaches is automation. For instance, deriving test cases or performing testing could be automated to save time and execute more test cases than with a manual procedure. Another approach is to focus testing activities on those parts that are expected to be defect-prone. To do this, different metrics are usually considered and relationships between such metrics and defect-proneness are exploited. For example, if a correlation between large modules and defect-prone parts were to be shown in a certain environment, such knowledge could be used in future development cycles to allocate test effort to particularly large modules. Common metrics are product metrics (e.g., size, complexity) or process metrics (e.g., number of developers per module, number of changes per module). Typically, metrics applied to defect information that is available early, e.g., from inspections or reviews, are currently not used to focus testing activities.

As static quality assurance techniques such as inspections and reviews, and dynamic quality assurance techniques such as testing typically have the same objective – finding defects, and thus improving the software – there is a basis for a joint application, and it seems promising to also consider data from inspections to provide a software testing focus. We have proposed the integrated inspection and testing approach In^{2}Test [5], which explicitly uses inspection defect data to focus testing activities. In^{2}Test uses so-called selection rules, which define a software testing focus by considering different metrics. This approach has to be calibrated first, which requires knowledge about the relationships between inspections and testing for the context (i.e., the development environment) at hand. Such kind of knowledge is typically not available [6]. Therefore, the In^{2}Test approach starts with assumptions about the relationships between inspections and testing and evaluates them. There are different possibilities for doing such an evaluation. In this article, we compare three different evaluation procedures (i.e., confidence,

quality classes, and F-measure including precision and recall) in order to analyze their suitability.

Earlier studies have shown that the In^{2}Test approach is applicable and more efficient than a non-integrated approach (within the context and limits of the studies) [5], [7]. We use the term non-integrated approach to refer to all approaches that combine inspections and testing such that these activities are performed independently from each other. The results from these studies have shown that using early defect data to focus test activities is slightly more efficient than considering only selected product data to focus test activities. In addition, a combination of early defect data and product data saved significantly more effort than using only product data [8]. However, we considered only an informal evaluation scheme, which offers only low significance when deriving long-term conclusions. Consequently, we extend and detail our analysis in this article, concentrating on the following overall research question:

> Which evaluation procedure leads to the highest validity of selection rules for focusing testing based on inspection defect data?

With the analysis presented in this article, we directly answer the overall research question and substantiate the results from our earlier studies with more detailed analyses. The validity of our analysis results is limited to the scope of the presented case studies and the resulting knowledge for focusing testing cannot be treated as universally applicable rules without new evaluations in other environments. However, this article shows a procedure for analyzing inspection data in different contexts to find valid rules for one's own context and, as a consequence, for optimizing quality assurance in a specific context. When presenting our results and the approach, we motivate that it is worth checking inspection and testing defect data and corresponding rules in industrial environments in order to further optimize testing activities.

The remainder of this article is structured as follows: Section 2 reports on related work regarding approaches that aim at providing a testing focus, while Section 3 presents the In^{2}Test approach at a glance, including the three different evaluation procedures. Section 4 presents results from two case studies from different environments where the calibration of the approach was performed and where the different evaluation procedures were applied. Section 5 points out the main lessons learned. Finally, Section 6 concludes this article with a summary and an outlook on future work.

2. Related work: Focusing testing activities

Various approaches for focusing testing activities have been proposed. One very popular and established approach is the use of different kinds of metrics for predicting defect-proneness. The idea is that certain correlations exist between metrics and defects in a system. If such relationships are found, they can be used to predict parts where defects are expected, and testing can be focused on such areas. One typical differentiation of data used to calculate such metrics is to distinguish product data, process data, and historical defect data. A lot of different studies have been performed to find such relationships. The first metrics that were considered were size or complexity. While in many studies, correlations could be found that led to suitable predictions of defect-prone parts, these findings were highly context-dependent. Some studies showed, for instance, that large modules contain more defects than small ones, while others showed the opposite [9], [10]. D'Ambros et al. [11] present an overview of several metrics and provide an extensive comparison of those metrics. They found that a combination of different metrics turned out to be best in their contexts, but mentioned that those metrics might be of lower quality in other environments. Arisholm et al.

confirmed these conclusions [12]. However, defect data from early quality assurance activities such as reviews or inspections are typically not considered when predicting defect-proneness in order to focus testing activities. Also, knowledge about the relationships between inspections and testing is very limited [6].

Another approach for focusing testing is to consider expert knowledge. People with a lot of experience in a certain environment usually know their software very well, and quality engineers often know intuitively where more problems might occur. For example, Nasser et al. [13] describe a knowledge-based approach for test case generation. Knowledge and experiences from inspection experts are sometimes used for defect number predictions to control inspection activities [14], but are typically not used to focus testing. In addition, expert-based approaches significantly depend on the experience level of the appropriate experts as well as on their availability (which is often an issue in practical situations). Furthermore, risk-based approaches can be considered to focus testing [15].

Hybrid approaches are another way to focus testing by combining expert knowledge and data. The Hydeep approach, for example, is able to predict effectiveness and defect values [16]. Such data can be used to focus testing on those parts where the highest number of defects is expected. Fenton et al. [17] propose using Bayesian nets to predict defect numbers, which can be used to decide when to stop testing, but also contribute to the identification of defect-prone parts that should be in focus during testing activities. However, inspection data are typically not used for focusing testing.

3. Background: The integrated inspection and testing approach In^{2}Test

3.1 The In^{2}Test approach at a glance

The basic idea of the integrated inspection and testing approach is to use inspection defect data to predict defect-prone parts or defect types for testing that are likely to appear during testing.

The process starts with an inspection (step 1). To remain flexible, no specific inspection technique is required; for example, a formal inspection, a team review, or a more informal peer desk-check could be used to find defects. Different kinds of inspection metrics can be applied, such as number of defects per part or defect density per part, or a defect classification can be used to classify each defect found.

As inspection is a manual activity and the result of an inspection depends on several context factors such as the experience of the inspectors, the kind of software product, or the inspection process, quality monitoring of the inspection process itself has to be conducted (step 2). This allows ensuring that the inspection results are reasonable and that one can rely on them, i.e., that the results are consistent with the context factors. For instance, if experienced inspectors perform an inspection of newly developed software, a certain number of defects are expected to be found. During quality monitoring, the reading rate of an inspection or the number of defects can be considered for monitoring. Ideally, historical context-specific data is available for comparison. If this is not the case, data from the literature can initially be taken (e.g., [18]).

Obviously, the more thoroughly an inspection is performed (i.e., the higher the process conformance) and the more defects are found, the more trustworthy is the basis for the following steps. Usually, a rather formal inspection process will lead to more valid results, but an informal inspection process applied by inspection experts can also lead to trustworthy and good results. This means that the inspection process, i.e., the inspection technique used and its specific customization, needs to be taken into account and that a sound analysis of the

outcomes has to be performed. If a company does not have much experience with inspections, we recommend starting with a more formal inspection process to ensure valid results. However, the approach does not dictate any specific inspection process, but rather considers the applied inspection process as part of the context. In summary, the quality monitoring checks the degree of process conformance.

The next step is the prioritization of the system parts and/or defect types (step 3). For this, the inspection data is used, and product and historical data can optionally be used in combination in order to improve this prioritization. Only system parts or defect types can be prioritized (this is the so-called 1-stage approach), or a combination of both can be considered (i.e., system parts such as code classes are prioritized, and in those code classes, certain defect types are given particular consideration; 2-stage approach). In order to be able to perform this prioritization, knowledge about the relationships between inspections and testing is necessary. For example, if we know that in our context, more defects are expected to be found during testing in parts where many inspection defects are found, we can focus testing on these parts. Of course, other relationships might be true in different environments.

