Truth and Myth of Independent Software Testing – A Controlled Human Experiment

Ali Sunyaev
University of Cologne
Albertus-Magnus-Platz
D-50923 Cologne
+492214705379
sunyaev@wiso.uni-koeln.de

Dirk Basten
University of Cologne
Albertus-Magnus-Platz
D-50923 Cologne
+492214705370
basten@wiso.uni-koeln.de

ABSTRACT
Organizations often outsource software testing activities to optimize software testing and verify developed software by independent experts. Related benefits include expectations for higher software quality and anticipated commercial benefits, but have not been empirically evaluated yet. To close this gap, we hypothesize independent software testing to positively impact software quality and test our hypothesis in a controlled human experiment. We investigate the functional quality of six functionally identical systems developed in software projects by six teams of five students each. Of these six teams, three tested their software themselves, while the others used independent testing teams. On system-testing level, we compare the fulfillment of predefined software functions by applying function point analysis. The results show that independent software testing has a positive impact on software quality. Using t-tests, we show functional completeness of software tested by independent testers to be significantly higher (16%) than the functional completeness of software tested by software developers themselves. Our work provides insights into decision-making on software test outsourcing. While the expected higher quality of the products is corroborated by the experiment, there is a need for further research studies. With our experiment, we raise practitioners' awareness to consciously deal with independent software testing.

Categories and Subject Descriptors
D.2 [**Software Engineering**]: Software/Program Verification – *validation, statistical methods.* Management – *software quality assurance (SQA).* Metrics – *product metrics*.

General Terms
Experimentation, Verification, Measurement, Human Factors.

Keywords
Independent software testing, software testing process, software development, external software testing centers, controlled human experiment, software quality.

SAC'15, April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695684

1. INTRODUCTION
Despite being an important activity of any software development, software testing is mostly not considered a core competency of a software development organization [6]. In this context, software development organizations increasingly outsource their testing activities to optimize software testing with regard to effectiveness and efficiency as well as to verify developed software by independent testing experts. For these purposes, many software testing centers exist which provide external and thus independent testing services. While the software testing market has been estimated to €50 billion in 2011 and is expected to continually grow [17], outsourcing of testing processes is likely to be an emerging trend in the software industry.

On the one hand, outsourcing testing activities to specialized centers is supposed to offer software developing organizations significant financial benefits [6, 14, 23]. Such benefits include reduced testing costs due to daily rate cutbacks, reduced testing duration, improved resource utilization, and saving procurement costs for specialized testing environments. Furthermore, the technical expertise and high reputation of independent testers create additional trust in the developed software.

On the other hand, software developing organizations face challenges in integrating external testers in their software development processes regarding the coordination of tasks and the communication between software developers and independent testers. Jones et al. [12] identified key problems that constantly recur in software development projects where external testers are involved. The authors found that neither software developers nor external testers are able to evaluate how well the software has been tested. Furthermore, communication between testers and developers is too inefficient to quickly and adequately address testing-related issues. As a result, the management of error reporting and error correcting is negatively impacted. These problems can interfere with the quality of the tested software. This is an important aspect as independent software testing (the same of course applies to software testing generally) should not be an end in itself, but it should contribute to improving software quality. Depending on the context, poor or 'simply good' quality is not sufficient. Software used for financial transactions may, for instance, lose its benefits if the quality is flawed.

So far, the impact of independent software testing on software quality has not been systematically evaluated (cf. the overview of related studies in Section 2). To close this gap, this study examines the effect of independent software testing on software quality in a human controlled experiment. We investigate whether software quality tested by independent testers is higher compared

to software quality tested by software developers themselves In order to evaluate software quality, we use the quality model of the standard ISO/IEC 9126-1 consisting of six quality characteristics: functionality, reliability, usability, efficiency, maintainability, and portability [12]. As a first step, we focus the evaluation on *functionality* as quality characteristic due to its measurability and comparability.

