Improvement of Design Specifications with Inspection and Testing

Dietmar Winkler, Bernhard Riedl, Stefan Biffl
Vienna University of Technology, Institute of Software Technology,
Karlsplatz 13, A-1040 Vienna, Austria
{ Dietmar.Winkler, Bernhard.Riedl, Stefan.Biffl } @qse.ifs.tuwien.ac.at

Abstract

Inspection and testing are common verification and validation (V&V) approaches for defect detection and removal in the software development processes. Testing approaches require executable code, typically available in later life-cycle phases. Software Inspection is a defect detection technique applicable to early life-cycle documents, e.g., during design. The Usage-Based Reading (UBR) technique approach is a structured method for inspection support.

In this paper we introduce a testing variant, usage-based testing (UBT-i) that integrates testing scenarios and inspection techniques. UBT-i is a paper based testing approach (i.e. a desk test without the need for executable software) applicable to design specifications. We present an initial empirical study on defect detection effectiveness and efficiency with respect to several defect severity classes and defect locations (code or design). Main results of the study are (a) UBR and UBT-i perform similarly regarding both effectiveness and efficiency and (b) the approaches focus on different defect classes regarding defect severity and defect location.

Key words: Verification & Validation, Software Inspection, Usage-Based Testing, Software Product Improvement.

1. Introduction

The delivery of high-quality software products is a major goal in software engineering. An important aspect is to achieve a very low number of defects in a software product based on a mature software and quality assurance development process. Following the software life cycle approach, Sommerville identifies five steps [20]: (a) requirements definition, (b) systems and software design, (c) implementation and unit testing, (d) integration and system testing, and (e) operation and maintenance. Concerning individual steps, the cost of defect repair increases rapidly, the later a defect is detected, and can be avoided [1] with additional activities in previous phases. In this paper we investigate software inspection and introduce a new testing variant, usage-based test with inspection (UBT-i) for defect detection in early stages of software development.

Software inspection is a structured approach for defect detection in written text documents, e.g., design specifications, applying reading techniques for inspection assistance. Therefore, software inspection does not require executable code. Reading techniques provide a structured process of individual reading approaches, using a checklist (CBR), use cases (UBR), etc. Several studies exist on the investigation of CBR and UBR reading techniques [1][10][21][22][23], based on the original version of Fagan's inspection [7].

Software testing, i.e., black-box and white-box testing scenarios, are common methods for quality improvement in implementation and testing phases. Traditional testing approaches require executable code to compare outcomes of the software product to the specification document based on predefined sets of test cases. The application of tests assumes the correctness and completeness of requirements and design documents. Critical defects in code documents may require rework of the specification document in case of an error in the specification phase. Therefore, the application of software testing depends on the existence of given requirements, test cases according to these requirements, and executable code. Once, detecting a defect in a testing scenario, testers have to locate the origin of the defect, because no detailed information exists on the defect location applying black-box testing. Therefore, additional effort is necessary for location and correction purposes.

Our testing approach (UBT-i) supports written text documents (design specifications) and a hardcopy of source code as well applying a desk test. Concerning

Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05)
0-7695-2431-1/05 $20.00 © 2005 IEEE

source code documents, the control flow of the code fragments are stimulated mentally in order to find and locate defects.

Test case generation is part of UBT-i. Therefore, UBT-i provides defect detection and location in early stages of software development and test case generation regarding expert-prioritized use cases. In contrast to usage-based testing [1], we merge test case generation and desk test execution according to an individual order of expert-ranked use cases, following four steps: (a) select use case with top priority, (b) generate test cases, (c) apply test cases and report defects, and (d) select the next use case until time is up or total coverage of all use cases.

This paper presents the results of an initial empirical study in an academic environment [6] to investigate the impact of UBR and UBT-i with respect to defect detection according to time measures, i.e. efficiency and effort, and inspection performance, i.e. the number of defects found during method application.