A known relation can be refined into a so-called selection rule in order to make it operational. An example of an instantiated selection rule for the code level could be: "Focus testing on those code classes where the inspection found more than 25 defects per thousand lines of code." In this case, a threshold is defined and all code classes whose defect density is higher than 0.025 are selected. Another example is: "Focus testing on 20% of the code classes that have the highest defect content." In this case, a sorted list of code classes is taken into account and the 20% top-prioritized code classes are selected. Unfortunately, valid knowledge about the relations between inspections and testing is often missing so that selection rules are based on context-specific assumptions about these relationships. Assumptions can be derived, for instance, based upon experience (i.e., based on experts' knowledge) or empirically (i.e., empirically valid knowledge can be adapted to the context at hand). Finally, the degree to which the selection rule has been evaluated for a specific context and the context itself are explicitly described in the so-called scope of validity. Examples of context are the experience of inspectors and testers, the inspection and testing process, the development process, or available resources.

Based on these context-specific assumptions and selection rules, the prioritized testing activity can be conducted (step 4). The concrete selection, respectively definition, of test cases and test exit criteria depends on the quality assurance strategy used. For example, if testing effort is to be saved, one reasonable strategy would be to focus 50% of the available test effort on the prioritized parts, put another 30% of the test effort on less prioritized parts, and save the remaining 20% of the effort. Finally, all data and experiences are packaged and stored in an experience database (step 5).

Until now, we have described a typical application case, which is summarized in Figure 1 (B). But how can we address the issue that most selection rules are typically based on assumptions (i.e., the relationships are unclear)? In this case, we have to calibrate the approach first, i.e., the assumptions need to be evaluated in the respective context. This can be done in the following way: After the inspection and its quality monitoring, a traditional testing activity (i.e., non-prioritized testing) is performed to gather test defect data. Then the assumptions are used to define selection rules. Once the selection rules have been defined, the evaluation can be done retrospectively because all inspection and test data are now available.

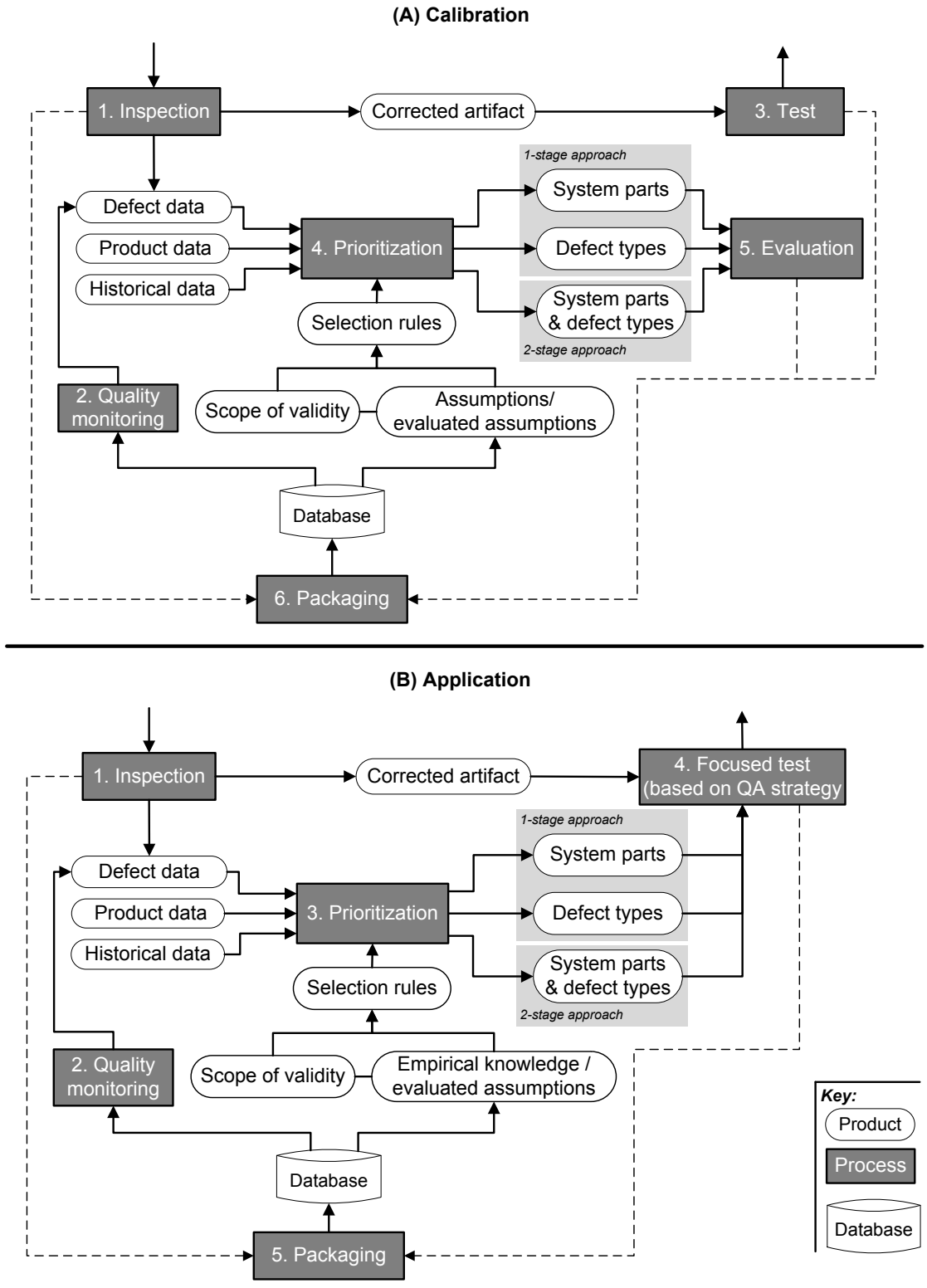


Figure 1. Calibration and application process for the In²Test approach

3.2 Three evaluation schemas

With respect to the evaluation, different possibilities exist. First, a *confidence counter* can be assigned to each selection rule. Initially, the value is zero. Each time the selection rule prioritizes correctly, the value is increased by one. Correct in this context means that all test defects are found with this prioritization, and no defect is overlooked. Second, a 4-scale

schema can be used that refines the way effectiveness is assessed. The four *quality classes* are:

- A: All parts in which test defects are found are prioritized, and parts in which no test defects are found are not prioritized.
- B: All parts in which test defects are found are prioritized; in addition, parts in which no test defects are found are also prioritized.
- C: Only some parts in which test defects are found are prioritized (includes also the prioritization of parts in which no test defects are found).
- D: No parts in which test defects are found are prioritized.