The remainder of this paper is structured as follows. We use Section 2 to outline related work concerning independent software testing. We describe the experiment setting based on Wohlin et al. [24] and discuss related threats to validity in Section 3. In Section 4, we present the experiment results. We discuss our findings and their relevance for research and practice in Section 5. Finally, we conclude our study in Section 6.

2. RELATED WORK

Empirical research on *independent software testing* appears scarce so far. As a first indicator, searching for the above term using Google Scholar (http://scholar.google.com), reveals 73 results only; most of them being books or being identified only because one of the authors works in the software testing industry (i.e., the publication does not necessarily cope with independent software testing).

In addition to this initial search, we systematically searched the following databases: EBSCOHost, ScienceDirect, ProQuest, IEEE Explore, ACM Digital Library, and AISel (electronic library of the Association for Information Systems). For article identification, we used different keyword combinations. Our search pattern needed to match the terms *independent* OR *autonomous* OR *outsourcing* OR *offshore* AND the term *software* AND the terms *testing* OR *evaluation* OR *agency* in title, abstract, or keywords. We thereby used variations of the wording (e.g., *outsourcing* and *outsourced*) to gather a representative set of publications.

The few identified studies deal with diverging topics, widely using qualitative research approaches. The topics include managers' responsibilities in outsourced software-testing projects [11], the influence of culture on outsourced software-testing practice [21], and client communication practices [18]. Further case studies focus on the relation between software development organizations and independent test agencies [12] and the challenge of bridging gaps between developers and testers in globally distributed software development [8].

Apart from the above, a number of publications using controlled experiments dealing with software testing in general exist (e.g., [1, 5, 15, 16, 20]). However, these do not cover independent software testing.

However, our search did not yield any publication that uses an experiment to evaluate independent software testing. Thus, to the best of our knowledge we are not aware of any experiment concerning independent software testing investigating an effect between independent software testing and software quality.

3. EXPERIMENTAL SETTING

We used the scoping and planning steps described by Wohlin et al. [24] to set up our experiment: defining the experiment goal, selecting the context, formulating hypothesis, selecting variables, and designing and instrumenting the experiment.

3.1 The Experiment Goal Definition

In accordance with the goal definition framework [2, 4], we define the goal of our experiment as follows:

- **Object of study:** Independent software testing
- **Purpose:** Evaluate
- **Quality focus:** Software quality
- **Perspective:** Researcher
- **Context:** Software developed within a programmer course

3.2 The Experiment Context

We conducted the experiment with advanced students from two German universities during the summer term in 2012. The experiment was embedded in a laboratory software development course with information system students at one of the oldest, largest, and best-ranked universities in Germany (University A). Advanced information system students in their last year of university studies developed and tested the software. We cooperated with advanced students from a second German university (University B) who served as independent testers for this software development. To enable a common ground, the experiment participants attempted a training course for several chosen testing techniques of the International Software Testing Qualifications Board (ISTQB; cf. for instance [23]). Testing specialists from an independent software testing center ensured that students from both universities were equally prepared to test the software.

The software tested was a real-world project for the faculty's deanship. It was a web-based application to manage the business school's research activities. In order to ensure identical software functionality, the requirements specification was not changed during the entire experiment. Although the students were required to define the functional system requirements on their own for teaching purposes, we provided the consistent and accurately defined requirements before beginning the implementation phase. Thus, we ensured that all teams used the same requirements (a total of 65 functional requirements were specified and documented) for developing the software.

3.3 The Experiment Hypothesis

The goal of the experiment was to answer the following question:

Is the quality of the software tested by independent testers higher compared to the quality of the same software tested by integrated testers?

Software testing by external specialists is advantageous for software development organizations [6, 23] despite several challenges and issues [6, 7, 12, 23]. Provided that software developers and independent testers succeed in dealing with the challenges and issues, this testing approach can have the positive effect on software quality. Considering quality to consist of six characteristics (i.e., functionality, reliability, usability, efficiency, maintainability, and portability) [12], we in this study focus on a single characteristic as a first step. For this purpose, we choose functionality due to its measurability and objective comparability. We are thus interested in independent testing's effect on **the degree of fulfillment of the functional requirements predefined for software** (henceforth referred to as *FDegree*). Accordingly, we formulate our hypotheses H_0 and H_1 as follows:

H_1: *FDegree* of software tested by independent testers (product 1) is higher than *FDegree* of software tested by integrated testers (product 2).