The experiment environment includes a taxi management system, consisting of a requirements document with use case notation and a design specification regarding a central and driver part, associated by a communication link. We apply existing basic material, already used in several experiments (see [26][23][22]) to provide comparability to previous experiments, and to keep the preliminary effort for experiment preparation and setup reasonable.

The reminder of this paper is structured as follows. Section 2 describes related work on usage-based testing and software inspection. Section 3 points out research hypotheses, section 4 outlines the experiment settings, Section 5 presents the results of the study, and section 6 discusses the results. Finally, section 7 concludes and describes directions for further research.

2. Usage-Based Reading and Testing

Software inspection is a team-oriented, static verification and validation (V&V) approach for software product improvement regarding software design specifications. Typical software inspections in industrial practice consist of four major steps: (a) inspection planning, (b) defect detection, (c) defect collection, and (d) defect repair [3]. In this paper we focus on individual defect detection and defect collection using nominal teams.

We separate *experiment preparation* effort, including document and checklist preparation as well as generation and prioritization of use cases (performed by experts) and *individual software inspection effort* (performed by study participants).

Inspectors perform an individual preparation phase, according to the reading technique applied and an individual inspection phase. Reading techniques aim to support inspectors during inspection process. In our paper we focus on Usage Based Reading (UBR). Inspectors use expert prioritized use cases, apply them sequentially to the document under inspection and record candidate defects. Use case prioritization is part of the overall preparation phase performed by experts, who are familiar with the application domain. This prioritization task is not considered in our evaluation of inspection effort. The application of prioritized use cases provides (a) a set of sorted use cases, according to their importance and (b) an active guidance through the inspection process [26]. Primary studies showed a better performance of UBR compared to checklist-based approaches [3][22][23].

Inspectors, applying UBR, perform the following sequence of steps:

1. Choose a use case with highest priority according to use case rating.
2. Apply the use case to the design specification and record candidate defects. Candidate defects are subjectively classified defects, rated by individual inspectors. The design specification was seeded with defects by experts. Candidate defects which agree with seeded defects are considered as matched defects.
3. Continue with the next use case until time is up or total use case coverage.

Because inspectors traverse the document under inspection several times according to a set of use cases, they find defects and they know the location of the defect as well as a common characteristic for all inspection approaches.

Another useful approach for software quality improvement is testing [11]. The traditional software testing process uses a set of test cases based on a design specification and executable code. Testing approaches, like black-box testing, enable defect detection by comparing test results against the expected behavior, i.e. the specification. Test cases are clustered according to their expected behavior using equivalence classes, i.e. selecting one member of a set of test cases promising to achieve similar test results (e.g. test case success, border cases or exceptional cases) to limit the number of test cases. Nevertheless, there are very few possibilities to locate defects during the test process without deeper knowledge of the code documents. After performing tests, software engineers have to locate defects regarding the test results. Several studies have been performed to investigate testing approaches [10] and have compared testing and inspection [1][19]. Andersson et al. introduced Usage-Based Testing

(UBT) [1] concerning expert prioritized use cases and test cases, which were applied to code documents. These artifacts are the basic material for UBT. Additional effort is necessary to set up and prioritize use cases and test cases.

Concerning the software life-cycle, UBR is originally located in early software development phases and UBT is typically located in the implementation phase or later. We assume an improvement of testing, based on a modification of UBT including inspection methods. Our approach includes a twofold benefit: (a) UBT-i may be applied to design specifications and code documents as well performing a desk test, and (b) the generation of test cases is an integral part of the testing process. Thus, additionally to defect detection, test case generation is an additional outcome of UBT-i. Executing our UBT-i approach, inspectors perform four major steps:

1. Choose the first prioritized use case.
2. Find equivalence classes and test cases according to the selected use case, applying guidelines for equivalence class derivation.
3. Apply test cases regarding use cases and record candidate defects.
4. Continue at step 1 until overall use case and document coverage or reached time limit.

One advantage of UBT-i is the knowledge of the defect location in the design specification as well as in the source code document.