	Inspection defects (all)	Inspection defects (critical)	Inspection defect density	Size (lines of code	Mean method length (lines of code)	Test defects
Code class 1	14	1	0.06	231	3.28	0
Code class 2	40	0	0.03	1364	13.54	0
Code class 3	39	0	0.06	701	8.11	6
Code class 4	7	1	0.06	115	7	0

Assumptions and selection rules	Selection of code classes	Confidence counter	Quality class	Precision	Recall	F-measure
Assumption 1: Parts of the code where a large number of inspection defects are found indicate more defects to be found with testing.						
Selection rule: Focus testing on those code classes in which the inspection determined: defect content > 25.	2, 3	+1	B	0.5	1	0.67
Selection rule: Focus testing on those code classes in which the inspection determined: defect density > 0.05	1, 3, 4	+1	B	0.33	1	0.5
Assumption 2: Parts of the code where a large number of inspection defects are found and which are of large size indicate more defects to be found with testing.						
Selection rule: Focus testing on those code classes in which the inspection determined: defect density > 0.05 & class length > 500 LoC	3	+1	A	1	1	1
Assumption 3: Parts of the code where a large number of inspection defects are found and which are of small size indicate more defects to be found with testing.						
Selection rule: Focus testing on those code classes in which the inspection determined: critical severity (defect content > 0) & mean method length < 10	1, 4	+0	D	0	0	0

Figure 2. Example of evaluations of selection rules

Third, precision and recall values can be calculated and aggregated into the *F-measure* [12]. Precision refers to the classification of defect-proneness being done correctly, i.e., it is a measure of the quality of our prediction. The value is between zero and one; a value close to one means that most of the classified parts are really defect-prone. Precision is defined as *true positives / (true positives + false positives)*. Recall refers to the classification of defect-proneness being complete, i.e., it is a measure of the completeness of our prediction. The value is again between zero and one; a value close to one means that almost all defect-prone parts are prioritized. Recall is defined as *true positives / (true positives + false negatives)*. The F-measure is the harmonic mean of precision and recall, calculated as *2 * ((precision * recall) / (precision + recall))*.

Examples of the three evaluation schemas are shown in Figure 2. In the upper part of the figure, exemplary data is given for four code classes, which is needed for evaluating the selection rules. In the bottom part of the figure, three assumptions with corresponding selection rules are stated, and their evaluation result is shown. For instance, the first selection rule selects code classes two and three because inspection defect content is higher than 25 for these code classes. As code class 3 contains more defects that were found during testing, the confidence counter of this selection rule is increased by one. With respect to the quality class, it is grouped into category "B" because all defects are found, but another code class is selected that does not contain any defects. Finally, the calculated F-measure is 0.67, which is quite good. Compared to the second selection rule of assumption one, it can be observed that the same confidence and the same quality class are selected, but a different F-measure and a different precision value are calculated; thus, the third evaluation procedure indicates a more fine-grained analysis. Two more selection rules show further examples of the evaluations: one very good one and one very bad one.

We take the three evaluation schemas into account to judge the selection rules in the remainder of this article, and compare them based on data from two case studies.

4. Case studies

In the following, we use data from two case studies [8] to compare three different evaluation procedures for judging relationships in the context of the In^{2}Test approach. The first case study was conducted in a research environment, while the second one was conducted in an industrial setting. The defect data used in both case studies are the same as those used before, while the different analyses are the major new contribution presented in this article. This means that the analysis is done based on existing (i.e., historical) data to find relationships between inspections and testing; however, the overall goal is to be able to predict future defect-proneness in order to optimize testing based on inspection data. Besides context-specific results from two case studies, we provide a general procedure for analyzing inspection data in different environments to find one's own valid rules for controlling testing activities and thus for optimizing quality assurance.

The overall focus of the case studies, the basis, and the main contributions of this article are summarized in Figure 3.

Objective	**1. Comparison of:** Three evaluation procedures for judging selection rules: • Confidence counter • Quality classes A-D • Precision, recall, and F-measure	**2. Confirmation / Refinement of:** Relationships between: • Inspection defect data / product data • Test defect data
Scope	• In^{2}Test approach: Calibration (i.e., retrospective analysis) • Code level	
Basis	• Inspection and test defect data, and product data from two environments • Assumptions and selection rules from two environments	
Contribution	**1. Case Study:** • Analysis of F-measure, precision and recall on existing assumptions and selection rules • Comparison of three evaluation procedures ➢ Similar significance of three evaluation procedures ➢ Confirmation of existing relationships, refined view on relationships	
	2. Case Study: • Definition of new relationships • Analysis of all three different evaluation procedures (confidence counter; quality classes; precision, recall, and F-measure) • Comparison of three evaluation procedures ➢ Different significance of three evaluation procedures ➢ Confirmation of existing relationships, refined view on relationships, insights about new relationships	
	General: ➢ Providing procedure how to find relationships and how to evaluate selection rules	

Figure 3. Characterization of the case studies

4.1 Case study 1

4.1.1 Goal

The different evaluation strategies are compared in the context of a case study that was primarily aimed at analyzing the applicability of the In^{2}Test approach and its effort reduction potential, and at finding initial relationships between inspection and testing [7]. The main result was that the approach could be applied and that an effort reduction for testing (i.e., test definition and execution) of up to 34% was possible depending on the concrete assumptions and selection rules. In the first analysis of the case study, only a small set of assumptions and selection rules was used. Three assumptions and 32 selection rules were considered. This evaluation was extended to allow a more systematic comparison of the quality of product and inspection metrics for the prediction of defect-prone parts [8]. It could be shown that selection rules using inspection metrics alone are of similar quality as selection rules using product metrics, and that a combination of both led to even higher effort reductions in our context. However, in the first analysis, we only used the 4-scale evaluation schema introduced above, which allowed a rather coarse-grained analysis. Therefore, based on the overall question stated in the introduction, the two refined research questions (RQ) that are relevant for the scope of this article and that are answered by a more detailed analysis are:

RQ 1. Which of the evaluation procedures is most applicable and provides the highest validity of the selection rules for focusing testing?

RQ 2. Which new or refined relationships can be found when applying the detailed evaluations of the In^{2}Test approach, i.e., which inspection and product metrics lead to the best predictions for defect-prone parts that should be focused on during testing?

4.1.2 Context

The In^{2}Test approach was applied twice within six months during the development and quality assurance activities of a tool called JSeq. The inspection and test data gathered during these activities form the basis for our analysis.

The tool was developed in Java and provides users performing a sequence-based specification (SBS). An SBS is used to derive a formal test model from natural language requirements [23]. One main developer was responsible for the development of the tool. During the case study, the tool consisted of about 80 code classes, about 650 methods, and had about 8,500 lines of code. Based on feedback from the developer, four central classes were selected for the first inspection, comprising about 1,000 lines of code. Those classes were also tested afterwards together with some additional classes. In the second quality assurance run, four code classes were again selected for the inspection, comprising about 2,400 lines of code, which were tested afterwards, among others. The code classes were completely different due to continuous tool development.

The inspectors used a Fagan-like inspection process [19]. After the planning phase, an overview meeting was conducted to introduce the tool and the checklists to the inspectors. Each inspector performed individual defect detection using a focused checklist. Each issue found was classified and it was checked whether it was a real defect that had to be corrected. The developer then corrected the code classes. With regard to testing, the developer used equivalence partitioning and experience-based testing. No effort numbers were known for testing activities during the first quality assurance run due to continuous development, rapidly changing code, and an unsystematic test. The effort for testing during the second quality assurance run (test definition and execution) was eleven hours.

Most of the four inspectors had a lot of programming experience and some inspection experience. One inspector was more familiar with inspections, but had only limited programming knowledge. The tester had large programming experience and was not involved in the inspection (for more details about the inspection and testing activities performed, see [7]).

The assumptions and selection rules were systematically derived by two quality assurance engineers who were not involved in the inspection and testing activities. The main sources were the metrics that could be gathered during the quality assurance activities, and existing empirical knowledge from the literature that was adapted for the given context.

4.1.3 Design

In this section, we describe how we defined the selection rules and how we applied the three evaluation procedures.