H_0: *FDegree* of software tested by independent testers (product 1) is lower than or equal to *FDegree* of software tested by integrated testers (product 2).

According to our definition of software quality, we measure and compare fulfillment of predefined functions for software. We describe our detailed measurement model in section 3.4.

3.4 The Experiment Variables

In our experiment, the independent variable is the choice of a testing approach. It can be one of two values: independent software testing by external testers and integrated software testing by software developers themselves.

The dependent variable was **quality of software developed** during the experiment. We measured *FDegrees* – degree of fulfillment of specified requirements measured in function points – of the software to assess their quality (the higher the degree, the higher the software quality). Using Function-Point-Analysis (FPA) [10] to determine *FDegree*, we calculate the total functional size $TFSize_x$ of each specified function x by decomposing the functions in basic logic components $BLCy_x$. Each $BLCy_x$ was rated with function points. Hence, we were able determine $TFSize_x$ as $\sum BLCy_x$. The total functional size of software was the sum of $TFSize_{spec}$: $\sum TFSize_x$.

To test our hypotheses, we compare of software quality (system-testing level) tested by external and integrated testers. Therefore, we calculate the *FDegree* of each software component. Accordingly, the functional size $IFSize_x$ per software component of each implemented function x was calculated for each team. Each implemented function x was tested with the aid of predefined functional test cases and weighted its $TFSize_x$ with *0* for an unavailable / not operational function, with *0.5* for an operational function with defects or with *1* for a complete operational function. The implemented functional size of software was the sum of $IFSize_{product}$: $\sum IFSize_x$.

With $IFSize_x$ and $TFSize_x$ it was possible to calculate the functional completeness of the product delivered – *FDegree* – as follows:

$$FDegree = \frac{\sum_{x=1}^{n} IFSize_x}{\sum_{x=1}^{n} TFSize_x}$$

3.5 The Experiment Design

Concerning this development, we built two groups and randomly assigned 15 participants to each group (based on a set of 30 students at University A).

Prior to the course, we taught the experiment participants how to use the programming environment to develop the software. We surveyed the participants about their programming experience using a four-point scale: beginner (< 1 year), intermediate (1-3 years), advanced (3-5 years) and expert (> 5 years). The experience refers to the programming language Java, which was the language the students had to use to develop the software system.

Based on their programming experience, we in both groups built three small teams – five students per team – having the same average programming experience. We attempted to minimize the differences in the programming experience across the developer teams as far as possible to ensure equally developed software products. To ensure an almost equal composition of the teams and thus an adequate initial situation for all teams, we thoroughly assigned the students to teams (cf. Table 1).

3.5.1 Software development.

Each of the six teams developed software based on identical functional requirements. The teams passed through all software development phases of conventional software development: system initiation, requirement analysis and specification, preliminary design, detailed design, implementation and integration [19]. The overall software development took 16 weeks. Figure 1 provides the detailed scheduling of this process.

3.5.2 Software testing.

After implementation, the software products were completed and deployed to the test infrastructure. While testing was performed in different releases (cf. Figure 1), we in this study evaluate quality of the final releases as we wanted the students to experience training effects within the first three partial releases.

For software testing, we made a distinction between the six teams from University A (cf. Table 1). While teams 1-3 tested their developed software themselves, teams 4-6 gave their developed software products to external testing team (15 students from University B; five testers per software component). To set equal conditions, both independent testers and integrated testers had the same period of time. As a systematic approach, the teams applied functional black-box tests using equivalence partitioning and boundary value analysis. Whereas the teams did not have support in form of a professional testing tool, the teams were free to choose general IT tools to manage their testing efforts (e.g., Microsoft Excel). The defects found were reported via an online bug tracking system. In total, three test iterations were scheduled to test the final software during the implementation phase (cf. Figure 1).