3. Research Questions

The main focus of this paper is the investigation of performance as well as time variables for UBR and UBT-i. UBR focuses on design specification using guidelines and use cases for defect detection in early software development phases [23][26]. Usage-based testing is an approach for defect detection for code documents and design specification [1]. UBT fits to implementation phases or later in the software development process. We introduce a new testing variant, UBT-i, merging the benefits of UBT and UBR. UBT-i is a desk test, without the requirement for executable code, and therefore, enables application to text documents as well. During desk testing, inspectors derive equivalence classes and test cases from use cases and design specification and aim to find defects in early phases of software development. In this paper we focus on inspection effort, effectiveness, efficiency, and some basic approaches on team composition.

3.1. Variables

We define *dependent* and *independent* variables. The *independent variable* is the technique applied, i.e. reading technique UBR and usage-based testing with inspection (UBT-i). UBR applies an expert ranked order of predefined use cases to traverse the document under inspection several times. UBT-i uses expert prioritized use cases as well (same basic material), finds equivalence classes and test cases, and applies them on the software artifacts. We control the influence of inspector capability by randomly assigning inspectors to reading techniques and testing approaches.

Dependent variables capture the performance of the individual technique applied during the experiment proceeding. Following a standard practice of empirical software engineering, we focus on performance measures and time variables. Additionally, we introduce a team comparison to investigate the influence of randomly combined individuals to teams of up to 6 members on inspection performance.

Performance measures are *effectiveness*, i.e. the number of found matched defects regarding the overall number of seeded defects and *efficiency*, i.e. the number of defects found per hour, according to three defect severity classes: (a) critical defects (class A), major defects (class B), and minor defects (class C). Critical defects mean a heavy adverse effect on functionality which will appear very often (highest risk). Major defects contain important rarely used defects or less important often used defects (medium risk) and minor defects are neither crucial nor very important (low risk).

For evaluation purposes we focus on critical defects, important defects (class A + class B) and all matched defects. The main time variable is the overall *effort* used for a technique (in minutes).

3.2. Hypotheses

In the experiment we observe the performance of inspectors applying UBR and UBT-i. As the main goal of this paper we investigate research hypotheses regarding effort, effectiveness, efficiency, and team composition and their influence on three different defect severity classes. In more detail, we evaluate the following hypotheses.

Inspection effort includes preparation and execution time of UBR and UBT-i. We do not include the overall preparation phase, i.e. generation of use cases and prioritization, because these preliminary tasks are similar for both approaches.

H1: Effort (UBR) < Effort (UBT-i): We expect an overall higher effort for UBT-i inspectors than for

UBR inspectors. In more detail, UBR inspectors take longer to traverse the documents following their technique than UBT-i users. But we expect a higher effort for the testing approach, because of two additional tasks applying this method. Based on prioritized use cases inspectors have to find equivalence classes and generate test cases according to every use case sequentially. Applying test cases to the document under inspection, inspectors find defects and report them. Concerning UBR, inspectors apply use cases to the design specification without performing these individual tasks. Beside from defect detection, UBT-i achieves a set of test cases for reuse purposes in the implementation and testing phase.

Effectiveness is the number of defects found concerning different severity classes, in relation to all seeded defects in these classes and document locations (i.e. design specification and source code documents).

H21: Effectiveness (UBR) > Effectiveness (UBT-i): Regarding the comparison of UBR and UBT, Andersson et al. [1] found significantly higher effectiveness of inspectors applying UBR (replicated hypothesis). Note that the UBT group does not have to perform additional tasks but applies given test cases to the design document. In our testing approach (UBT-i), inspectors have to create test cases as an additional task. Because of an overall upper time limit, the inspectors might spend less time for their defect detection task. Therefore, we expect stronger differences and UBR to be more effective than UBT-i.

