Delft University of Technology
Software Engineering Research Group
Technical Report Series

Software Engineering in the Netherlands: The State of the Practice

Frens Vonken, Jacob Brunekreef, Andy Zaidman, Frank Peeters

Report TUD-SERG-2012-022

TUD-SERG-2012-022

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note:

Software Engineering in the Netherlands: The State of the Practice

Frens Vonken[b], Jacob Brunekreef[b], Andy Zaidman[a], Frank Peeters[b]

[a]*Delft University of Technology*
a.e.zaidman@tudelft.nl
[b]*Fontys University of Applied Sciences*
{f.vonken, f.peeters, j.brunekreef}@fontys.nl

Abstract

In order to determine whether there is a gap between the current state-of-the-practice and state-of-the-art in software engineering, we performed a broad survey among Dutch software producing organizations. Our survey covers aspects of the software engineering cycle ranging from requirements engineering, over design and implementation to testing. From our analysis of the data that we have obtained from our 99 respondents, we extracted 22 interesting observations, some representing unexpected insights from an academic point of view. From these observations, we have identified a number of avenues for future research.

1. Introduction

The term software engineering first appeared in the late 1960s and was introduced by Bauer to describe ways to develop, manage and maintain software so that the resulting products are reliable, correct, efficient and flexible [1, 2, 3]. Some 15 years later, Zelkowitz et al. performed an in-depth survey of 30 companies in which they tried to establish the current state of practice in the software production industry [2]. Their survey revealed that — at that time — practice was around 10 years behind on software engineering research. Almost 20 years later, in 2003 to be precise, Reifer observed that industry is a little behind academia, but industry has the capacity to close the gap very quickly [4].

Going on a quote from Strachey at the 1969 NATO conference on software engineering where he states that "there is a need for a greater mutual

understanding between the communities of software development practice and software research" [5, 1], we can only observe that this gap has existed for a very long time. In this respect, literature suggests two distinct ways for reducing the gap that exists between industry and academia: *education* [6, 7] and *technology transfer* [8].

For example, Lethbridge et al. [7] argue that we should better understand the dimensions of the field so that we can focus education appropriately. Focussing on education can also have direct benefits on practice, firstly, because academia is currently educating the next generation of software engineers, ready to take recent knowledge from academia to industry, and secondly, because we should also remember to continue educating existing practitioners to continually increase the level of professionalism in software engineering [9].

Rombach and Achatz have proposed a comprehensive model to transfer technology from academia to industry, which is very relevant, as Lethbridge et al. point out "very little of the advanced knowledge developed by the research community, such as that contained in the papers presented at ICSE[1] conferences, will have impact on practitioners." [7].

Knowing about the gap between academia and industry and its possible solution, combined with the observation of Ludewig that surveys on the actual state of software engineering in industry are rare [3], it is our goal to get a clear picture of current software engineering practices in the Dutch software production industry, with a particular focus to identify (1) topics for future/further research, (2) opportunities where existing research can be transferred to industry and (3) opportunities for fine-tuning educational programs.

From our survey of the Dutch software production industry, we have gathered 22 observations that characterize the state-of-the-practice and which could be interesting starting points for future research. We also relate each of these observations to current research efforts and insights obtained from literature.

The structure of this paper is as follows. Section 2 details our research methodology. Section 3 deals with our findings of what developer methodologies are being used, after which Section 4 looks at how requirements are gathered. Section 5 discusses our findings concerning how developers cope with the design of their software, while Section 6 takes a closer look at im-

[1] International Conference on Software Engineering

plementation. Section 7 focusses on testing. We conclude our paper with Section 8 that discusses our findings, presents threats to validity and proposes a research agenda.

2. Research methodology

In the spring of 2011 a survey was carried out to get insight into the state-of-the-practice of Dutch software development organizations. The survey should be considered as a broad survey that gauges how software development organizations work in general and how they ensure the quality of their software. The survey is meant to be a *tour d'horizon* with the explicit aim to gain insights into possible follow-up research initiatives and educational gaps. We explicitly opted for survey research [10, 11, 12] as our intended population was too large to observe directly.

2.1. Questionnaire

The survey questionnaire contains 50 questions[2]. The questions cover a range of software engineering practices, from questions on (1) the development methodology that is currently in use, (2) how requirements are specified, (3) how architecture and design are handled, (4) how software systems are realized and (5) testing. Moreover, several questions address the perception of quality, of the delivered software product and of the development process.

Distribution of the survey. The survey was sent out in distinct ways, namely:

1. Last-year students from Fontys University of Applied Sciences took the survey to companies where they had done an internship.
2. Emails were sent to companies that both Delft University and Fontys University of Applied Sciences have worked with in the past.
3. A broad call for participation was sent out using Twitter, including retweets by colleagues from other universities. A similar broad call was made through LinkedIn.

For the first category of respondents, they answered the survey on paper and the paper copy was returned to us. For the latter two categories the survey could be filled in using Google Forms[3].

[2]The questionnaire can be found at: http://swerl.tudelft.nl/bin/view/Main/SurveyDutchSoftwareIndustry

[3]http://www.google.com/google-d-s/forms/

In a period of just under two months, we had 99 people participating in our survey. The responses from both the electronic survey and the paper survey were collected in an Excel spreadsheet[4]. We first explored the data using descriptive statistics and boxplots, after which we used SPPS to compute correlations to identify statistically relevant observations (unless mentioned otherwise, we assume significance at the .05 level). During this process we found some remarkable results, which form the basis for the 22 observations that we report on in this paper.

2.2. Characterization of the subjects

The 99 people participating in our survey can be characterized as follows.

Function. From the 99 participants, 60 consider themselves *developers*, while 37 call themselves a *teamlead* or *project manager*. Two participants refrained from answering this question. While entering the company name was not a requirement, from those participants that did, we could gather that those 99 participants worked for at least 40 different companies.

Education. Figure 1 shows the educational background of the participants. Most of the participants have a CS educational background with 58 participants indicating to have followed a college-level computer science (CS) curriculum, 33 participants reporting to have followed a university-level CS curriculum (mind that some participants have followed more than one curriculum).

Experience. Looking at the years of experience of the developers, we see that the professional experience of the participants of our survey is very mixed. The largest group of respondents indicates to have 2–5 years of experience, but other experience groups are also well-represented. Figure 1 provides an overview.

In the following sections the results of the survey are presented. We have mirrored our results with findings reported by other scientists, looking for similarities and differences.

[4]The scores can be found on the same websites as the questionnaire

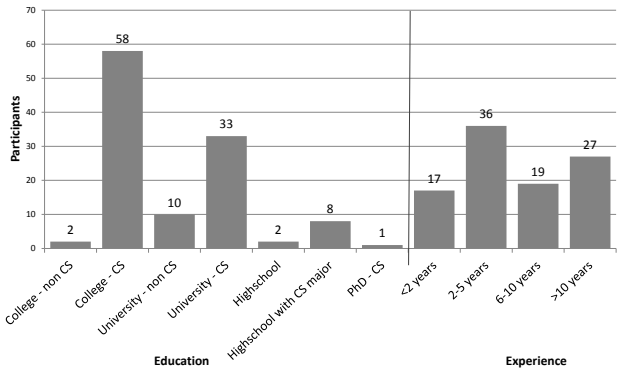

Figure 1: Educational background and experience of the participants.

3. Development Methodology

The history of software development shows an ongoing debate about what methodology should be used. Starting with a linear (waterfall) approach, nowadays iterative methods seem to be used more frequently [13]. In order to get an actual image of what kind of methodologies are currently in use at Dutch software companies, we have asked our respondents what methodology they are using. The results are shown in Figure 2. Also of interest is the fact that 41 respondents have reported to use some other development methodology. Analyzing the results indicates that most use a very ad-hoc development methodology or a company-specific methodology.

Observation 1. *Agile development / Scrum is often mentioned as development methodology, but the waterfall methodology is still in use.*

Looking at the boxplots in Figure 2 we see that many respondents indicate that they work with an agile development process. Scrum is used much more than other representatives of agile methodologies like XP and DSDM. On the other hand, 54 respondents (or more than 50%) indicate that they are still using a waterfall process (to some degree). Most of the waterfall practitioners are using this methodology for a longer time (over 3 years), while most of the agile/Scrum practitioners are using their methodology for a short time (less than or equal to 3 years).

Iterative development methodologies have been around for a decade now and carry a lot of interest from companies, as they seek to get their products to market more quickly [14]. Several empirical studies have been performed

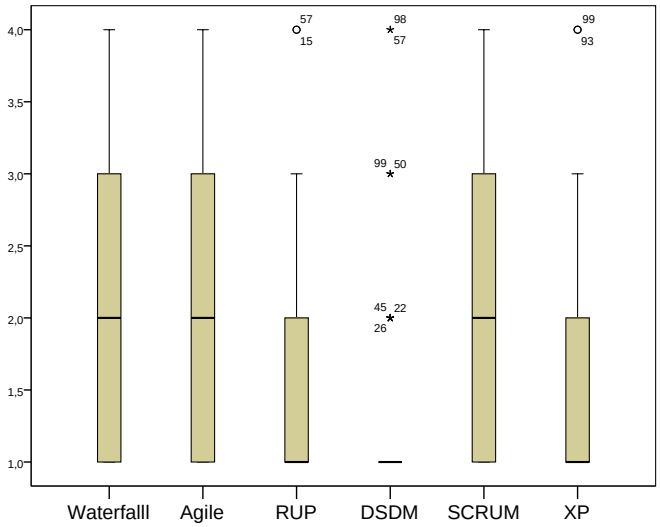

Figure 2: Boxplots showing which development methodologies are used and to what degree. (Scale from 1 (*not*) to 4 (*extensive*)).

on the quality of the resulting software and the gains in productivity while using iterative methodologies.