3.5.3 Statistical design.

In order to receive a statistically evaluable data set, we designed our experiment in accordance with the three general design principles described by Wohlin et al. [24]: randomization, blocking, and balancing.

In our case, the principle of randomization was preserved by randomly dividing the experiment participants into two large groups that pursued two different testing approaches, that is, two treatments of the independent variable (cf. section 3.2).

The controlled blocking factor was whether the students were involved in developing the software. Those that were involved were assigned to the internal testing teams. Those that were not involved were assigned to the external testing teams. This does not contradict the principle of randomization, because, as mentioned above, the assignment to each treatment was randomized as well. Our experiment design was balanced as we used the same number of participants in each team.

As a consequence of our experiment definition and selection of the experiment variables, the type of our experiment design is one factor with two treatments. The factor was the testing approach and the treatments were independent software testing by external testers and software testing by integrated testers.

3.6 Threats to Validity

We used students as experiment participants. On the one side, concerning the external validity of our experiment it would be better to run it in a large real software development project with professionals. However, it is almost impossible to find many small professional developer teams which would take part in such an experiment with implementing the same software under similar conditions in a real-world setting.

Although the experiment participants have on average up to three years of programming experience, we acknowledge that they cannot have the same longtime programming and testing experience like professionals. On the other side, students can be good experiment subjects in software engineering experimentation. They are approximately of the same age as software professionals and have practical experience. They are easy to access and relatively cheap to use [3, 9]. Berander [3] conducted an experiment to evaluate the conditions under which

Table 1. Programming experience across teams

	Team	Team Member 1	Team Member 2	Team Member 3	Team Member 4	Team Member 5
Integrated testing teams	Team 1	beginner	intermediate	intermediate	intermediate	advanced
	Team 2	beginner	intermediate	intermediate	intermediate	advanced
	Team 3	beginner	intermediate	intermediate	intermediate	advanced
Independent testing teams	Team 4	beginner	intermediate	intermediate	intermediate	intermediate
	Team 5	beginner	intermediate	intermediate	intermediate	advanced
	Team 6	intermediate	intermediate	intermediate	intermediate	expert
beginner (< 1 year), intermediate (1-3 years), advanced (3-5 years), and expert (> 5 years).						