As we had no context-specific knowledge about the relationships between inspections and testing from this environment in the beginning, we followed the calibration procedure of the In^{2}Test approach. This means that in both quality assurance runs, the inspection including quality monitoring was performed first (here, we considered reading rate and number of defects, which turned out to be valid). Afterwards, the traditional (i.e., unfocused) test was done to gather the data necessary for the evaluation. Next, the definition of assumptions and selection rules was done in order to be able to apply the three different evaluation procedures.

Table 1. Metrics gathered from the JSeq study

		Product			Inspection								Test
			LOC	complex.	defect content				defect density				
	Code class	LOC (all)	(method mean)	(McCabe mean)	minor	med.	major	all	minor	med.	major	all	defect content
QA run 1	I	469	4.42	1.61	17	7	2	26	0.04	0.01	0.00	0.06	3
	II	37	9.00	5.00	4	2	0	6	0.11	0.05	0.00	0.16	0
	III	275	7.43	2.14	16	8	3	27	0.06	0.03	0.01	0.10	4
	IV	243	177.00	44.00	6	0	2	8	0.02	0.00	0.01	0.03	0
QA run 2	V	231	3.28	1.78	10	3	1	14	0.04	0.01	0.00	0.06	0
	VI	1364	13.54	3.90	31	9	0	39	0.02	0.01	0.00	0.03	0
	VII	701	8.11	2.91	25	14	0	40	0.04	0.02	0.00	0.06	6
	VIII	115	7.00	2.00	5	1	1	7	0.04	0.01	0.01	0.06	0

We considered the gathered inspection data, test data, and product data from these two quality assurance runs. Table 1 shows the relevant data. It can be observed that the code classes correspond to typical code classes as found in industrial applications. We empirically defined a set of assumptions and selection rules in a group session. The three assumptions we started with are listed below (for each, we found empirical evidence in the literature; e.g., the first one assumes a Pareto distribution for defects ([20], [21]), and Nagappan et al. [22] showed a high correlation between complexity and defect numbers, which is used in assumption three):

1. Parts of the code where a large number of inspection defects are found indicate more defects to be found with testing.
2. Parts of the code where a large number of inspection defects are found and which are of small size indicate more defects to be found with testing.
3. Parts of the code where a large number of inspection defects are found and which are of high complexity indicate more defects to be found with testing.

Each assumption was refined into a set of selection rules considering different inspection metrics we were able to derive (i.e., defect content, defect density, defect severity) as well as product metrics (i.e., size (in lines of code (loc)), mean method length (loc), and complexity (McCabe)). While we started with this small set of three assumptions and 32 refined selection rules, this set was systematically extended in order to perform a much larger analysis. Figure 4 gives an overview of how we arrived at these. Assumptions 1 to 6 use inspection data or product data only; assumptions 7 to 14 combine inspection and product data. For example, selection rule 1.1, assuming a Pareto distribution, was stated as “Focus testing on those code classes where the inspection found a large number of defects considering all inspection defects.” Selection rules 1.2 to 1.4 focus on those inspection defects that were classified as having “major”, “medium”, or “minor” severity. Selection rules 1.5 to 1.8 do the same for defect density.

		Sum SR
Assumption 1	Large #inspection defects → more test defects	8
Selection rule 1.1	Large defect content (all defects)	
Selection rule 1.2	Large defect content (major defects)	
Selection rule 1.3	Large defect content (medium defects)	
Selection rule 1.4	Large defect content (minor defects)	
Selection rule 1.5	Large defect density (all defects)	
Selection rule 1.6	Large defect density (major defects)	
Selection rule 1.7	Large defect density (medium defects)	
Selection rule 1.8	Large defect density (minor defects)	
Assumption 2	Small #inspection defects → more test defects	8
Assumption 3	Large size of code class → more test defects	2
Assumption 4	Small size of code class → more test defects	2
Assumption 5	High complexity of code class → more test defects	1
Assumption 6	Low complexity of code class → more test defects	1
Assumption 7	Large #inspection defects & large size of code class → more test defects	16
SR 7.1	Large defect content (all defects) & large class length	
SR 7.2	Large defect content (major defects) & large class length	
SR 7.5	Large defect content (all defects) & large m. method length	
SR 7.9	Large defect density (all defects) & large class length	
SR 7.13	Large defect density (all defects) & large m. method length	
SR 7.16	Large defect density (minor defects) & large m. method length	
Assumption 8	Large #inspection defects & small size of code class → more test defects	16
Assumption 9	Small #inspection defects & large size of code class → more test defects	16
Assumption 10	Small #inspection defects & small size of code class → more test defects	16
Assumption 11	Large #inspection defects & large complexity of code class → more test defects	8
Assumption 12	Large #inspection defects & small complexity of code class → more test defects	8
Assumption 13	Small #inspection defects & large complexity of code class → more test defects	8
Assumption 14	Small #inspection defects & small complexity of code class → more test defects	8
		118

Figure 4. Assumptions and selection rule (SR) composition

Each selection rule was further refined by defining a threshold to allow selecting concrete code classes. These thresholds were discussed in a group session and selected context-specifically. For most of the rules, the definition was obvious; for example, considering the defect content, two code classes have very high defect content, while two do not (see Table 1 again). With respect to code classes containing the defect severity "major", we decided to take code classes that have at least one such defect, as the consequences might be dramatic. However, in this case, other selections might have been possible.

These 118 selection rules were initially analyzed with respect to the 4-scale quality classification introduced above [7]. Their confidence counter was not calculated explicitly, which is something we did in this study. Furthermore, we calculated the precision and recall value for each rule, and completed our analysis with an aggregated F-measure in this study. The three classification procedures were compared with each other to judge their validity. The values were calculated independently for each selection rule in each quality assurance run, and were later analyzed with respect to their quality across both quality assurance runs.

4.1.4 Execution

After the assumptions and selection rules had been derived systematically, we evaluated each selection rule with respect to the three evaluation procedures. Because all inspection and test defect data were available from the two quality assurance runs (see Table 1), the calculation was straightforward and consumed only a small amount of time.

4.1.5 Results

We first analyzed the data from the two quality assurance runs independently (Figure 5), and considered data from both runs in a trend analysis afterwards (Figure 6).

	QA run 1		QA run 2	
	quality	# SR	quality	# SR
confidence counter	*1*	19	*1*	35
	0	99	*0*	83
quality class	*A*	15	*A*	11
	B	4	*B*	24
	C	33	*C*	0
	D	66	*D*	83
F-measure	*1.00*	15	*1.00*	11
	0.80	4	*0.67*	19
	0.67	16	*0.50*	5
	0.50	17	*0.00*	83
	0.00	66		

Figure 5. Evaluation results for selection rules

With respect to the first research question, the calculation of all three evaluation procedures was straightforward based on the defined selection rules and the existing defect data. Obviously, the confidence counter provides a coarse-grained view, as it checks whether all test defects are found during the prioritization. About 16 percent of the selection rules in the first run, and about 30 percent in the second run, provided satisfactory results (Figure 5). Considering the four quality classes, a more refined view on the validity of the selection rules is given. The most appropriate rules can be found in category A, which comprised 15 selection rules in the first run, and 11 selection rules in the second run (i.e., all defects were

found when using these selection rules). The selection rules in category B also provided an appropriate selection of code classes for the prioritization, but were not as efficient as those rules in category A. The rules in categories C and D are negligible, and their number is the same as that of the rules classified with a confidence of zero. The calculation of the F-measure slightly refines the evaluation. In the first run, the category C selection rules were split into two sets. The rules in both sets did not prioritize all defect-prone code classes, but the first set (value: 0.67) focuses on a smaller set of code classes and would therefore save more effort than the rules classified in the second set (value: 0.5). With respect to the second quality assurance run, quality class B was refined into two sets. The selection rules in both sets found all defects with the prioritization, but the first one required less effort because fewer code classes were selected.