Dybå and Dingsør [15] present a systematic review of 36 studies, including topics like customer perception, product quality, team quality. From their study we can distil that most studies reported that agile development practices are easy to adopt in various environments and typically work well. Another finding is that developers are mostly satisfied with agile methods. In particular, for companies using XP, it seems that employees are more satisfied with their job *and* with the product they are delivering.

Hansson et al. [16] make an interesting observation: many industrial software development teams are already working more or less in an agile way, without knowing it. Companies that indicate that they work in a linear way all deal with changing requirements. They observe that most companies quite expertly combine iterative and linear practices and adjust their practices according to the situation at hand. They plea for more research concerning the combination of linear and iterative software development methodologies.

Petersen and Wohlin [17] report on a large industrial case study. They come up with positive results concerning agile software development. In another large-scale industrial case study, Petersen concludes that the waterfall

model is not suitable, mainly due to changing requirements [18].

Our findings are in line with the observations we gathered from literature: developers that are working in an agile setting are a little bit more satisfied with the quality of the software they produce and much more satisfied with the quality of the development process itself than developers that are using waterfall. See Figure 3.

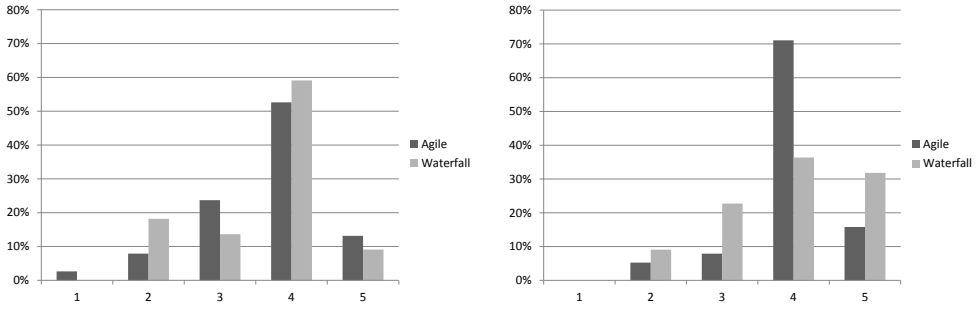

Figure 3: Developer satisfaction wrt Product Quality (left) and Process Quality (right). 1 = Very Unsatisfied, 5 = Very Satisfied

Layman et al. [19] describe a comparative case study in which the use of an agile development methodology, in particular XP, is compared to using a linear methodology. They report a 50% increase in productivity, a 65% improvement in pre-release quality, and a 35% improvement in post-release quality.

Several surveys (including ours) show that linear methodologies like waterfall are still in use. About 20% of our respondents still uses the waterfall methodology. This is less than the 33%, found by Laplante and Neill in a survey conducted in 2003 [20], but it is still a considerable percentage. We do not know whether this decrease is caused by ongoing time, or by other causes like the nature of companies that have been questioned. We also do not predict the end of the waterfall era. As several authors (like Boehm [21] and Sliger [14]) suggest, a combined approach with the best of both worlds may be successful in the future.

Observation 2. *Developers using waterfall are less positive about the quality of the development process and the quality of the software product than developers using agile methods.*

In Table 1 we have applied Kendall's tau correlation to determine whether a correlation exists between the use of the waterfall process and three process

aspects, namely the satisfaction with the development process, the realization of the project planning and the realization of the project budget. We opted for Kendal's tau correlation, a non-parametric test that measures rank correlation, which fits our need as our questionnaire uses a 5-point Likert scale. As can be seen in Table 1 we found a negative statistical correlation concerning developers using waterfall and their satisfaction with the development process, the realization of the project planning and the project budget.

	Process Satisfaction	Planning Realization	Budget Realization
Correlation Coeff.	-.350	-.227	-.239
Sig. (2-tailed)	.001	.025	.018

Table 1: Kendall's tau correlation between Waterfall developers and three process aspects.

Surprisingly, we did not find a positive statistical correlation concerning developers working in an agile manner and their satisfaction with process quality. Our observation contrasts reports from other studies. In particular, Cockburn and Highsmith report on an increasing employee morale when working in an agile manner [22], Mannaro et al. observe diminishing employee stress, while both Melnim and Maurer and Dybå and Dingsør witnessed an increasing job satisfaction [23, 15].

As can be seen from literature, software engineers seem to be quite positive about agile methods, yet our observation on process quality does not follow suit. Although it seems that agile development processes increase employee morale, diminish stress and increase job satisfaction, software engineers are not convinced of a better quality of the process.

Observation 3. *Agile developers share more team responsibilities with respect to product and process than waterfall developers.*

We consider this observation as a to-be-expected consequence of working agile: when working in short iterations, team members cannot focus for a longer period on just one activity or responsibility. We have also observed that sharing team responsibilities with respect to the development process correlates positively with the satisfaction concerning process quality: developers like sharing responsibilities. On the other hand, we found that sharing

team responsibilities with respect to the product-to-be-delivered correlates negatively with customer satisfaction and budget realizations. See Table 2.

This might, in part, be explained by the unease of customers with the unpredictability of the requirements. In particular, Murru et al. describe the experiences of an Italian internet company with XP [24]. They report that customers do not readily accept the unpredictability of requirements. This goes somewhat in the direction of our own observations from Table 2, where we saw a (weak) negative correlation between customer satisfaction and sharing responsibilities. On the developer-side they report that developers using XP have more sense of project control and are more aware of planning issues and budget issues.

	Customer Satisfaction	Budget Realization	Proc. Quality Satisfaction
Correlation Coeff.	-.191	-.175	.196
Sig. (2-tailed)	.032	.042	0.24
	Sharing Product Responsibilities		Sharing Process Responsibilities

Table 2: Kendall's tau correlations between sharing responsibilities and three process aspects.

4. Requirements specification

The success of a software system depends on how well it fits the needs of its users and its environment [25]. Software requirements comprise these needs, and requirements engineering (RE) is the process by which the requirements are determined. Successful requirements engineering involves understanding the needs of users, customers, and other stakeholders; understanding the context in which the to-be-developed software will be used; modelling, analyzing, negotiating, and documenting the stakeholders' requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution [26].

The quality of software requirements has been repeatedly recognized to be problematic [27]. In their early empirical study, Bell and Thayer observed that inadequate, inconsistent, incomplete, or ambiguous requirements

are numerous and have a critical impact on the quality of the resulting software [28]. In particular, the mismanagement of changing requirements or the lack of customer involvement have been identified as major risks in the development process [29].

Observation 4. *About 70% of the respondents report about direct customer/user involvement when specifying the requirements. In many cases customers and/or users are available on-site for consultation purposes.*

Observation 4 directly relates to customer involvement during requirements engineering: 70% of the respondents report customer involvement. As customer collaboration is one of the main topics in agile approaches, Observation 4 is in line with the frequent usage of agile development methodologies (see Observation 1).

Many authors stress the importance of customer involvement, often related to an agile way of working. Paetsch et al. [30] describe the role of customers in all steps of the requirements process, from elicitation up to validation. Cao and Ramesh [31] report about an empirical study concerning agile requirements engineering. One of their main findings is the importance of face-to-face communication between customers and the development team. DeLucia et al. [32] also discuss requirements engineering in an agile context. They come up with a set of guidelines for agile requirements engineering, including the importance of customer involvement.

With respect to direct versus indirect customer involvement, in our survey we found only a slight difference between respondents using agile and respondents using waterfall processes: 70% of the agile respondents and 64% of the waterfall respondents mention direct customer involvement. However, where the majority of the agile respondents report on-site customer involvement, only 20% of the waterfall respondents indicate that they work with their customers on-site in the requirements process. Our survey shows the importance of on-site customer involvement with respect to customer satisfaction and product satisfaction (see Table 3).

From Table 3, we have extracted Observation 5.

Observation 5. *On-site involvement of customers correlates positively with both the customer satisfaction and developer satisfaction concerning the quality of the delivered software product.*

	Customer Satisfaction	Product Satisfaction
Correlation Coeff.	.195	.182
Sig. (2-tailed)	.029	.040

Table 3: Kendall's tau correlation between On-site customer involvement and Customer satisfaction, Product satisfaction.

Managing the over time changing of requirements is a major issue in the requirements engineering field. Many authors consider agile methods as a good solution for this problem: where linear methods focus on fixing requirements in an early stage, the iterative way of working allows, or even stimulates, the change of requirements during a project, see, e.g., Eberlein and Leite [33] and Sillitti et al. [34].

Another issue is the way in which requirements are documented. As the Agile Manifesto favours working code over comprehensive documentation, a potential problem with respect to requirements documentation might occur [31]. Especially in the maintenance phase of the software lifecycle, when the development team often is out-of-sight, well-documented requirements are an important factor, amongst others to perform impact analysis [35]. As such, better tool support for documenting requirements seems like a logical next step. A first step in this direction has been presented by Mugridge and Cunningham with the Fit framework that allows to specify requirements through acceptance tests [36].