H22: Effectiveness_location (UBT-i) > Effectiveness_location (UBR). Because UBR focuses on design specifications (written text documents) and UBT-i focuses on source code, we expect a higher effectiveness at UBT-i, concerning defect detection rates in code documents. We apply a similar argument to UBR, that UBR users find more defects in design specifications.

Efficiency combines inspection performance (effectiveness) and inspector time variables (effort). We define efficiency as the number of defects found in a time interval, i.e. defects per hour. Additionally, we regard different defect severity classes.

H31: Efficiency (UBR) > Efficiency (UBT-i): Due to additional tasks of UBT-i, we expect a lower efficiency of inspectors applying UBT-i, because UBT-i inspectors have to spend additional time on finding equivalence classes and test case generation.

Nominal teams: We assume a higher effectiveness concerning nominal teams involving both inspection approaches, in relation to a team applying a uniform technique, because of individual focus of every technique.

H41: Effectiveness (Mix) > Effectiveness (uniform): In our evaluation, we use a uniform distribution of UBR and UBT-i for even team size and one additional UBR inspectors for odd team size to emphasize software inspection (see table 1 for details).

Because of individual defect detection approaches, we expect some kind of synergy effect applying teams. UBR focus on design specifications and UBT-i seems to pay more attention on source documents. Following this approach, inspection teams must involve UBR and UBT-i as well. Therefore, we use an almost uniform team sampling approach. Table 1 displays team composition in more detail (we use the short-cut R for UBR and T for UBT-i).

Table 1. Team Composition

Team Size	UBR	UBT-i	UBR/UBT-i
1	1R	1T	1R
2	2R	2T	1R1T
3	3R	3T	2R1T
4	4R	4T	2R2T
5	5R	5T	3R2T
6	6R	6T	3R3T

In this paper we set up an upper team size of 6 team members [3], summarizing all matched defects found by the individuals. Note, that we count seeded defects only once, even if the defect was found several times by different members of the nominal team. To achieve comparable results we randomly build 10 teams and calculate mean value and standard deviation.

4. Experiment description

The initial study was conducted at Vienna University of Technology in academic environment in December 2004. This study is an extension of previous studies [26][6], concerning usage based reading techniques and a modified version of replicated experiment conducted at Lund University, Sweden [1][22][23]. In this section we will briefly describe key aspects of the experiment proceeding, software artifacts and experiment participants.

4.1. Experiment Proceeding

The experiment consists of three major steps: (a) experiment preparation, (b) experiment execution, and (c) data evaluation.

During the *experiment preparation* phase, experts had to prepare the software artifacts, i.e. requirements document, design specification including interspersion of defects, construction and prioritization of use cases,

and generation of code documents, reflecting program functionality. Furthermore, several guidelines and supporting documents, e.g. questionnaires had to be provided.

The *experiment execution* phase was performed in three major steps: a training and preparation phase, individual application of the method, and data submission. Inspectors got an overview of the application area (45 min) and the inspection process (45 min) and got familiar with the software artifacts. Afterwards the inspectors applied the randomly assigned inspection approaches to the documents under inspection (300 min) and passed their results to a database.

After data submission experts mapped all candidate defects, i.e. defects noted by the individual inspectors, to real defects, i.e. defects seeded by experts. The data were checked for correctness and consistency. We excluded data from inspectors who did not follow the experiment process properly. Note, that multiple candidate defects that refer to the same seeded defect was counted only once at the first clock time of notation. For statistical evaluation purposes, we use the Mann-Whitney test at a significance level of 95%.

4.2. Software Artifacts

The artifacts in this initial study describe a taxi management system including a central and driver part of the system. We do not separate both parts in our evaluation.

The document framework consists of (a) a textual description of requirements and use case definitions, (b) a design specification, containing seeded defects, (c) source code documents, (d) guidelines for experiment proceeding, and (e) questionnaires for inspector feedback on inspector capability and reading technique approach used.