Considering both quality assurance runs and calculating the validity of the selection rules during both runs, 11 selection rules focused testing on all defect-prone code classes during both runs, which is indicated by the value 2 of the confidence counter (Figure 6). This corresponds to quality classes AA, AB, and BB (i.e., three rules are classified into category A in both the first and the second quality assurance runs; six rules are classified into category A in the first quality assurance run and into category B in the second quality assurance run; and two rules are classified into category B in both quality assurance runs). The quality classes provide a much more detailed view than the confidence counter. The calculation of the mean F-measure refines the evaluation again. Rules with an F-measure of 1 correspond to rules classified as AA. However, it can be observed that based on the calculated F-measure, some other selection rules got high values compared to the eight rules classified as AB and BB. More specifically, some selection rules classified as AB and as CA got the same mean F-measure of 0.835. Finally, all evaluation procedures identified the 54 worst selection rules.

> RQ 1 conclusion: All three evaluation procedures are easily applicable provided that the necessary data is available. Considering only one quality assurance run, few to no new insights are obtained by the calculation of the F-measure. With respect to the initial trend analysis, a more detailed and slightly different set of selection rules is calculated as best rules. Considering additional quality assurance runs and their evaluation, it is expected that the calculated mean F-measure values will make it easier to get an overview of the best rules than the quality classes.

With respect to the second research question, the calculation of the F-measure for the selection rules basically confirmed the insights we got from the evaluation of the selection rules with respect to the confidence counter and the quality classes. Some more detailed insights could be gained. Table 2 shows an excerpt of the selection rules and the corresponding evaluation results.

confidence counter	#	quality class	#	mean precision	#	mean recall	#	mean F-measure	#
2	11	AA	3	1	6	1	11	1	3
1	53	AB	6	0.83	6	0.75	12	0.835	9
0	54	AD	6	0.75	2	0.5	20	0.75	1
		BB	2	0.585	1	0.25	21	0.735	1
		BD	2	0.5	26	0	54	0.67	1
		CA	4	0.415	4			0.65	1
		CB	8	0.33	2			0.585	3
		CD	21	0.25	17			0.5	14
		DA	4	0	54			0.4	2
		DB	8					0.33	20
		DD	54					0.25	9
								0	54

Figure 6. Evaluation results for selection rules over two QA runs

First of all, it was confirmed that rules using inspection defect data led to good predictions of defect-proneness, and that a combination of inspection defect data and product data led to the best selections (see rules 1-3 in Table 2). Some product data alone also led to suitable results, but not to the best results (see rules 6 and 8 in Table 2). Interestingly, the same F-measure was calculated for rules that had been classified into different quality classes before; for instance, rules 3 and 4 belong to different quality classes (rule 3 prioritized all defect-prone code classes in both runs, rule 4 only in one run), but both have the same F-measure. Furthermore, rules 4 and 5 belong to the same quality classes (both CA), but have a different F-measure. In addition, some rules using only product data (rules 6 and 8, both in category BB) and finding all defect-prone parts were evaluated worse than rules that overlooked defects (e.g., rules 4 and 5). This shows that the F-measure does not consider finding all defects as the only criterion for suitability, but also considers the effort consumed. Rule 9 showed inconsistent results during the two quality assurance runs, which are reflected in all evaluation procedures. Finally, a lot of rules are classified as rule 10, which provides no valuable selection of code classes for finding defects during testing.

Table 2. Selection rules (excerpt) and evaluation results

No.	selection rule	confidence counter	quality class	mean precision	mean recall	mean F-measure
1	Large defect content (all defects) & small mean method length	2	AA	1.00	1.00	1.00
2	Large defect content (all defects) & high class length	2	AB	0.83	1.00	0.84
3	Large defect content (all defects)	2	AB	0.83	1.00	0.84
4	High defect density(all defects) & high class length	1	CA	1.00	0.75	0.84
5	Low defect density (major defects) & small mean method length	1	CA	0.75	0.75	0.75
6	High class length	2	BB	0.59	1.00	0.74
7	Low defect density (crash defects) & high class length	1	CB	0.75	0.75	0.67
8	Low mean method length	2	BB	0.50	1.00	0.65
9	Low complexity	1	AD	0.50	0.50	0.50
10	Low defect content (all) & high complexity	0	DD	0.00	0.00	0.00

RQ 2 conclusion: The calculation of the F-measure confirmed that inspection defect data is a good predictor in our context, best in combination with certain product data. Rules using product data were rated slightly worse, and rules that we considered as middle-rate based on the quality classes (e.g., CA) were rated better when calculating the F-measure, which takes into account the number of defects found, but also the effort.

4.1.6 Threats to validity

Conclusion validity: The results we presented are based on two quality assurance runs and the conclusions we draw are only valid for this context. However, we could show that all three evaluation procedures are applicable with different levels of expressiveness, and we confirmed positive trends for using inspection results as (additional) predictor for defect-proneness.

Construction validity: We reused a set of already defined assumptions and selection rules that had been derived systematically. Additional ones can strengthen such analyses in the future.

Internal validity: Thresholds that were defined to operationalize selection rules could have been defined differently, which might have led to the selection of different code classes. To mitigate this threat, all thresholds were discussed in a team of quality assurance experts.

External validity: The considered tool was rather small and we selected only a subset of code classes for inspection and testing. Consequently, our conclusions have to be treated with caution when applying the approach in another context to find relationships between inspection / product data and testing.

4.2 Case study 2

4.2.1 Goal

After the first case study, we conducted a second case study in an industrial environment. An earlier evaluation of the considered data [8] showed that inspection metrics provided good results when predicting defect-proneness, but were very similar to size metrics. In that earlier study, we only considered the number of test defects found for an ad-hoc evaluation, and did not evaluate selection rules based on the three evaluation procedures.

Therefore, our main goal in this study was to evaluate whether the calculation of the F-measure leads to a better evaluation (and thus selection) of the most appropriate selection rules that predict defect-proneness compared to calculating the confidence counter or quality classes. We considered the same two research questions as in the first case study:

RQ 1. Which of the evaluation procedures is most applicable and provides the highest validity of the selection rules for focusing testing?
RQ 2. Which new or refined relationships can be found when applying the detailed evaluations of the In^{2}Test approach (in order to focus testing)?

4.2.2 Context

We used defect data from an organizational unit at Vector, a company that has been developing software for embedded systems for over 20 years. Vector uses a product family approach for development. One main focus is the automotive domain. Features are implemented in the form of components. At the time of the analysis, the software consisted of about 140 modules, each comprising a set of code classes. These code classes have a length of between 100 and 14,000 statement lines of code. The size of a module varies between several hundreds to 40,000 lines of code; also, one much larger module existed.

Due to the sensitive context, different quality assurance activities are conducted during development. For example, code inspections are performed on all released source code. Testing is done on all elements of the product family: the product itself, programs, deliveries, and components. The defect data and all feedback from customers are stored in the form of change requests. Even though the development procedure of the product family has changed over time, test and inspection data are available from more than ten years.

4.2.3 Design

In this section, we describe how we gathered the data, how we derived the assumptions and selection rules, and how we applied the three evaluation procedures.

Though we had initial experiences with the calibration of the In^{2}Test approach from the first case study, we were aware that we had to calibrate the approach for this new context. To do so, we reused some of our experiences from the first case study, e.g., we started with certain assumptions that had turned out to be valid before.