Observation 6. *Requirements are mainly specified in unstructured text, using word processors.*

More structured descriptions like use cases, acceptance tests and GUI prototypes are sometimes used, but the textual description of requirements remains a firm favourite. Figure 4 provides an overview of the specification methods used for requirements. If we then turn to the tools that are used for specifying requirements, we see that tools like Enterprise Architect, Visual Paradigm and Rational Rose are sometimes used, but a word processor remains the clear favourite. In fact, 56.6% of the respondents indicated that they extensively used a word processor for specifying requirements.

A striking observation is that what we would describe as a more professional way of dealing with requirements, i.e., specifying requirements through

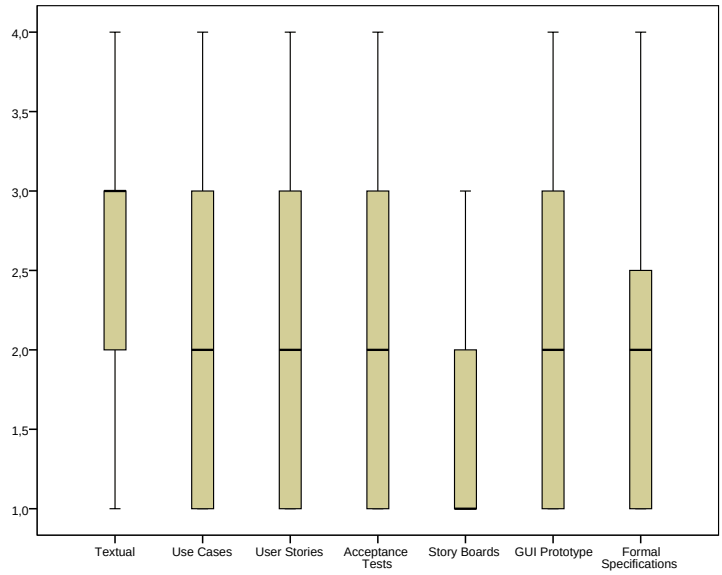

Figure 4: Usage of specification methods for requirements (Scale from 1 (*not*) to 4 (*extensive*).

use cases, resulted in a slightly negative correlation with regard to realizing projects within time and within budget amongst our respondents (see Table 4).

	Planning Realised	Budget Realised
Correlation Coeff.	-.201	-.191
Sig. (2-tailed)	.026	.035

Table 4: Kendall's tau correlation between applying Use Cases for requirements and realisation of planning and budget.

5. Design

Van Vliet states that "software design concerns the decomposition of a system into its constituent parts" [37]. Many design methods were proposed in the 1960s and 1970s [38, 39]. This research arose in response to the unique problems of developing large-scale software systems first recognized

in the 1960s [40]. The main aim of these software design methods was to enforce discipline in addressing the growing complexity of the applications being developed [38].

Design entails balancing a set of competing requirements. The products of design are models that enable us to reason about structures, make trade-offs when requirements conflict, and in general, provide a blueprint for implementation [38]. The above clearly indicates that design is an activity separate from implementation, requiring special notations, techniques and tools to ease working with large-scale software.

Towards the 1990s, the term software architecture came into fashion [39]. With systems becoming ever larger and more complex, software architecture is meant to provide a higher level of abstraction for software systems than software design could. Where software architecture stops and (detailed) software design starts however, is unclear from literature (e.g., [41, 42]).

When trying to determine which specific activities belong to software design, the software engineering community is rather ambiguous on this matter. No clear-cut distinction between the two development phases of analysis and design is made. Some experts will assign, for instance, a detailed description of the flow of events of a use case towards the analysis phase, while others, like Lauesen, prefer to label this as belonging to the design phase [43]. Lauesen likes to work with task hierarchies, compared to use cases, when talking about analysing user requirements. The same confusion arises when referring to usability requirements and other quality attributes (or in more general terms: non functional requirements). We assume (and hope) that a requirements engineer specifies only needs, conditions and perhaps wishes. To our opinion, a requirements engineer should avoid to specify parts of solutions, but we acknowledge that the difference between *"what"* and *"how"* is not always that clear.

Coming back to our own survey, we observed that how people deal with software design seems to be determined individually, as evidenced by the small number of observations that we made in this context. We observed a wide spectrum of answers, without a clear consensus. We are now turning towards the observations that we made while analysing the results of our survey.

Observation 7. *The majority (76%) of the respondents somehow rely on a software design, varying from a paper sketch to a class diagram made with the help of tools like Visio, Rational Rose, Rhapsody, Visual Paradigm or*

Enterprise Architect. Those who are using a tool are mostly working with Visio.

Our observation is supported by the boxplots in Figures 5 and 6. Figure 5 shows that our participants use a variety of methods to capture design, however, sketches and textual descriptions are used the most. Another (equally) popular way of communicating design is by creating a GUI design. Somewhat less frequently used are UML class and sequence diagrams.

When turning our attention to the tools that are being used for capturing the design, we see in Figure 6 that Visio is actually the most used tool, by far.

Smith makes a distinction between drawing tools (e.g., Visio), code-centric UML tools (e.g., Rational Rose) and UML framework tools (e.g., Rational Rose Real-Time and Rhapsody) [44]. The biggest difference between these classes of tools lies in the fact that code-centric tools offer facilities to generate concrete programming code, but mostly restricted to the static structure of a class diagram. The framework tools are more ambitious with respect to round trip engineering. Drawing tools on the other hand, of which Visio is a prime example and according to our results also frequently used, gives the practitioner the opportunity to make a quick design. Such a design makes it possible to communicate within the development project about design issues.

A possible explanation of the relatively low usage of UML tools like Enterprise Architect, Rational Rose, Rhapsody and Visual Paradigm is that respondents are not convinced enough of the advantages of the code-centric or framework tools. In this context, Dobing and Jeffrey performed a survey in 2006 [45]. One of their questions focused on the reasons for not using a UML class diagram, see Table 5 for the results.

Problem identified	% of respondents agreeing
Not well understood by analysts	50
Not useful for most projects	13
Insufficient value to justify cost	13
Information captured would be redundant	25
Not useful with clients	25
Not useful with programmers	25

Table 5: Reasons for not using a class diagram (% responses) [45]

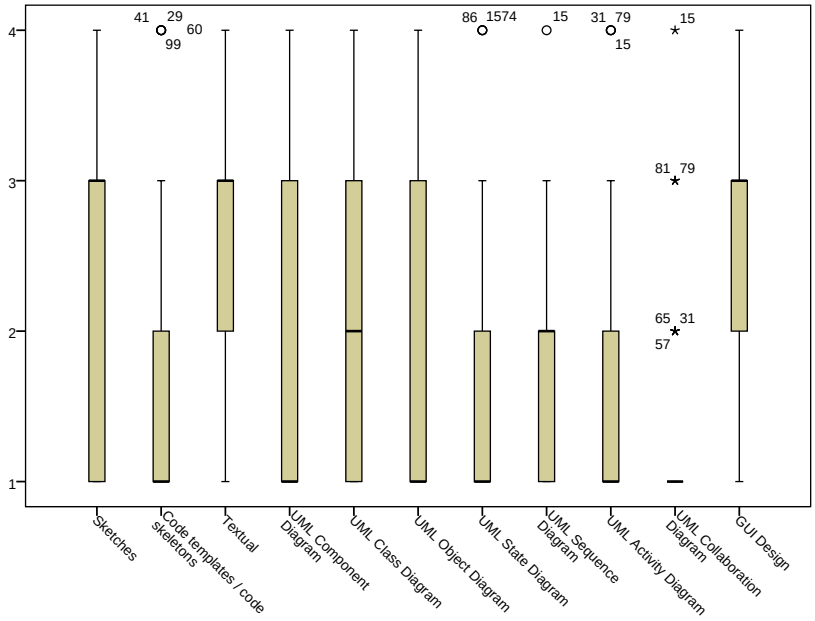

Figure 5: How is design captured? (1 = not, 2 = somewhat, 3 = average, 4 = extensive)

It is remarkable that 50% of the analysts did not have a good understanding of the finesses of a class diagram. Dobing et al suggest "that more extensive educational programs are needed, both to increase the number of analysts familiar with UML and to provide ongoing support to help them make fuller use of its capabilities" [45]. A recent literature research on empirical evidence about the use of the UML by Budgen et al. produced a same sort of advice with respect to the need of an adequate level of training in the use of the UML [46].

This observation seems also to be in line with results of a research by Cherubini et al. [47]. They did a survey at Microsoft about how and why software developers use drawings. Their research revealed the fact that developers are using diagram drawings in the first place to discuss about design issues. Afterwards those sketches are mostly thrown away. They serve mainly a short term goal: to understand the problem and get more quickly towards a reasonable solution. Most of the software developers are focused on producing code. So solutions are eventually captured by code and (unfortunately) not by design based on diagrams supplemented with explaining text. It seems

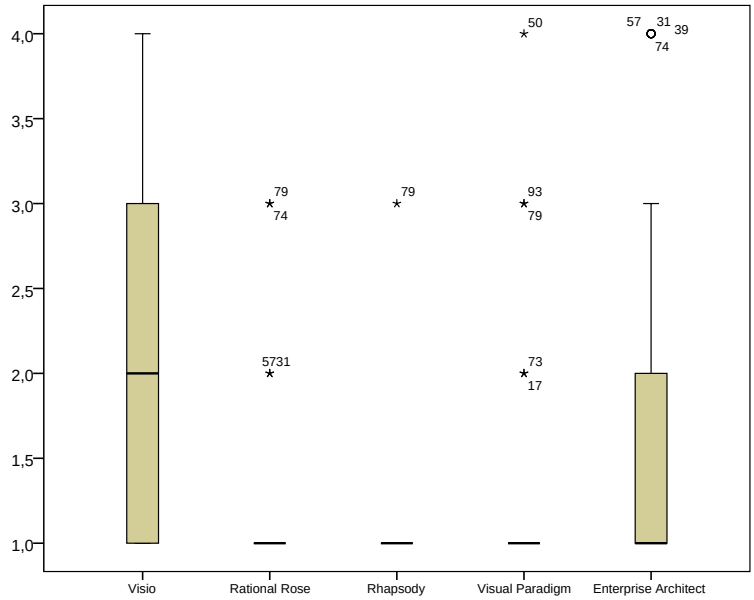

Figure 6: Which of the mentioned tools are used during design? (1 = not, 2 = somewhat, 3 = average, 4 = extensive)

that in the software industry the following adagium still reigns: "Code is the king" [48].