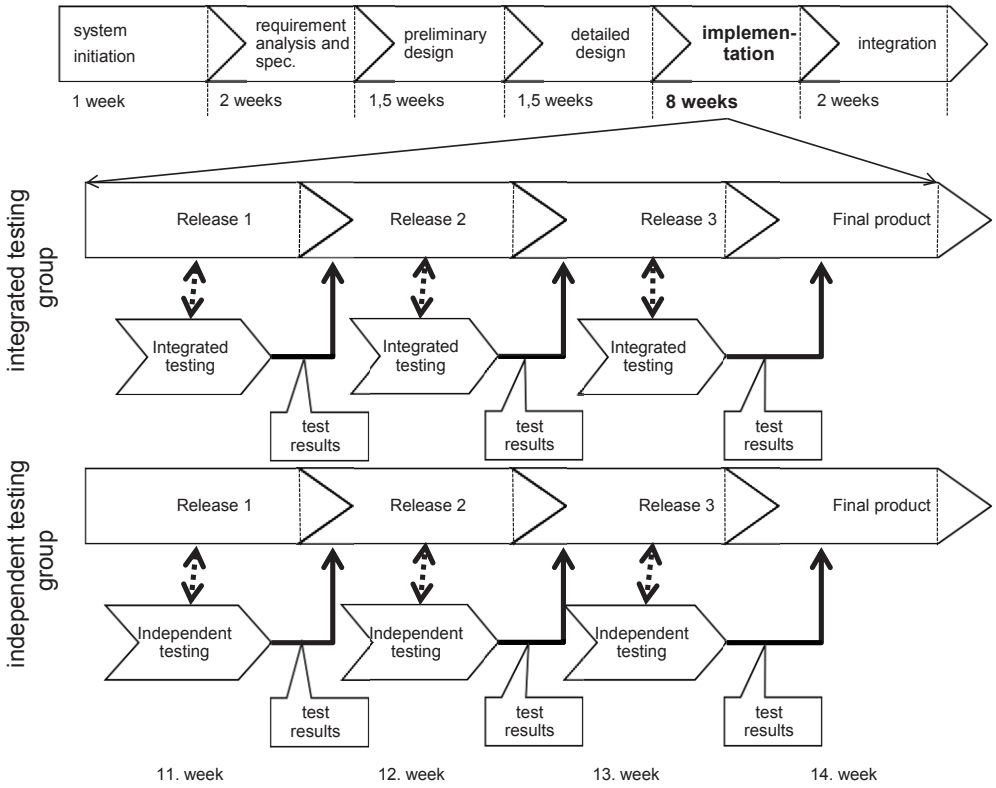

Figure 1. Test iterations during the implementation phase.

students could be used as subjects in experimentation. The experiment results indicate that in project work students are almost suitable as representatives for professionals. Höst et al. [9] conducted a comparative study of students and professional software developers to evaluate the validity of studies using students instead of professionals. They found that the results of these studies can be generalized under certain conditions: first, students are last-year software engineering students and second, performed tasks are in the context of maturity and understanding of dependencies and relationships in software engineering [9]. Both conditions are fulfilled in our setting.

Considering the experience, we were unable to assess the impact of the partially unequal distribution across the six teams. For instance, Team 6 is the only with an expert team member. This higher experience might have had an effect concerning the performance of this team. While the teams were instructed to equally share the testing effort, we cannot guarantee that work within the teams has not been disproportionately shifted to rather (un-)experienced team members, thus biasing the average team experience.

The systems were very complex and the development took four months. Conducting a similar experiment with professional developers was not possible due to financial limitations. For example, based on an average hourly wage of a software developer / software tester of €50, labor costs for the period of four months would amount to €44,800 for each professional. This would be an enormous financial effort.

Sjøberg et al. [22] argue that software engineering researchers should increase the realism of controlled software engineering experiments by setting up realistic tasks, realistic subjects, and a realistic environment. We chose the programmer course as experiment environment as this course was suitable to realize large real software projects under controlled circumstances. Additionally, we used a software project with real software requirements.

Our sample size is another limitation. Using t-tests to compare the mean values between groups requires normally distributed data. While it is unlikely for non-normality to be detected with small sample sizes of 10 observations or fewer, we had six software systems to compare, which we believe to be a representative set.

From the statistical point of view, we fulfilled all assumptions for the deployed statistical procedures. We therefore believe that the results of our study are to be considered as valid.

4. EXPERIMENT RESULTS

In the following, we present the results of (1) our function-point analysis and (2) our statistical test to falsify our hypothesis. For our analyses, we used the software IBM SPSS Statistics 21.0.

4.1 Results of Function-Point Analysis

As described above, we (i.e., the authors) rated the system functions with function points and calculated their functional size. The total functional size of the software $\sum TFSize_x$ summed up to 631 function points (cf. Table 2). The six final software systems were tested using predefined functional test cases provided by the external testing company, which was also in charge of the student training. Moreover, the respective *FDegrees* were calculated. At this point, we did not compare the predefined test cases to the ones developed by the teams since this was not in the focus of our study. The summary of the calculation is provided in Table 3.

Table 2. Function-Point-Analysis.