- The *textual requirements document* spans 8 pages including 2 component diagrams and is assumed to be accurate.
- The *design document* describes an overview of the software modules and their context including an internal (relationship between two or more modules) and an external representation (relationship between the user and the system). The design documents consist of 8 pages (including approximately 2400 words, 2 component diagrams and 2 UML diagrams). Furthermore, we provide *prioritized use case descriptions* containing 24 use cases from user viewpoint and an overall number of 23 sequence diagrams.
- We provide *source code* (some 1400 lines of code) and a 9 page method description using JavaDoc.
- *Guidelines* support the inspectors in performing the individual tasks. We introduce *questionnaires* to achieve knowledge of inspector capability (*experience questionnaire*) and feedback of the individual inspection approaches.
- The document package (design specification and source code documents) contains 60 seeded defects (27 defects (45%) in the design document and 33 defects (55%) in the source code documents) according to three different defect severity classes. Concerning different severity levels, the design specification and the source code, contains 29 (49%) critical (class A) defects, 24 (24%) major (class B), and 7 (12%) minor (less important, class C) defects. Table 2 presents the nominal numbers of seeded defects according to defect severity classes and document types.

Table 2. Seeded Defects according to defect severity classes

	Number of defects		
	Design	Source	Sum
Critical (class A)	10	19	29
Major (class B)	12	12	24
Minor (class C)	5	2	7
Summary	27	33	60

4.3. Subjects

The subjects in this initial study were 29 software engineering students. UBR and UBT-i were assigned randomly to the experiment participants to control the influence of inspector capability. During the experiment, a supervisor supported the participants to ensure the compliance to the experiment proceeding and the given instructions. The experiment was fully integrated in the practical part of course for software quality assurance to learn key aspects of software product improvement in early stages of software development.

We assigned 15 inspectors (52%) to Usage Based Reading (UBR) and 14 inspectors (48%) to Usage Based Testing with Inspection (UBT-i).

5. Experiment Results

In this section we present the results of the empirical study concerning effectiveness, efficiency, and some preliminary results of nominal team aspects.

5.1. Effort

Effort is the overall session duration, involving individual preparation and execution time. In this

evaluation, we summarize both time intervals for effort, because there is no additional effort within the inspection/testing execution. Table 3 displays mean values and standard deviation of inspection effort.

Table 3. Inspection Effort in Minutes

	UBR	UBT-i
Mean	272.5	268.8
Std.Dev.	38.0	29.1

Both approaches have on average similar effort. We do not recognize a significant difference concerning inspection effort, but there is somewhat higher effort for UBR but also a higher standard deviation.

5.2. Effectiveness

Effectiveness is the number of defects found in relation to the overall number of seeded defects per defect class. The experiment setup consists of 60 defects (27 defects in the design document and 33 defects in the source code). Concerning defect severity classes, we pay attention to 29 critical defects (class A) and 53 important defects (class A and class B defects).

We do not observe any significant difference concerning all matched defects regarding document location (design specification and source code) but we observe a somewhat higher effectiveness for UBR inspectors (mean values: 38.7 (UBR) and 34.3 (UBT-i)). Further investigations show significant differences concerning critical (class A) defects and source code documents.

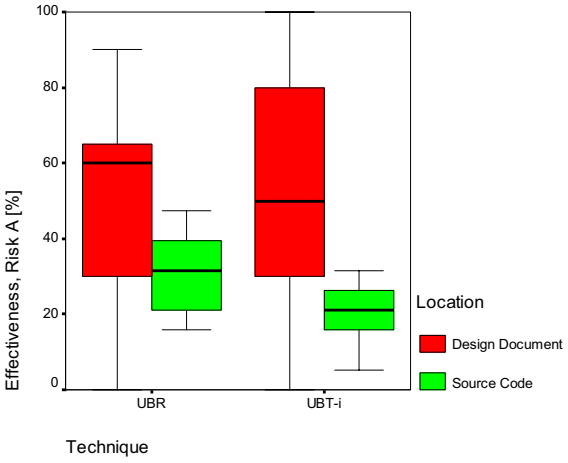

Figure 1. Effectiveness according to critical defects and document location

Figure 1 depicts the overall effectiveness for critical defects according to document location. Surprisingly, UBR approaches achieve significantly higher effectiveness with respect to critical defects in source code documents.