Observation 8. *Traceability between requirements and software design seems important, but not to everyone.*

The question "To what degree are the requirements coupled with the design" is only answered by half of the respondents. Which leads us to believe that either people are not aware of this traceability being implemented or are not aware of the possible benefits of doing so. The other halve, who did respond to the question, are relatively strict in applying this form of traceability with a median score of 4 on a 5-point Likert scale (see Figure 7). The large deviation in the answers on this question in combination with absence of 50% of respondents leads us to the impression that the importance of this form of traceability is underestimated by our respondents.

In this area, De Lucia et al. performed an experiment with their Information Retrieval-based traceability recovery tool ADAMS [49]. One of their findings was that the tool significantly reduces the time spent by the software engineer to trace links between decisions in design and requirements.

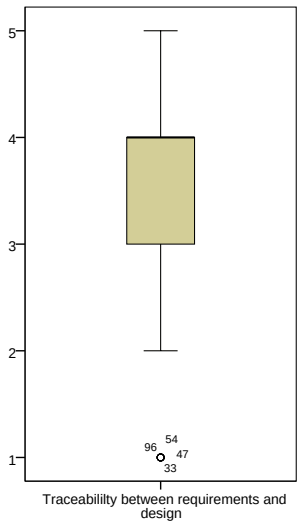

Figure 7: Coupling between requirements and design (1 = not at all, ..., 5 = completely).

It especially helps software engineers who are more or less novices in this field. They also came to another conclusion: the experiment demonstrates that the use of a traceability recovery tool in general improves the tracing *accuracy* of a software engineer, regardless of his ability.

Another reason why traceability is very important, is to do impact analysis, i.e., determining which parts of a software system are affected when requirements change [50].

Observation 9. *There exists a negative correlation between the use of class diagrams on the one hand and getting finished on time or within budget on the other hand.*

We observed a rather peculiar negative correlation between certain UML design activities with respect to the opinion about certain aspects of the software development process. In particular, the use of a class diagram is negatively correlated (-0.37) with delivering within time and also negatively correlated (-0.34) with staying within the (financial) budget (see Table 6). This is an indication that our respondents do not experience the positive effect of the use of class diagrams on the project planning. We suppose that practitioners experience a lot of pressure, to reach the deadline of the

project due to the requirement of creating class diagrams. This correlates with observations from Budgen et al., who have stated that it is clear that UML does not, in general, reach a sufficient cost-benefit ratio in the minds of developers to warrant its use [46].

	Finished on time	Finished within budget
Correlation Coeff.	-.371	-.338
Significance (2-tailed)	.000	.001

Table 6: Kendall's tau correlation between the design of a class diagram versus deadlines and budget.

The observation about the potential contra-productive effects with the use of class diagrams seems to be in contrast with the research of Nugroho and Chaudron [51]. They conducted a survey based on 80 respondents who are responsible for implementing UML models. This survey revealed a positive correlation between a strict use of UML-models (like a class diagram, sequence diagram and package diagram) and the perceived productivity in the implementation phase. In an earlier survey performed by Lange et al., a result of interest was the fact that software designers do not feel sure enough about UML descriptions because of four main problem classes [52]:

- Design choices are too much scattered over multiple views;
- Architects focus on what they think is important, there are no objective criteria on which views really need a detailed elaboration;
- Some system parts that are perceived as being complex, are elaborated with a disproportional detail;
- UML design products within one team are in some way inconsistent, because of different styles and different understandings of the system.

Also worth mentioning is the fact that Budgen et al. give advice to evaluate the UML, because a lot of its artifacts are too complex, not needed or ill defined. They claim that the UML is nowadays too much an accepted frozen status quo. Further scientific research is needed, with the ultimate result that practitioners feel more comfortable when using a simplified and more usable version of the UML [46].

6. Implementation

With the term "implementation" we refer to the software development phase where the coding is done. During this phase developers concentrate on specifications of individual components, translating these into executable code based on the design from the design phase [37].

To get an idea of how software systems are realized, several questions in our questionnaire revolve around which tools and techniques are applied during software implementation. The extent of these questions goes from what programming languages and IDE's are used, what standards and ways of working like pair programming, refactoring, etc. are in use, as well as whether tools like version control and bug tracking are in place during the implementation phase.

Observation 10. *Application of coding standards is common practice and is related to satisfaction with the development process.*

Looking at the application of standards in software development we see that coding standards are commonly used. More precisely, 80% of the respondents report an average or extensive application of coding standards (see Table 7). Additionally, the use of coding standards is significantly positively related to satisfaction with the development process, but not statistically significant to the satisfaction of the developers with the product being developed (see Table 8).

In the same vein, Boogerd and Moonen [53] could not find a straightforward relation between the application of coding standards and the occurence of software faults.

	Coding standards	Documenting standards	GUI standards
not	4%	22%	23%
somewhat	15%	31%	29%
average	43%	33%	30%
extensive	37%	14%	18%

Table 7: Percentages of application of coding-, documenting- and GUI standards.

In contrast to the frequent use of coding standards, respondents answered that documentation and GUI standards are used in respectively 47% and

48% of the respondents' companies (see Table 7). Interestingly, we observe a significant positive mutual relation between the application of each of these standards. An explanation for this might be that the application of standards in general can become a practice whithin a development team or company.

		Application of coding standards	Application of GUI standards
Satisfaction with product quality	Correlation	.139	.201
	Significance (2-tailed)	.124	.023
Satisfaction with process quality	Correlation	.208	.206
	Significance (2-tailed)	.019	.017

Table 8: Kendall's tau correlation between application of standards and product / process satisfaction.

A similar significant positive mutual relation can be found between the application of process support systems like version control, bug tracking and issue management systems (see Table 9). For the latter this may not be surprising because they are often integrated in one tool. Interesting is though that the application of each of these process support systems is again significantly positively related to the application of coding standards (see Table 10).

	Application of bug tracking systems	Application of issue management systems	Application of issue management systems
Correlation	.252	.225	.642
Significance (2-tailed)	.005	.014	.000
	Application of version control systems		Application of bug tracking systems

Table 9: Kendall's tau correlation between application of version control, bug tracking and issue management systems.

The positive relation between the application of coding standards and satisfaction with the development process (see Table 8) may be explained by the fact that application of these standards implies that the development process did get explicit attention. The relation between the application of

		Application of coding standards
Application of version control systems	Correlation Significance (2-tailed)	.317 .001
Application of bug tracking systems	Correlation Significance (2-tailed)	.238 .007
Application of issue management systems	Correlation Significance (2-tailed)	.218 .015

Table 10: Kendall's tau correlation between application of version control, bug tracking and issue management systems on the one hand and coding standards on the other.

coding standards and the other standards and process support systems mentioned above may suggest that the application of coding standards could be a good starting point in professionalising the process of software realisation. In a similar context, Paulk describes that the application of coding standards is one of the XP practices leading to a level 3 process maturity in the Capability Maturity Model (CMM) [54].

A final remark we want to make, related to the above, concerns the very common use of version control systems. These systems were reported to be used in an average or extensive way by 88% of the respondents. On the other hand we did not find any relation between the use of version control systems and the reported opinions about product or process quality.

Observation 11. *The application of pair programming correlates with the application of refactoring and unit testing.*

Observation 12. *The application of refactoring and unit testing do not correlate.*

As pair programming, refactoring and unit testing are all mentioned as extreme programming practices [55], we were not surprised to find that the application of pair programming correlates positively to the application of refactoring and unit testing (see Table 11). In contrast, considering the role of unit testing as a safeguard during refactoring, it is surprising to find that the application of these two practices turns out to be unrelated.

		Application of refactoring	Application of unit testing
Application of pair programming	Correlation Significance (2-tailed)	.390 .000	.265 .003
Application of refactoring	Correlation Significance (2-tailed)		.125 .150
Application of unit testing	Correlation Significance (2-tailed)	.125 .150	

Table 11: Kendall's tau correlation between application of Pair programming, Refactoring and Unit testing.

A possible explanation can be found in the fact that the original refactoring guidelines from Fowler [56] do not include guidelines for adapting unit tests. Additionally, van Deursen and Moonen established that in many cases refactoring the production code will entail a necessary refactoring of the test code as well, e.g., due to changes to interfaces [57].