Complexity	count of functions	$\sum TFSize_x$
Functions with low complexity ($TFSize_x$ <= 5 FP)	13	53
Functions with average complexity (5 FP < $TFSize_x$ <= 10 FP)	30	293
Functions with high complexity ($TFSize_x$ > 10 FP)	22	285
	65	**631**

Here, the *counts of complete operational functions* (CCOF) in products tested by independent testers (products 4, 5 and 6) are higher than or equal to the CCOF in products tested by integrated testers (products 1, 2, and 3). The reverse effect can be observed in the *counts of operational functions with defects* (COFD). However, the groups differ in the *counts of unavailable / not operational functions* (CNOF): product 5 has more unavailable / not operational functions than products 1 and 3. The comparison of products 3 and 5 is of particular interest. Both products have the same CCOF = 48. Although their sums of COFD and CNOF are equal, the sum of $IFSize_{product5}$ is greater than the sum of $IFSize_{product3}$. This difference is caused by the complexity of complete operational functions of product 5 being higher than complexity of complete operational functions of product 3. Accordingly, the *FDegree* of product 5 is greater than the *FDegree* of product 3.

Table 3. Degrees of the fulfillment of functions.

Soft-ware product	CCOF ($\sum IF\text{-}Size_x$)	COFD ($\sum IF\text{-}Size_x$)	CNOF	$\sum IF\text{-}Size_x$	*FDeg-ree*
product 1	34 (339)	23 (106.5)	8	445.5	70.60 %
product 2	36 (349)	15 (58.5)	14	407.5	64.58 %
product 3	48 (427)	10 (55.5)	7	482.5	76.47 %
product 4	51 (527)	10 (32.0)	4	559.0	88.59 %
product 5	48 (497)	8 (17.5)	9	514.5	81.54 %
product 6	57 (556)	4 (10.5)	4	566.5	89.78 %

In order to falsify our hypotheses we compared the calculated ***FDegree***s with statistical tests.

4.2 Results of Statistical Tests

4.2.1 Shapiro-Wilk-normality-test.

For testing the normality of our data, we applied a two-independent samples normality test. For this purpose, applied the Shapiro-Wilk test [13]. As illustrated in Table 4, the assumption of our data being normally distributed is corroborated.

Table 4. Results from the Shapiro-Wilk test.

treatment / dependent variable	n	W	Skewness	Sig.
integrated software testing / ***FDegree***	3	1.000	-0.038	0.986
independent software testing / ***FDegree***	3	0.856	-1.594	0.256

The *p*-values 0.986 and 0.256 being greater than the chosen significance level $\alpha = 0.05$ indicate normal data distribution.

4.2.2 Levene-test.

We conducted the Levene-test to compare the sample variances. Table 5 provides the according descriptive statistics. Our calculation shows an F value of 0.072 (cf. Table 5). Considering a significance level of 0.05, we assume the variances to be equal.

Table 5. Descriptive Statistics.

treatment / dependent variable	n	mean	std. err.	std. dev.
integrated software testing / ***FDegree***	3	70.550	3.432	5.945
independent software testing / ***FDegree***	3	86.637	2.571	4.454
combined	6	78.593	4.076	9.985
difference		16.087	4.289	
Levene's Test for Equality of Variances	F = 0.072		Sign. = 0.082	

4.2.3 t-test.

We used the two-tailed t-test [13] to compare the *FDegree*s of the software, tested by independent testers and integrated testers. Table 6 provides the according results.

Table 6. t-test for equality of mean values.

	t	Df	Sig. (2-tailed)	Mean Difference
Equal variance assumed	-3.751	4.000	0.020	-16.087
Equal variances not assumed	-3.751	3.707	0.023	-16.087

As we assume the variances to be equal (cf. section 4.2.2), the t-test for equality of means is significant ($p = 0.02 < 0.05$). As the power is rather low due to our sample size, we report the results for both variances assumed to be equal and variances not assumed to be equal. In both cases, our hypothesis can be considered to be confirmed. Concluding, we gathered evidence corroborating the hypothesis stating that the *FDegree* of software tested by independent testers is higher than the *FDegree* of software tested by integrated testers. The functional completeness of software tested by external testers is on average up by 16.1% higher than the functional completeness of software tested by integrated testers.