In order to be able to conduct such an analysis, we had to gather the relevant defect data. For this, we first selected about 10% of the modules at random. The main reason for this was that no separation of modules was possible based on any classification, as the modules were highly heterogeneous with respect to size, complexity, and age.

After the selection of these modules, we extracted the inspection and test defect data, and derived further product metrics. The code inspection defect data had to be extracted manually from change requests. Different numbers of such change requests existed for every module. The test defect data could be derived automatically from a database, but different test phases were not distinguished. Finally, two code metrics were considered: size in lines of code and a waste factor, which expresses the degree of changes within a module (a high value means that many lines of code were changed or deleted) [8].

Based on the experience from the first case study and the data available in the given context, we defined 16 assumptions, eight that consider only inspection or only product data, and eight that combine inspection and product data. For each assumption, we derived three selection rules. Due to the long timeframe we considered in our analysis and the randomly selected code classes, the defect distribution was highly heterogeneous. Therefore, it did not make sense to define concrete thresholds when deriving selection rules from the assumptions. Consequently, we used the assumptions to create sorted lists, and defined selection rules that focused on the top-25%, top-50%, and top-75% of the modules to select them for focused testing. The concrete assumptions are shown in Figure 7.

		Sum SR
Assumption 1	Large #inspection defects (all) → more test defects	3
Selection rule 1.1	25% of the modules where the inspection found the most defects	
Selection rule 1.2	50% of the modules where the inspection found the most defects	
Selection rule 1.3	75% of the modules where the inspection found the most defects	
Assumption 2	Large #inspection defects (all scaled) → more test defects	3
Selection rule 2.1	25% of the modules where the inspection found the most defects (for modules which are not completely inspected, the current ratio between detected defects and inspected parts of a module was used to calculate how many defects had been found if the module had been inspected completely)	
Assumption 3	Large #inspection defects (all w/o comments) → more test defects	3
Selection rule 3.1	25% of the modules where the inspection found the most defects (without considering comments)	
Assumption 4	Large #inspection defects (all w/o comments scaled) → more test defects	3
Assumption 5	Small size → more test defects	3
Assumption 6	Large size → more test defects	3
Assumption 7	Small waste → more test defects	3
Assumption 8	Large waste → more test defects	3
Assumption 9	Large #inspection defects (all) & large size → more test defects	3
Assumption 10	Large #inspection defects (all) || large size → more test defects	3
Assumption 11	Large #inspection defects (all) & small waste → more test defects	3
Assumption 12	Large #inspection defects (all) || small waste → more test defects	3
Assumption 13	Large #inspection defects (all w/o comments) & large size → more test defects	3
Assumption 14	Large #inspection defects (all w/o comments) || large size → more test defects	3
Assumption 15	Large #inspection defects (all w/o comments) & small waste → more test defects	3
Assumption 16	Large #inspection defects (all w/o comments) || small waste → more test defects	3
		48

Figure 7. Assumptions used in the second case study

Assumptions 1 to 4 use inspection data only. We defined the assumptions based on our experiences from the first case study, i.e., we assumed a Pareto distribution. Assumption 1 uses all inspection defect data from one module. The three selection rules were: "Select 25%/50%/75% of the modules for testing where the inspection found the most defects." The second assumption considered scaled inspection defect data, i.e., not all modules were completely inspected, but for some modules a rate was estimated regarding the inspected parts, which was used to calculate the defect content if the module had been inspected

completely. Assumption 3 used all inspection defect data except comments, and assumption 4 did the same, but in a scaled manner.

Assumptions 5 to 8 used the two product metrics size and waste. As empirical evidence is unclear as to whether a high or a low value leads to more defects, we analyzed both.

Finally, assumptions 9 to 16 combined some inspection metrics with the product metrics, using an AND and an OR connector. In summary, we defined 16 assumptions and 48 selection rules for that context.

These 48 selection rules were analyzed twice regarding their validity. First, we considered the test defect data as provided by Vector (original). Second, due to the wide range of the test defect data, we were interested in finding out whether other assumptions and selection rules would turn out to be valid if we considered only the most defect-prone modules. Therefore, we set the test defect number to zero for a module if a module had less than 10% of the defects of the module with the highest defect number (adapted) and checked if any differences in the evaluation occurred. This procedure is justified by the fact that typically, test managers have to allocate the available test effort to the testing of certain parts, where one strategy is to consider mainly those parts that were most defect-prone in the past. Furthermore, the already mentioned Pareto distribution of defects can often be observed in practice. With this, we could again focus only on the top defect-prone modules, which was also reflected in the calculations. Afterwards, we evaluated the rules again with respect to the three evaluation procedures.

4.2.4 Execution

We refined the new assumptions and selection rules, and evaluated each rule with respect to the three evaluation procedures. Because all defect data were already available from an earlier study, the calculation was again easy and consumed only a small amount of time.

4.2.5 Results

Figure 8 shows an overview of the evaluation of the 48 selection rules with respect to the three evaluation procedures.

	original test defect data		adapted test defect data	
	quality	# SR	quality	# SR
confidence counter	*1*	1	*1*	4
	0	47	*0*	44
quality class	*A*	0	*A*	0
	B	1	*B*	4
	C	47	*C*	44
	D	0	*D*	0
F-measure	0.947	2	*0.750*	5
	0.909	1	*0.727*	2
	0.857	3	*0.714*	2
	0.842	7	*0.667*	5
	0.824	2	*0.625*	2
	0.750	5	0.600	2
	0.737	4	0.571	9
	0.667	2	0.545	5
	0.625	4	0.500	6
	0.571	4	0.462	2
	0.462	10	0.429	1
	0.333	2	0.364	1
	0.182	2	0.333	2
			0.286	1
			0.250	2
			0.182	1

Figure 8. Evaluation results for selection rules

Considering the confidence counter for the original test data, only one selection rule prioritized all defect-prone modules. With respect to the quality classes, this selection rule was rated as B, but the remaining 47 selection rules were rated as C. When we look at the evaluation regarding the adapted test defect data, this becomes slightly better indeed, but no significant improvement can be observed. However, these two classification procedures do not present an adequate overview of the suitability of the different selection rules. The main reason is that almost all modules contain a certain number of defects, and both classification procedures do not differentiate between modules with a low or a high number of defects. Considering the F-measure, much better grading is possible, which allows evaluating individual selection rules much more precisely.

RQ 1 conclusion: All three evaluation procedures are easily applicable in an industrial context provided that the data is available (it might take some effort to gather the necessary data). Calculating the F-measure provides the most appropriate evaluation of selection rules in the given context.

We analyzed each selection rule in detail and compared them with each other. Figure 9 presents an excerpt of this analysis, showing the selected assumptions and selection rules, the precision, recall and F-measure, and the number of defects found together with the number of selected modules.