A similar insight comes from the field study that Kim et al. [58] performed at Microsoft: *"developers need a better validation tool that checks correctness of refactoring, not a better refactoring tool"*. Furthermore, developers acknowledge that *"the primary risk is regression, mostly from misunderstanding subtle corner cases in the original code and not accounting for them in the refactored code"*. Yet another problem signalled by Kim et al. is the possible lack of sufficient unit tests, which according to the developers that they interviewed, would inhibit them to start refactoring.

Building upon that explanation, the fact that the application of refactoring and unit testing are unrelated may also explain why we did not find any relation between the application of refactoring and satisfaction with the quality of the product (see Table 12). When refactoring is performed without adaptation of test code the refactoring may possibly introduce unnoticed errors.

Our findings seem to be consistent with findings from Mohammad, who found no clear positive relation between refactoring and software quality in terms of adaptability, maintainability, understandability, reusability and testability [59].

For refactoring to have a clearer effect on product quality, a possible improvement may be extending the refactoring guidelines as proposed by Basit

		Application of Refactoring
Satisfaction with Product quality	Correlation Significance (2-tailed)	.069 .439

Table 12: Kendall's tau correlation between application of refactoring and satisfaction with the product quality.

et al. [60], such that all entities effected by the refactoring, including the unit tests, are covered in the description of how to perform the refactoring. An even further improvement may be the introduction of tools that help to establish traceability links between production code and test code as proposed by Hurdugaci and Zaidman [61] and Qusef et al. [62].

Observation 13. *Pair programming has a significant positive correlation with satisfaction with the quality of the development process.*

This satisfaction exists irrespective of the development process leading to more timely delivery. The application of pair programming was unrelated to the extent to which the original planning was met (see Table 13).

		Application of Pair programming
Satisfaction with quality of the development process	Correlation Significance (2-tailed)	.201 .024
Degree to which planning was met	Correlation Significance (2-tailed)	.053 .559
Satisfaction with quality of the realised product	Correlation Significance (2-tailed)	.082 .367

Table 13: Kendall's tau correlation between application of pair programming and satisfaction with the development process and the degree to which the planning was met.

Satisfaction with the development process seems to be typically based on personal experiences of the developer. This is for instance confirmed by studies from Williams et al. [63] who found that pair programmers are more satisfied with their job, are more confident about the result and admit to

work harder and smarter. Bipp et al. [64] observed that pair programmers rated their pair programming partners as helpful and motivating. Several other results in this direction can be found in the review study by Dybå and Dingsøyr [15].

Observation 14. *Despite more satisfaction with the development process there is no positive relation between the application of pair programming and perceived client satisfaction or product satisfaction.*

The data in Table 13 support our Observation 14. Strangely enough, this observation conflicts with the ambition of pair programming to improve software quality [55]. An ambition that is reported to be reached by researchers like Williams et al. [63] and Begel et al. [65]

On the other hand, other researchers like Bipp et al. [64], Arisholm et al. [66] and Hannay et al. [67] did not find a simple straightforward positive effect of pair programming on software quality.

These contradicting results may be explained by variability in the tasks of the development process which are done in pairs, by characteristics of the products being developed and by characteristics of the developers themselves.

Observation 15. *Quality monitoring and related process adaptation are more often applied according to project- and teamleaders than according to developers.*

Observation 16. *According to the project- and teamleaders the degree of quality monitoring and related process adaptation has a significant positive correlation to satisfaction with the development process.*

Observation 17. *According to developers the degree of quality monitoring and related process adaptation has a significant positive correlation to satisfaction with the product developed and with client satisfaction.*

The observation that quality monitoring and related process adaptation are more often performed according to project- and teamleaders than according to developers may be explained by the different roles they play in the development process. As project- and teamleaders have the development process as their core responsibility they may be more sensible to activities related to the development process (see Figure 8).

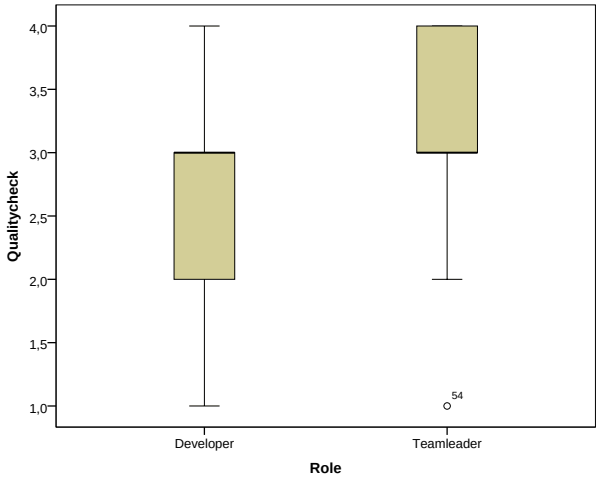

Figure 8: Application of quality monitoring and process adaptation according to developers and teamleaders/projectleaders (1 = not, 2 = somewhat, 3 = average, 4 = extensive).

The same explanation may hold for the case where there is a statistically significant positive correlation between applying quality monitoring on the one hand and satisfaction with the development process on the other hand. This relation may simply be noticed more by the individuals that are more involved with or responsible for the development process (see Table 14).

		Degree of quality monitoring by developer	Degree of quality monitoring by project/teamleaders
Satisfaction with the quality of the realised product	Correlation Significance (2-tailed)	.266 .021	.159 .280
Satisfaction with the development process	Correlation Significance (2-tailed)	.170 .133	.328 .026
Satisfaction of the client	Correlation Significance (2-tailed)	.250 .034	.235 .111

Table 14: Kendall's tau correlation between degree of quality monitoring and satisfaction with the developed product, the development proces and client satisfaction for developers and project- teamleaders.

The fact that we found a significant positive relation for developers between quality monitoring and satisfaction with the product developed, as well as with client satisfaction, corresponds to findings from the realm of Test-Driven Development (TDD). In particular, Crispin reports that client satisfaction increased, when using test-driven development, partly because of a lower defect rate that she attributed to TDD [68]. Another factor that Crispin raises is that amongst others TDD allowed the company under study to implement features that competing companies thought were too complex to automate. It must be said though, that TDD is just one factor, and other factors that were used during development might also have played a role here.

We were surprised to observe that according to project- and teamleaders no significant positive relation was found between quality monitoring on the one hand and satisfaction with the product developed and client satisfaction on the other hand. This phenomenon may be explained by the fact that such a positive effect, if it exists, may not be prominent and recognisable enough for project- and teamleaders to be noticed.

7. Testing

Studies estimate that software testing can consume fifty percent, or even more, of the total development costs [69, 70, 71]. Because of these high costs, and due to the pressure of tight deadlines [71], software testing is sometimes neglected, resulting in a situation where software potentially affects the lives of millions of people, yet at the same time is vulnerable to defects. In this context, Zaidman et al. have shown that sometimes production code and test code do not evolve together [72].

Garousi and Varma have nevertheless established that between 2004 and 2009 the attention given to (automated) testing in industry has increased [73]. Therefore, understanding the costs, but also the potential benefits of software testing, we therefore wonder what the current state-of-the practice is.

Observation 18. *Team members are more satisfied with the quality of the software development process when the V model is used.*

From Figure 9 we learn that the V model is only averagely used, with a median score of 3. Yet, at the same time, statistically we observe a positive significant correlation with the satisfaction of team members of the quality of the software development process. See Table 15 for details.

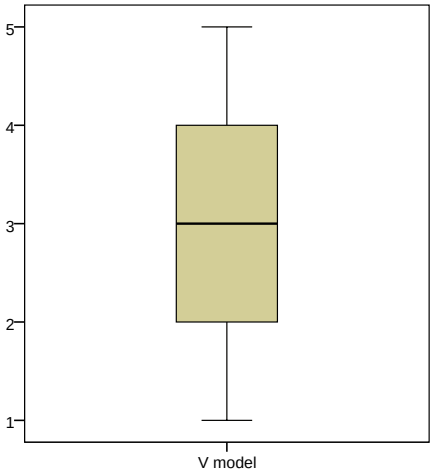

Figure 9: Degree to which the V model is applied (1 = not at all, ..., 5 = completely).

		Usage of the V model
Satisfaction with the development process	Correlation	.191
	Significance (2-tailed)	0.025

Table 15: Kendall's tau correlation between usage of the V model and satisfaction with the development process.

This observation can be explained through the fact that the V model is a valuable guideline for software developers and software testers to understand how and when to test [74]. This corresponds with the observation from Pol et al. who claim that "There is a real hunger in the software testing world for a more structured approach for testing. Testing professionals find satisfaction in knowing that they have done a good job. But how can you do a good job if you don't know what to do?" [75].

Observation 19. *System and acceptance tests are typically coupled to requirements.*

Figure 10 shows that with a median score of 4, most respondents indicate that system and acceptance tests are coupled to requirements. As defined in the IEEE Standard 1012-1986 [76], acceptance testing is a "formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system". By definition, it seems logical to couple the acceptance tests to the requirements that they check. This is also confirmed by Uusitalo et al. who carried out a series of practitioner interviews at five Finnish organizations and established that organizations are increasingly interested in binding requirements and testing more closely together [77].