5. DISCUSSION

General agreement exists that software testing helps to improve software quality. The objective of software testing should be not merely to improve quality but to achieve best possible quality. It is therefore particularly important to choose the correct testing approach. The experiment results in this study indicate that independent software testing has a positive effect on functional software quality. So far, this is only one study and we should consider its limitations (cf. Section 3.6). A possible explanation for the comparably higher quality in terms of functional completeness might be the objective perspective of independent software testers. There are, what we believe, several reasons for the assumption of higher objectivity. First, there are fewer conflicts of interest between software developers and software testers since both sides do not know each other well. Second, independent testing teams might have a more thorough approach to software testing since they are not as familiar with the software as integrated testing teams (i.e., the developers themselves) and need to explore the product in-depth. Third, testers might be more eager to identify errors when they are not testing software coded by themselves. While our study indicates the superiority of independent software testing at least concerning the quality characteristic *functionality*, the findings should be scrutinized. An overall assessment should focus on other characteristics as well. In general, quality should not be the only criterion to decide whether to outsource software testing. Aspects like legal data privacy, trust, and in case of offshoring cultural difference should be regarded as well. Thus, further research is necessary to investigate the degree to which independent software testing is superior to integrated software testing. It is necessary to replicate the experiment under industrial conditions with professional software developers and testers. More case studies with software development project managers are also needed to gather their experiences with independent software testing. In this context, the experiment should also be replicated with a specific software testing team within the development company acting as independent testing team (i.e., it would be an intermediate stage between integrated software testing and independent software testing). Whereas our study focusses on sequential software development, independent software testings' impact on software quality in other context, for instance agile development, should be analyzed. We acknowledge that none of the developed software in our experiment was completed to 100% (cf. Table 3). This means that although independence in software testing helped to improve software quality, the best possible quality was not achieved in any examined system. Both CNOF (functionality not provided) and COFD (operational functions with defects) might be attributed to time pressure. The project was quite unique and we lack experience concerning software development effort. In case of underestimated effort, function may either not be implemented or insufficiently implemented. Another possible explanation could be that the cooperation between independent testers and software developers in the independent testing group was not optimal and not all reported defects were fixed. This can be derived by studying defect reports in the online bug tracking system. Most likely, this is not the only reason and other factors such as lack of programming experience probably led to non-100%-fulfillment of software as well.

6. CONCLUSION

Independent software testing has been insufficiently studied. To the best of our knowledge, no controlled experiments concerning this testing strategy exists so far. Thus, we experimentally

investigated independent software testing and examined its effect on software quality. Our findings indicate that independence has a positive impact on quality of the analyzed software with regard to their functionality. With our findings, we provide further insights and verification for making strategic decisions on independent software testing. However, there is a need for further investigations. Such studies should, for instance, deal with a holistic investigation for studying the extent to which independent software testing has an impact on other software quality characteristics like reliability, usability, efficiency. Moreover, our experiment should be replicated in an industrial context using professionals in order to increase the external validity of the experiment results. Such investigations help to improve the basis of deciding for or against independent software testing and to raise researcher`s and practitioners' awareness to deal consciously with independent software testing.

7. REFERENCES

[1] Andersson, C., Thelin, T., Runeson, P., and Dzamashvili, N. 2003. An Experimental Evaluation of Inspection and Testing for Detection of Design Faults. In Proceedings of the 2003 International Symposium on Empirical Software Engineering. IEEE Computer Society, Los Alamitos, 174-184.

[2] Basili, V. R. 1995. The Experience Factory and Its Relationship to Other Quality Approaches. Advances in Computer 41, 65–82.

[3] Berander, P. 2004. Using Students as Subjects in Requirements Prioritization. In Proceedings of the 2004 International Symposium on Empirical Software Engineering. IEEE Computer Society, Los Alamitos, 167–176.

[4] Briand, L. C., Differding, C. M., and Rombach, H. D. 1996. Practical guidelines for measurement-based process improvement. Software Process: Improvement and Practice 2, 4, 253–280.