No.	Assumption	SR	pre-cision	recall	F-measure	% test defects found	# selected modules
1	Large #inspection defects (all, original)	25%	1.000	0.300	0.462	31%	3
		50%	1.000	0.600	0.750	82%	6
		75%	1.000	0.900	0.947	93%	9
3	Large #inspection defects (all w/o comments, original)	25%	1.000	0.300	0.462	31%	3
		50%	1.000	0.600	0.750	82%	6
		75%	1.000	0.900	0.947	98%	9
4	Large #inspection defects (all w/o comments scaled, original)	25%	1.000	0.300	0.462	79%	3
		50%	1.000	0.600	0.750	82%	6
		75%	0.889	0.800	0.842	94%	9
5	Small size (original)	25%	1.000	0.300	0.462	9%	3
		50%	0.833	0.500	0.625	17%	6
		75%	0.778	0.700	0.737	21%	9
6	Large size (original)	25%	1.000	0.300	0.462	79%	3
		50%	0.833	0.500	0.625	83%	6
		75%	0.778	0.700	0.737	91%	9
5'	Small size (adapted)	25%	0.333	0.200	0.250	7%	3
		50%	0.333	0.400	0.364	13%	6
		75%	0.222	0.400	0.286	13%	9
6'	Large size (adapted)	25%	1.000	0.600	0.750	87%	3
		50%	0.500	0.600	0.545	87%	6
		75%	0.444	0.800	0.571	93%	9
7	Small waste (original)	25%	1.000	0.300	0.462	25%	3
		50%	0.833	0.500	0.625	89%	6
		75%	0.778	0.700	0.737	93%	9
11	Large #inspection defects (all) & small waste (original)	25%	1.000	0.100	0.182	16%	1
		50%	1.000	0.300	0.462	79%	3
		75%	1.000	0.700	0.824	93%	7
15	Large #inspection defects (all w/o comments) & small waste (original)	25%	1.000	0.100	0.182	16%	1
		50%	1.000	0.300	0.462	79%	3
		75%	1.000	0.600	0.750	91%	6
14	Large #inspection defects (all w/o comments) || large size (original)	25%	1.000	0.400	0.571	82%	4
		50%	0.889	0.800	0.842	86%	8
		75%	0.818	0.900	0.857	98%	11

Figure 9. Detailed comparison of selection rules (excerpt)

First of all, selection rules using inspection defect data turned out to be the best ones. They have the highest F-measure, especially for the 75%-rule (rules #1 (considering all inspection defects), #3 (considering all inspection defects without those classified as comments), and #4 (considering all inspection defects without those classified as comments, and estimating the overall defect number for those modules where only parts were inspected)). The best rules are much easier to identify when using the F-measure than when only the percentage of defects found was considered, as in an earlier study [8]. The more modules were selected, the better the F-measure. With respect to the 25%- and the 50%-rules, the values are the same or very similar for different assumptions; the reason is that many modules contain at least some defects.

Besides considering the F-measure, it is also important to take the relative number of defects found into account. For example, have a look at assumptions 5 and 6, which are shown with respect to the original and the adapted data. Each of the three rules of assumptions 5 and 6 has exactly the same precision and recall values and the same F-measure, though the number of found defects varies heavily. The reason is that not the absolute number of defects is considered by the calculation, but rather defect-proneness as such is used. When considering the adapted defect data, where the defect distribution is similar to a Pareto distribution, this is reflected better in the calculation of the F-measures.

Assumption 7 shows that in our context, modules that hardly changed tended to have more defects, which is different than reported in the literature [11].

Assumptions 11, 14, and 15 combine different inspection and product metrics. The 75%-rule from assumptions 11 and 15 found more than 90% of the defects by selecting only seven, respectively six modules. In other words, only 50% of the modules were selected and 90% of the defects were found, which is a suitable ratio and gives an indication of the potential of the approach regarding effort savings. However, in this scenario, some defects would not have been found. Assumption 14 found nearly all defects when considering the 75%-rule, but also selected almost all modules, which does not constitute focusing in its intended meaning.

Comparing again the evaluations regarding original and adapted data, the general trend remains the same for identifying the best rules. For example, the best 25%-rules for the adapted data consider inspection data (all scaled; all those without comments scaled; both 0.750). Some differences can be found (e.g., the one mentioned regarding the size metric). The F-measure for the adapted data turns out to be smaller, as a few more "defect-free" modules were selected, too.

RQ 2 conclusion: It could be confirmed that inspection defect data used in selection rules are one of the best predictors for defect-proneness. Especially when considering the 75%-rules, the F-measure is much higher compared to product metrics. Considering a lower number of modules, no difference can be observed with respect to the original data, but a difference can be identified with respect to the adapted data. Furthermore, some combinations of inspection and product metrics led to a suitable number of found defects although fewer modules were used.

4.2.6 Threats

Conclusion validity: The results are only based on the analysis of a subset of all available modules from the context. However, results from an earlier analysis could be confirmed and slightly refined, i.e., the positive trend using inspection defect data was shown again by the new evaluations of assumptions and rules.

Construction validity: Many other assumptions and selection rules exist that could be considered in future evaluations. Furthermore, refined selection rules might be stated differently and different concrete values might be used. In addition, due to a lack of information, we were unable to differentiate between critical and non-critical test defects. Furthermore, adapting the original defect data by setting the defect content to zero for some of the modules makes the analysis more coarse-grained, and the rationale for doing so might not be understood by everybody. However, as the trend for both the original and the adapted data is similar, this threat is negligible.

Internal validity: The available historical defect data are subject to a certain degree of inaccuracy. However, the general trend of defect-proneness was realistic.

External validity: First, the results are valid for the given context and cannot be generalized. However, the evaluation confirmed that inspection defect data is a suitable predictor in the given environment. Such positive trends indicate that defect data that are available early from inspections and reviews might be an additional metric that can be used to optimize quality assurance. Nevertheless, in a new context, a sound evaluation of assumptions and selection rules has to be done first (calibration) before the approach can be applied to predict defect-proneness. Ideally, inspection and product metrics should both be considered.

5. Discussion

5.1 Lessons learned from the studies

We derived the following essential lessons learned based on the experiences we made during the evaluations:

- Using a confidence counter or the four quality classes to judge selection rules might be sufficient for small environments, but turned out to be insufficient in an industrial setting for us. The calculation of F-measures led to much better selections of valuable selection rules. This calculation confirmed the potential of selection rules using inspection data for defect-proneness prediction in our context.
- The corresponding precision and recall values show the completeness and the quality of each selection rule appropriately. However, especially in an environment where most of the modules contain at least a small number of defects, the concrete values can be misleading (e.g., same precision or recall value for modules with completely different numbers of defects found). Consequently, we recommend also considering the number of defects found per module.
- Such an evaluation analysis (i.e., calibration) can be automated and therefore consumes only little effort. We used a spreadsheet tool and scripts to calculate the values for a high number of different selection rules. Further analyses are therefore easily possible. The more often calibration is performed, the more reliable the results will be. After the analysis of data from two to three quality assurance runs, first trends emerge.
- Sound extraction of the defect data is necessary. This is often difficult for historical data, where a certain degree of inaccuracy can be observed. Furthermore, a clear definition of the context is needed in order to be able to understand in which environment the identified relationships are valid.
- A definition of thresholds works in small and controllable environments, but turned out to be difficult in the industrial environment. One reason was the large heterogeneity of the modules under consideration, and the resulting different defect numbers. Therefore, defining metrics in combination with creating sorted lists seems to be more worthwhile. In our environment, quality classes A and D were not selected, and B was rarely used. Again, using the F-measure was more applicable for us.
- In general, we could confirm first trends from earlier studies that the In^{2}Test approach could be a valuable addition to established approaches that use product or process data only when predicting defect-proneness. Of course, generalizing results has to be done with caution, as we only have results from two different environments. Indeed, the lessons we learned during the evaluations and the initial relationships we found can be reused when the In^{2}Test approach is to be applied in a different environment.

5.2 Open issues

The In^{2}Test approach should be further developed in the future to address several open challenges.