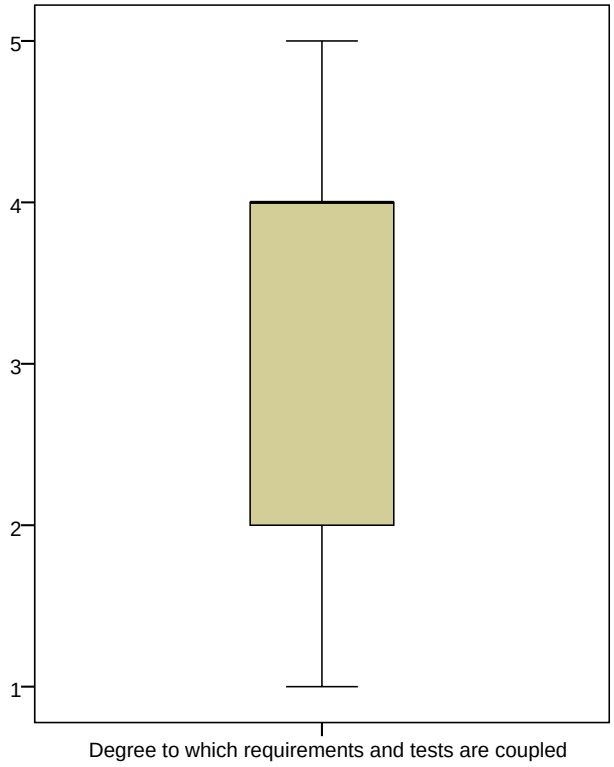

Figure 10: Degree to which requirements are coupled to system and acceptance tests (1 = not at all, ..., 5 = completely).

However, Young reports that 85% of defects are estimated to originate from inadequate requirements [78]. In this respect, recent research has focused on making it easier to define acceptance tests. One of the efforts in this area is Fit (Framework for Integrated Test), an an open source framework used to express acceptance test cases and a tool for improving the communication between analysts and developers [36]. In 2007 then, Ricca et al. performed a controlled experiment that showed that acceptance tests defined using the Fit framework can help in clarifying requirements [79].

Observation 20. *Test-driven development is not frequently used and it seems to have an adverse effect on implementing all requirements within time.*

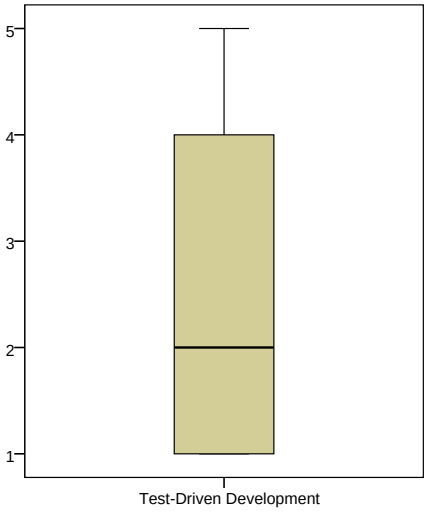

Figure 11: Degree to which Test-Driven Development is applied (1 = not at all, ..., 5 = completely).

		Application of the V model
Reaching all requirements	Correlation Significance (2-tailed)	-.186 .032

Table 16: Kendall's tau correlation between application of Test-Driven Development and fullfilling all requirements.

Figure 11 shows that although there is quite some variation in the application of Test-Driven Development, as evidenced by the spread in answers on the 5-point Likert scale, the median value remains at 2, indicating that Test-Driven Development is not frequently used amongst our respondents. Furthermore, we also observed a slight negative correlation between delivering all

requirements and applying Test-Driven Development, indicating that Test-Driven Development might actually slow down development speed, maybe at the cost of increased quality.

In a study at IBM, Maximilien and Williams observed that the extra effort to implement TDD is marginal, contradicting our observation [80].

Similarly, Erdogmus et al. performed a controlled experiment in which they observed that on average the student participants appeared to be more productive when applying a test-first strategy [81]. One of the underlying reasons for this phenomenon that Erdogmus et al. identified is that fact that requirements are better understood when first writing tests.

Contrasting the previous two studies is a study that George and Williams performed with 24 professional pair programmers. One group developed a small Java program using TDD, while the other (control) group used a waterfall-like approach [82]. Their observations are that TDD programmers produce higher quality code because they passed 18% more functional black-box test cases. However, the TDD programmers took 16% more time. This is in sharp contrast with the previous two studies and this result is more in line with the results of our survey.

Observation 21. *A lot of focus is put on the quality of requirements, in particular:*

- *60% of the respondents check the requirements for completeness*
- *56% of the respondents check the requirements for unambiguity*
- *54% of the respondents check the requirements for consistency*
- *32% of the respondents check the requirements for adherence to legislation*

Observation 22. *Focussing on the quality of the requirements seems to have positive effects on the satisfaction of the customer, the satisfaction of developers towards the development process and the product and the successful completion of the project within time.*

As Table 17 shows there is a significant positive correlation between checking the requirements and customer satisfaction, product satisfaction, satisfaction with the development process and reaching the requirements.

		Checking requirements against ambiguity	Checking requirements against consistency	Checking requirements against completeness
Satisfaction of customer	Correlation Significance	.180 .049	.263 .003	.259 .004
Satisfaction with product	Correlation Significance	.181 .045	.259 .003	.246 .006
Satisfaction with process	Correlation Significance	.302 .001	.257 .003	.182 .039
Requirements reached	Correlation Significance	.234 .009	.312 .000	.324 .000

Table 17: Correlations of checking the requirements

A possible explanation of why requirements are important for both customers and members of the development team can be sought with Bjarnason et al.; they established that communication is essential for software development [83]. In particular, they argue that efficient communication is a key factor in developing and releasing successful software products. Requirements are the single most important vehicle for enabling communication during development. Through a semi-structured interview with nine practitioners they found that communication gaps lead to failure to meet the customers' expectations and quality issues. Furthermore, it was also observed by Nuseibeh and Easterbrook that *validating* the requirements, i.e., the process of establishing that the requirements and models elicited provide an accurate account of stakeholder requirements, is very important [84].

Comparing these observations with the results of our survey, we can observe significant correlations:

- **Checking for requirements ambiguity** correlates with customer satisfaction, satisfaction of the developers with both the process and the product and the successful implementation of all requirements.
- **Checking for requirements consistency** correlates with customer satisfaction, satisfaction of the developers with both the process and the product and the successful implementation of all requirements.
- **Checking for requirements completeness** correlates with customer satisfaction, satisfaction of the developers with both the process and the

product and the successful implementation of all requirements (both within time and within budget).

Looking at our results (see Table 17), we can say that validating the requirements seems to pay out clearly with a heightened user satisfaction. This is in line with observations from Hofmann and Lehner [85], who state that requirements engineering is a critical success factor in software projects. Within the requirements engineering process, validating the requirements, i.e., certifying that requirements meet the stakeholders' intentions, is deemed very important. This contrasts an observation that Hofman and Lehner made during a field study: requirement engineering teams focus significantly more on eliciting and modelling requirements than on validating and verifying them. Hofman and Lehner also note that more than half of the projects perform peer reviews or walk-throughs as a means to verify or validate requirements [85].

Concerning the observation that 32% of the respondents of our survey check the requirements against the legislation, we refer to the survey performed by Otto and Antoón [86]. Their survey confirms that regulations and legislation are playing an increasingly important role in requirements engineering and system development. Given the criticalness of more and more software systems, we can only estimate that legislation will become ever more important in requirements engineering in the future.

8. Discussion & Research Agenda

8.1. Discussion

With this survey we aimed to get an initial insight into the current state of the practice of software developing organizations in the Netherlands. In total, we had 99 respondents for our survey that tried to cover all phases of the software development lifecycle, going from requirements engineering to quality assurance. Both the survey and the (anonymized) answers can be found online[5].

We analyzed the data that we thus obtained and from our analysis we distilled 22 *observations*. These observations mark insights into the development process that we found "unusual", or *surprising*, at least from an academic point of view. This unusualness could either stem from certain

[5] http://swerl.tudelft.nl/bin/view/Main/SurveyDutchSoftwareIndustry

principles being applied less or more frequently than one would expect, or from unexpected correlations that were observed between factors.

Each of these 22 observations in themselves warrant further research, but a number of observations – or combinations of observations – seem to touch important topics that would, ideally, require immediate attention. We list these in Section 8.3, in which we present our research agenda. Before presenting our research agenda, we first discuss possible threats to the validity of our study in Section 8.2.

8.2. Threats to Validity

Internal validity. While we have tried to formulate our questions as neutral as possible, the topic of a question, e.g., "To what extent do you perform unit testing", might have inclined respondents to answer more positively, resulting in leading question bias. We tried to avoid this by specifically asking the respondents to answer truthfully and formulate an answer for the latest project they have been working on.

Construct validity. An important threat in this department is that the factor *"satisfaction of the client"*, as used in Observation 5 is not measured directly, i.e., by asking the client, but indirectly, i.e., by asking the software developer or team leader. There might be a difference in perception between both stakeholders. We have no immediate strategy to mitigate this issue, as the survey research approach that we took did not allow us to raise this question with the client. We regard gauging the level of satisfaction of software project results and subsequently comparing these gauges as an interesting avenue of future research.

External validity. Generalizability of our observations is hard to claim, as our population of respondents was restricted to the Dutch software engineering industry. However, through an open call (see Section 2.2 for details), we managed to gather 99 respondents, working for at least 40 different organizations (some respondents chose not to fill in their current employer). We have no way of claiming that the Dutch software engineering industry itself is representative for how software is produced in general. A mitigating factor is certainly the high quality education available for IT specialists in the Netherlands, but confounding factors remain, e.g., cultural aspects that could explain why certain technologies get adopted more quickly or more slowly.