[5] Cai, X. and Lyu, M. R. 2002. The Effect of Code Coverage on Fault Detection under Different Testing Profiles. In Proceedings of the 1st International Workshop on Advances in Model-based Testing. ACM, New York, 1–7.

[6] Desikan, S. and Ramesh, G. 2006. Software Testing: Principles and Practice. Dorling Kindersley, Bangalore.

[7] Frühauf, K., Sandmayr, H., and Ludewig, J. 2007. Software-Prüfung: Eine Anleitung zum Test und zur Inspektion. Vdf-Lehrbuch Informatik. vdf Hochschulverlag AG, Zürich.

[8] Grechanik, M., Jones, J. A., Orso, A., and van der Hoek, A. 2010. Bridging Gaps between Developers and Testers in Globally-distributed Software Development. In Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. ACM, New York.

[9] Höst, M., Regnell, B., and Wohlin, C. 2000. Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empirical Software Engineering 5, 3, 201–214.

[10] Hürten, R. 2005. Function-point analysis - Theorie und Praxis: Die Grundlage für ein modernes Softwaremanagement. Edition expertsoft 38. Expert-Verl., Renningen.

[11] Jain, R. P., Poston, R. S., and Simon, J. C. 2011. An Empirical Investigation of Client Managers' Responsibilities in Managing Offshore Outsourcing of Software-Testing Projects. IEEE Transactions on Engineering Management 58, 4, 743–757.

[12] Jones, J. A., Grechanik, M., and van der Hoek, A. 2009. Enabling and Enhancing Collaborations between Software Development Organizations and Independent Test Agencies. In Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering. IEEE Computer Society, Los Alamitos, 56–59.

[13] Marques de Sá, J. P. 2008. Applied Statistics Using SPSS, STATISTICA, MATLAB and R. Springer-Verlag, Berlin, Heidelberg.

[14] Naik, K. and Tripathy, P. 2008. Software Testing and Quality Assurance: Theory and Practice. John Wiley & Sons, Hoboken.

[15] Nascimento, L. and Machado, P. 2007. An Experimental Evaluation of Approaches to Feature Testing in the Mobile Phone Applications Domain. In Proceedings of the Workshop on Domain Specific Approaches to Software Test Automation in Conjunction with the 6th ESEC/FSE Joint Meeting. ACM, New York.

[16] Offutt, J., Ma, Y.-S., and Kwon, Y.-R. 2004. An Experimental Mutation System for Java. ACM SIGSOFT Software Engineering Notes 29, 5, 1.

[17] Pierre Audoin Consultants. 2011. Growth Market Software Testing: Market Trends, Service Providers and Success Factors.

[18] Poston, R. S., Simon, J. C., and Jain, R. P. 2010. Client Communication Practices in Managing Relationships with Offshore Vendors of Software Testing Services. Communications of the Association for Information Systems 27, 9.

[19] Royce, W. W. 1987. Managing the Development of Large Software Systems: Concepts and Techniques. In Proceedings of the 9th International Conference on Software Engineering. IEEE Computer Society, Los Alamitos, 328–338.

[20] Saff, D. and Ernst, M. 2004. An Experimental Evaluation of Continuous Testing during Development. In Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM, New York.

[21] Shah, H., Harrold, M., and Nersessian, N. 2011. Studying the Influence of Culture on Outsourced, Offshored Software-Testing Practice: An Ethnographic Approch. In Sixth IEEE International Conference on Global Software Engineering Workshop. IEEE Computer Society, Los Alamitos, 105–107.

[22] Sjøberg, D. I. K., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E., and Vokác, M. 2002. Conducting Realistic Experiments in Software Engineering. In Proceedings of the 2002 International Symposium on Empirical Software Engineering. IEEE Computer Society, Washington.

[23] Spillner, A., Linz, T., and Schaefer, H. 2011. Software Testing Foundations. O'Reilly Media, Santa Barbara.

[24] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. 2000. Experimentation in Software Engineering: An Introduction. The Kluwer international series in software engineering 6. Kluwer Academic, Boston.