One question is how to treat with In^{2}Test large sets of code classes or large documents that cannot fully be inspected (e.g., due to time restrictions or due to a code basis that is too large). Different strategies might be possible in this case to select a representative set of parts to be inspected, e.g., grouping all code classes into different size categories (e.g., small, medium, huge, extra-huge) and selecting some substitutes from each size category. The results from inspecting such a much smaller set of code classes can be used to focus testing activities on the most problematic areas. Different categories (complexity in addition to size, for instance) can be combined to further find those areas that are expected to be most defect-prone. Similar to what we did, another possible strategy might be to select a random set of code classes. However, it has to be carefully explored how to scale the In^{2}Test approach. In the end,

inspection data might be one additional metric besides established product or process metrics that can further support the focusing of testing.

Another question is how to apply the approach in a more agile setting, especially if a test-first approach is used and thus test cases are defined before code is written. One pragmatic idea is to change the order of the two quality assurance activities in this case, i.e., testing activities might be used to focus the inspection activity on the code level, as the In^{2}Test approach is flexible enough in this regard. However, the current In^{2}Test approach can also be used more on the user story level and results from the inspection of these artifacts can be used to focus once again where to test. Another issue is that companies often try to improve a certain coverage criterion such as branch coverage with their test suite, and thus concentrate on something like complex code classes that contain lots of branches. However, even in this case, it has to be shown for the given context that complex classes are really more defect-prone. As shown in our studies, a combination of inspection defect data and established product metrics might improve such predictions and thus increase trust in high quality of the code, more so than simply concentrating on achieving coverage thresholds. To sum up, some open issues remain regarding when to apply the In^{2}Test approach in an agile environment.

Finally, the approach should be extended by considering (a) new selection rules (e.g., different inspection, product, or process data), (b) more quality assurance activities (e.g., different inspection and testing techniques), and (c) more levels of the software development lifecycle (e.g., design, requirements) in order to allow a more holistic and integrated quality assurance strategy.

6. Summary and outlook

In this paper, we compared three evaluation procedures to assess selection rules that describe relationships between inspections and testing. The confidence counter provided a simple overview, which, however, was often too coarse-grained. The quality classes gave more detailed insights about relationships, but were not applicable in a suitable manner in the industrial environment. Calculations of the F-measure provided the best and most detailed results, but could sometimes be misleading. This drawback can be reduced when defect numbers or the most defect-prone parts are considered.

The results of our analysis can be used in several ways, and are especially relevant for industrial contexts where quality assurance, and testing in particular, should be improved. Evaluated knowledge about these relationships can be used in the integrated inspection and testing approach In^{2}Test to focus testing activities based on inspection defect data, and consequently, to improve testing. For researchers, one main challenge is to identify additional relationships between inspections and testing, and on a more generalized level, between static and dynamic quality assurance techniques.

With respect to future work, we plan to evaluate the approach in more environments, identify new relationships between inspections and testing, and confirm the ones found. This will also include systematic derivation of new assumptions and selection rules. The In^{2}Test approach is currently being adapted, applied, and evaluated in the MBAT project, where a total of 20 case studies are being performed by 38 partners from the automotive, railway, and aerospace domains. Besides inspections, other analysis techniques are also being considered. Furthermore, the aspects mentioned in Section 5.2 will have to be addressed in the future.

In addition, we plan to consider additional input for the predictions, such as process data, expert knowledge, or data from other static analyses. Finally, data from different levels (e.g., design, requirements) should be considered to determine an improved test strategy, which

should lead to a comprehensive and holistic quality assurance strategy exploiting synergy effects across all quality assurance activities.

Acknowledgments

Parts of this work have been funded by the Stiftung Rheinland-Pfalz für Innovation project "Qualitäts-KIT" (grant: 925) and the ARTEMIS project "MBAT" (grant: 269335). We would also like to thank Sonnhild Namingha for proofreading.

References

[1] Harrold, M.J., Testing: A roadmap, *Proceedings of the Conference on The Future of Software Engineering*, 2000, 61-72.

[2] Juristo, N., Moreno, A.M., Strigel, W., Software testing practices in industry, *IEEE Software*, 23 (4), 2006, 19-21.

[3] Pressman, R., *Software engineering: a practitioner's approach*, 7th edition, McGraw-Hill, 2009.

[4] Elberzhager, F., Eschbach, R., Münch, J., Rosbach, A., Reducing test effort: a systematic mapping study on existing approaches, *Information and Software Technology*, 54 (10), 2012, 1092-1106.

[5] Elberzhager, F., Münch, J., Rombach, D., Freimut, B., Optimizing cost and quality by integrating inspection and test processes, *International Conference on Software and Systems Process*, 2011, 3-12.

[6] Elberzhager, F., Münch, J., Tran, V., A systematic mapping study on the combination of static and dynamic quality assurance techniques, *Information and Software Technology*, 54 (1), 2012, 1-15.

[7] Elberzhager, F., Eschbach, R., Muench, J., Using inspection results for prioritizing test activities, *21st International Symposium on Software Reliability Engineering*, 2010, 263-272.

[8] Elberzhager, F., Kremer, S., Münch, J., Assmann, D., Guiding testing activities by predicting defect-prone parts using product and inspection metrics, *38th Euromicro Conference on Software Engineering and Advanced Applications*, 2012, 406-413.

[9] Gyimóthy, T., Ferenc, R., Siket, I., Empirical validation of object-oriented metrics on open source software for fault prediction, *IEEE Transactions on Software Engineering*, 31 (10), 2005, 897-910.

[10] Fenton, N., Ohlsson, N., Quantitative analysis of faults and failures in a complex software system, *IEEE Transactions on Software Engineering*, 26 (8), 2000, 797-814.

[11] D'Ambros, M., Lanza, M., Robbes, R., An extensive comparison of bug prediction approaches, *Working Conference on Mining Software Repositories*, 2010, 31-41.

[12] Arisholm, E., Briand, L.C., Johannesson, E.B., A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, *Journal of Systems and Software*, 83 (1), 2009, 2-17.

[13] Nasser, V.H., Du, W., MacIsaac, D., Knowledge-based software test generation, *International Conference on Software Engineering and Knowledge Engineering*, 2009, 312-317.

[14] Emam, K.E., Laitenberger, O., Harbich, T., The application of subjective estimates of effectiveness to controlling software inspections, *Journal of Systems and Software*, 54 (2), 2000, 119-136.

[15] Karolak, D.W., *Software Engineering Risk Management*, IEEE Computer Society Press, Wiley, 1996.

[16] Klaes, M., Elberzhager, F., Muench, J., Hartjes, K., Graevemeyer, O.v., Transparent combination of expert and measurement data for defect prediction – an industrial case study, *International Conference on Software Engineering*, 2010, 119-128.

[17] Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., Mishra, R., Predicting software defects in varying development lifecycles using Bayesian nets, *Information and Software Technology*, 49 (1), 2007, 32-43.

[18] Barnard, J., Price, A., Managing code inspection information, *IEEE Software*, 11 (2), 1994, 59-69.

[19] Aurum, A., Petersson, H., Wohlin, C., State-of-the-art: software inspections after 25 years, *Journal of Software Testing, Verification and Reliability*, 12 (3), 2002, 133-154.

[20] Hamill, M., Goseva-Popstojanova, K., Common trends in software fault and failure data, *IEEE Transactions on Software Engineering*, 35 (4), 2009, 484-496.

[21] Shull, F., Basili, V., Boehm, B., Brown, A.W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M., What we have learned about fighting defects, 8th IEEE Symposium on Software Metrics, 2002, 249-258.

[22] Nagappan, N., Ball, T., Zeller, A., Mining metrics to predict component failures, *International Conference on Software Engineering*, 2006, 452-461.

[23] Prowell, S.J., Trammell, C.J., Linger, R.C., Poore, J.H., *Cleanroom Software Engineering: Technology and Process*, Addison-Wesley Professional, 1999.