8.3. Research Agenda

We now present our research agenda that we have distilled from analyzing and interpreting the results of our survey of the Dutch software industry. While we are by no means saying that these are the only issues that should be addressed, we do think that these issues are relevant for practitioners and could have immediate effect.

We do want to make two notes before we present our research agenda. Firstly, please be aware that not each of these points on the research agenda is about performing additional research. In some cases, we, as academia, should also make sure that existing insights are transferred in a better way to our students, the software engineers of tomorrow. As such, our students should be better aware of some of the benefits from the proposed approaches or become more fluent in some of the latest advances. Secondly, the research agenda proposed below is not only about design research, it is also about performing strong empirical research, in which academia tries to get insight into why and how certain proposed methods improve for example efficiency. Along the way, this empirical research can also help to convince managers of the benefits (or drawbacks) of some approaches.

A.1: Investigating better tool-support for requirements engineering and traceability from requirements When we combine the insights that we obtained from Observation 8, "Traceability between requirements and software design seems important, but not to everyone", and Observation 6, "requirements are mainly specified in plain text, using word processors", we started wondering whether better tool support for requirements engineering would actually drive software engineering teams away from purely textual requirements specifications. Additionally, these tools should, to our opinion, also contain facilities to improve traceability from requirements to other artifacts, because this would greatly help other software engineering activities. A prime example here being *impact analysis*, or determining what the impact of a change request on a particular feature would be. In their work, Cheng and Atlee also made notice of the fact that "if the transition between requirements engineering tasks and other development tasks were more seamless, management would view requirements engineering efforts more positively, because the resulting requirements knowledge and artifacts would make more concrete progress towards achieving downstream milestones" [26].

A.2: Methods for validating and verifying requirements Supported by Observation 22 which indicates that focussing on the quality of the require-

ments seems to have positive effects on he satisfaction of the customer, the satisfaction of developers towards the development process and the product and the successful completion of the project within time, we started investigating whether there are methods for attaining high-quality requirements. As Hofmann and Lehner point out, methods for validating and verifying requirements are relatively scarce [85]. The main approaches for validating and verifying requirements are peer reviews, inspections, walk-throughs, and scenarios, but to the best of our knowledge, there are no structured methods. Additionally, as also pointed out by Hofmann and Lehner, it would be useful to not only record decisions with regard to requirements, but also their rationales, as this would become useful during validation.

A.3: The value of refactoring Connected to Observation 12, in which we observed that the application of refactoring and unit testing do not correlate, we are interested in knowing what the *value* of refactoring is. More precisely, how does it improve the code and what is the ultimate return on investment (ROI) of performing a refactoring. From their field study at Microsoft Kim et al. note: *"The value of refactoring is difficult to measure. How do you measure the value of a bug that never existed, or the time saved on a later undetermined feature? How does this value bubble up to management? Because there is no way to place immediate value on the practice of refactoring, it makes it difficult to justify to management.*

A.4: The value of unit tests during refactoring Related to our previous item on our research agenda and given the focus that refactoring literature puts on the value of having unit tests available during refactoring, we also think it is worthwhile to investigate what unit tests bring into the refactoring process. Questions that deserve attention in future research are whether unit tests serve as documentation during refactoring, whether unit tests prevent regressions and what extra cost there is to maintaining unit tests during refactoring.

A.5: Measuring stakeholder satisfaction We already noted before that we suspect that stakeholder satisfaction is quite different when measured at the client side or at the software engineering side. As such, in order to better understand both parties it would be a worthwhile exercise to determine the main commonalities and divergences in the satisfaction of each of the stakeholders in a software project. Having such knowledge would certainly serve the software engineering community with a better sense of how client

satisfaction can be improved.

A.6: Evidence on Test-Driven Development In Observation 20 we established that test-driven development is not frequently used and it seems to have an adverse effect on implementing all requirements within time. While we can understand that test-driven development involves extra steps during the development process, there have been suggestions that test-driven development improves the structure of the software [87], thereby reducing the need to refactor later on, i.e., reducing the *technical debt*. In this context, it would indeed be interesting to see how test-driven development might actually have an initial cost, which is paid back later on.

Acknowledgements

This work was funded by the RAAK-PRO project EQuA (Early Quality Assurance in Software Production) of the Foundation Innovation Alliance, the Netherlands. Our gratitude goes out to all participants of our survey.

References

[1] P. Naur, B. Randell, Software Engineering: Report of a conference sponsored by the NATO Science Committee, Scientific Affairs Division, NATO, Brussels, 1969.

[2] M. Zelkowitz, R. Yeh, R. Hamlet, J. Gannon, V. Basili, Software engineering practices in the US and Japan, Computer 17 (6) (1984) 57–66.

[3] J. Ludewig, Software engineering in the years 2000 minus and plus ten, in: R. Wilhelm (Ed.), Informatics, Vol. 2000 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2001, pp. 102–111.

[4] D. Reifer, Is the software engineering state of the practice getting closer to the of the art?, IEEE Software 20 (6) (2003) 78–83.

[5] L. J. Osterweil, A future for software engineering?, in: Future of Software Engineering (FOSE), IEEE Computer Society, 2007, pp. 1–11.

[6] K. Beckman, N. Coulter, S. Khajenoori, N. Mead, Collaborations: closing the industry-academia gap, IEEE Software 14 (6) (1997) 49–57.

[7] T. C. Lethbridge, J. Diaz-Herrera, R. J. J. LeBlanc, J. B. Thompson, Improving software practice through education: Challenges and future trends, in: Future of Software Engineering (FOSE), IEEE Computer Society, 2007, pp. 12–28.

[8] D. Rombach, R. Achatz, Research collaborations between academia and industry, in: Future of Software Engineering (FOSE), IEEE Computer Society, 2007, pp. 29–36.

[9] E. Towell, B. Thompson, A further exploration of teaching ethics in the software engineering curriculum, in: Proceedings of the 17th Conference on Software Engineering Education and Training (CSEET), IEEE Computer Society, 2004, pp. 39–44.

[10] E. Babbie, The practice of social research, Wadsworth Belmont, 2007, 11th edition.

[11] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: F. Shull, J. Singer, D. I. K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer London, 2008, pp. 285–311.

[12] B. A. Kitchenham, S. L. Pfleeger, Personal opinion surveys, in: F. Shull, J. Singer, D. I. K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer London, 2008, pp. 63–92.

[13] T. Mens, Introduction and roadmap: History and challenges of software evolution, in: T. Mens, S. Demeyer (Eds.), Software Evolution, Springer, 2008, pp. 1–11.

[14] M. Sliger, Bridging the gap: Agile projects in the waterfall enterprise, Better Software (2006) 26–31.

[15] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review, Information and Software Technology 50 (9-10) (2008) 833–859.

[16] C. Hansson, Y. Dittrich, B. Gustafsson, S. Zarnak, How agile are industrial software development practices?, Journal of Systems and Software 79 (9) (2006) 1295–1311.

[17] K. Petersen, C. Wohlin, A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case, Journal of Systems and Software 82 (9) (2009) 1479–1490.

[18] K. Petersen, C. Wohlin, D. Baca, The waterfall model in large-scale development, in: Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES), Vol. 32 of Lecture Notes in Business Information Processing, Springer Verlag, 2009, pp. 386–400.

[19] L. Layman, L. Williams, L. Cunningham, Exploring extreme programming in context: An industrial case study, in: Proceedings of the Agile Development Conference, IEEE Computer Society, 2004, pp. 32–41.

[20] P. A. Laplante, C. J. Neill, The demise of the waterfall model is imminent and other urban myths, ACM Queue 1 (10) (2004) 10–15.

[21] B. Boehm, Get ready for agile methods, with care, IEEE Computer (2002) 64–69.

[22] A. Cockburn, J. Highsmith, Agile software development: The people factor, Computer 34 (11) (2001) 131–133.

[23] G. Melnik, F. Maurer, Comparative analysis of job satisfaction in agile and non-agile software development teams, in: Proceedings of the International Conference on Extreme Programming and Agile Processes in Software Engineering (XP), Vol. 4044 of LNCS, Springer Verlag, 2006, pp. 32–42.

[24] O. Murru, R. Deias, G. Mugheddue, Assessing xp at a european internet company, IEEE Software 20 (3) (2003) 37–43.

[25] B. Nuseibeh, S. M. Easterbrook, Requirements engineering: a roadmap, in: Proceedings of the International Conference on on Software Engineering (ICSE) — Future of SE Track, ACM, 2000, pp. 35–46.

[26] B. H. C. Cheng, J. M. Atlee, Research directions in requirements engineering, in: Proceedings of the International Conference on Software Engineering (ICSE) — Workshop on the Future of Software Engineering (FOSE), IEEE Computer Society, 2007, pp. 285–303.

[27] A. van Lamsweerde, Requirements engineering in the year 00: a research perspective, in: Proceedings of the international conference on Software engineering (ICSE), ACM, 2000, pp. 5–19.

[28] T. Bell, T. Thayer, Software requirements: Are they really a problem?, in: Proceedings of the International Conference on Software Engineering (ICSE), IEEE Computer Society, 1976, pp. 61–68.

[29] K. Holtzblatt, H. R. Beyer, Requirements gathering: the human factor, Commun. ACM 38 (5) (1995) 31–32.

[30] F. Paetsch, A. Eberlein, F. Maurer, Requirements engineering and agile software development, in: International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), IEEE Computer Society, 2003, pp. 308–313.

[31] L. Cao, B. Ramesh, Agile requirements engineering practices: An empirical study, IEEE Software 25 (1) (2008) 60–67.

[32] A. De Lucia, A. Qusef, Requirements engineering in agile software development, Journal of Emerging Technologies in Web Intelligence 2 (3) (2010) 212–220.

[33] A. Eberlein, J. C. S. do Prado Leite, Agile requirements definition: A view from requirements engineering, in: Proceedings of the International Workshop on TimeConstrained Requirements Engineering (TCRE), 2002.

[34] A. Sillitti, M. Ceschi, B. Russo, G. Succi, Managing uncertainty in requirements: a survey in documentation-driven and agile companies, in: 11th IEEE International Symposium on Software Metrics, IEEE Computer Society, 2005, p. 17.

[35] A. von Knethen, Change-oriented requirements traceability. support for evolution of embedded systems, in: Proceedings of the International Conference on Software Maintenance (ICSM), IEEE Computer Society, 2002, pp. 482–485.

[36] R. Mugridge, W. Cunningham, Fit for Developing Software: Framework for Integrated Tests, Prentice Hall, 2005.

[37] H. van Vliet, Software Engineering: Principles and Practice, 3rd Edition, Springer, 2008.

[38] G. Booch, Object-Oriented Analysis and Design with Applications, Addison-Wesley, 2011.

[39] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, SIGSOFT Softw. Eng. Notes 17 (4) (1992) 40–52.

[40] F.P. Brooks, Jr., The Mythical Man-Month, Addison-Wesley, 1972.

[41] C. Hofmeister, R. Nord, D. Soni, Applied Software Architecture, Addison-Wesley, 1999.

[42] M. Shaw, D. Garlan, Software architecture: perspectives on an emerging discipline, Prentice Hall, 1996.

[43] S. Lauesen, User Interface Design, a Software Engineering Perspective, Addison Wesley, 2005.

[44] H. H. Smith, On tool selection for illustrating the use of UML in system development, Journal of Computing Sciences in Colleges 19 (5) (2004) 53–63.

[45] B. Dobing, J. Parsons, How UML is used, Communications of the ACM 49 (5) (2006) 109–113.

[46] D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, R. Pretorius, Empirical evidence about the UML: a systematic literature review, Software: Practice and Experience 41 (4) (2011) 363–392.

[47] M. Cherubini, G. Venolia, R. DeLine, A. J. Ko, Let's go to the whiteboard: how and why software developers use drawings, in: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI), ACM, 2007, pp. 557–566.

[48] T. D. LaToza, G. Venolia, R. DeLine, Maintaining mental models: a study of developer work habits, in: Proceedings of the International Conference on Software Engineering (ICSE), ACM, 2006, pp. 492–501.

[49] A. de Lucia, R. Oliveto, G. Tortora, Assessing IR-based traceability recovery tools through controlled experiments, Empirical Sotware Engineering 14 (1) (2009) 57–92.

[50] M. Lindvall, K. Sandahl, Traceability aspects of impact analysis in object-oriented systems, Journal of Software Maintenance: Research and Practice 10 (1) (1998) 37–57.

[51] A. Nugroho, M. R. Chaudron, A survey into the rigor of UML use and its perceived impact on quality and productivity, in: Proceedings of the international symposium on Empirical software engineering and measurement (ESEM), ACM, 2008, pp. 90–99.

[52] C. Lange, M. Chaudron, J. Muskens, In practice: UML software architecture and deign description, IEEE Software 23 (2006) 40–46.

[53] C. Boogerd, L. Moonen, Evaluating the relation between coding standard violations and faultswithin and across software versions, in: Proceedings of the Working Conference on Mining Software Repositories (MSR), IEEE Computer Society, 2009, pp. 41–50.

[54] M. Paulk, Extreme programming from a CMM perspective, IEEE Software 18 (6) (2001) 19 –26.

[55] K. Beck, Extreme programming explained: embrace change, Addison-Wesley Professional, 2000.

[56] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, MA, USA, 1999.

[57] A. Van Deursen, L. Moonen, The video store revisited–thoughts on refactoring and testing, in: Proc. 3rd Intl Conf. eXtreme Programming and Flexible Processes in Software Engineering, 2002, pp. 71–76.

[58] M. Kim, T. Zimmermann, N. Nagappan, A field study of refactoring challenges and benefits, in: Proceedings of the International Symposium on Foundations of Software Engineering (FSE), ACM, 2012, to appear.

[59] A. Mohammad, Empirical investigation of refactoring effect on software quality, Information and Software Technology 51 (9) (2009) 1319–1326.

[60] W. Basit, F. Lodhi, U. Bhatti, Extending refactoring guidelines to perform client and test code adaptation, in: A. Sillitti, A. Martin, X. Wang, E. Whitworth, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, C. Szyperski (Eds.), Agile Processes in Software Engineering and Extreme Programming, Vol. 48 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2010, pp. 1–13.

[61] V. Hurdugaci, A. Zaidman, Aiding developers to maintain developer tests, in: Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR), IEEE Computer Society, 2012, pp. 11–20.

[62] A. Qusef, R. Oliveto, A. D. Lucia, Recovering traceability links between unit tests and classes under test: An improved method, in: Proceedings of the International Conference on Software Maintenance (ICSM), IEEE Computer Society, 2010, pp. 1–10.

[63] L. Williams, R. Kessler, W. Cunningham, R. Jeffries, Strengthening the case for pair programming, IEEE Software 17 (4) (2000) 19–25.

[64] T. Bipp, A. Lepper, D. Schmedding, Pair programming in software development teams — an empirical study of its benefits, Information and Software Technology 50 (3) (2008) 231–240.

[65] A. Begel, N. Nagappan, Pair programming: what's in it for me?, in: Proceedings of the international symposium on Empirical Software Engineering and Measurement (ESEM), ACM, 2008, pp. 120–128.

[66] E. Arisholm, H. Gallis, T. Dybå, D. I. K. Sjøberg, Evaluating pair programming with respect to system complexity and programmer expertise, IEEE Transactions on Software Engineering 33 (2) (2007) 65 –86.

[67] J. E. Hannay, T. Dybå, E. Arisholm, D. I. K. Sjøberg, The effectiveness of pair programming: A meta-analysis, Information & Software Technology 51 (7) (2009) 1110–1122.

[68] L. Crispin, Driving software quality: How test-driven development impacts software quality, IEEE Software 23 (6) (2006) 70–71.

[69] A. Bertolino, Software testing research: Achievements, challenges, dreams, in: Future of Software Engineering (FOSE), IEEE Computer Society, 2007, pp. 85–103.

[70] B. Beizer, Software Testing Techniques (2nd ed.), Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[71] G. J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, 3rd edition, Wiley, 2011.

[72] A. Zaidman, B. Van Rompaey, A. van Deursen, S. Demeyer, Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining, Empirical Software Engineering 16 (3) (2011) 325–364.

[73] V. Garousi, T. Varma, A replicated survey of software testing practices in the canadian province of alberta: What has changed from 2004 to 2009?, Journal of Systems and Software 83 (11) (2010) 2251–2262.

[74] R. F. Goldsmith, D. Graham, The forgotten phase, in: Software Development, 2002, pp. 45–47.

[75] M. Pol, R. Teunissen, E. van Veenendaal, Software Testing: A Guide to the TMap Approach, Addison–Wesley, 2001.

[76] IEEE, IEEE Std 1012-1986, IEEE standard for software verification and validation plans (1986).

[77] E. Uusitalo, M. Komssi, M. Kauppinen, A. Davis, Linking requirements and testing in practice, in: Proceedings of the International Requirements Engineering Conference (RE), IEEE Computer Society, 2008, pp. 265–270.

[78] R. Young, Effective Requirements Practice, Addison-Wesley, 2001.

[79] F. Ricca, M. Torchiano, M. Ceccato, P. Tonella, Talking tests: an empirical assessment of the role of fit acceptance tests in clarifying requirements, in: Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, ACM, 2007, pp. 51–58.

[80] E. M. Maximilien, L. A. Williams, Assessing test-driven development at ibm, in: Proceedings of the International Conference on Software Engineering (ICSE), IEEE Computer Society, 2003, pp. 564–569.

[81] H. Erdogmus, M. Morisio, M. Torchiano, On the effectiveness of the test-first approach to programming, IEEE Transactions on Software Engineering 31 (3) (2005) 226–237.

[82] B. George, L. Williams, A structured experiment of test-driven development, Information and Software Technology 46 (5) (2004) 337–342.

[83] E. Bjarnason, K. Wnuk, B. Regnell, Requirements are slipping through the gaps — a case study on causes & effects of communication gaps in large-scale software development, in: Proceedings of the International Requirements Engineering Conference (RE), IEEE Computer Society, 2011, pp. 37–46.

[84] B. Nuseibeh, S. Easterbrook, Requirements engineering: a roadmap, in: Proceedings of the Conference on The Future of Software Engineering (ICSE), ACM, 2000, pp. 35–46.

[85] H. Hofmann, F. Lehner, Requirements engineering as a success factor in software projects, IEEE Software 18 (4) (2001) 58–66.

[86] P. Otto, A. Anton, Addressing legal requirements in requirements engineering, in: Proceedings of the International Requirements Engineering Conference (RE), IEEE Computer Society, 2007, pp. 5–14.

[87] K. Beck, Test-Driven Development: By Example, Addison-Wesley, 2003.

TUD-SERG-2012-022
ISSN 1872-5392