Table 4. Effectiveness, Source Code

	Defect class	UBR	UBT-i
Mean	Class A	30.2	21.3
	Class A+B	35.3	29.7
	All defects	33.7	28.4
Std.Dev	Class A	10.8	10.0
	Class A+B	11.4	12.2
	All defects	11.0	11.4

Table 4 presents the summarized results for source code documents. Nevertheless, effectiveness concerning UBR is somewhat higher at every defect severity class and for both document locations.

5.3. Efficiency

Similar to effectiveness, we recognize a somewhat higher efficiency concerning UBR approaches, but we do not notice any significant difference using the Mann-Whitney Test at a significance level of 95%.

Table 5 displays a summary concerning mean value and standard deviation of efficiency regarding matched defects.

Table 5. Efficiency, Matched Defects

	Defect Class	UBR	UBT-i
Mean	Class A	2.4	2.2
	Class A+B	4.7	4.4
	All Defects	5.2	4.8
Std.Dev	Class A	0.8	1.2
	Class A+B	1.6	2.3
	All defects	1.8	2.7

Again, we find a significant difference concerning critical defects and source code documents. Table 6 gives a summary of our findings with respect to source code documents.

Table 6. Efficiency, Source Code

	Defect	UBR	UBT-i
Mean	Class A	1.3	1.0
	Class A+B	2.4	2.1
	All defects	2.5	2.2
Std.Dev	Class A	0.4	0.6
	Class A+B	0.8	1.0
	All defects	0.8	1.0

Using the Mann-Whitney test to observe statistical significance levels, we notice a significant difference concerning critical (class A) defects.

Table 7. P-Values Efficiency, Source Code

	Class A	Class A+B	All defects
UBR / UBT-i	0.040 (S)	0.230 (-)	0.222 (-)

Table 7 presents the p-values of efficiency for source code documents according to defect severity classes. We observe a significant difference for class A (critical) defects.

5.4. Nominal Teams

To investigate the team effect of nominal teams, we merged individuals applying the same and different inspection approaches randomly. Table 1 displays the setup for team composition. We consider a nominal team as a collaboration of two or more members without interaction [4][5].

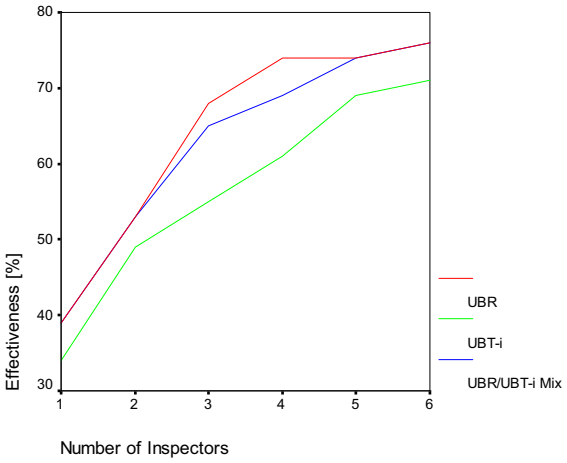

Figure 2. Effectiveness for Nominal Teams.

Figure 2 displays effectiveness of nominal teams concerning a twofold evaluation task: (a) presents teams consisting of uniform inspection approaches, i.e. all team member applied UBR or UBT-i, and (b) describes the mix of both approaches, as described in section 3.2. We observe a continuous increasing number of effectiveness, because inspectors detect different defects (Note, that we count defects only once per team). UBR teams achieve the best performance, UBT-i teams perform worst; our randomly selected sample of both approaches draws near to UBR. Concerning efficiency, we observe a similar effect.

Table 8. Mean Values acc. to Effectiveness and Efficiency, Nominal Teams.

No	Effectiveness			Efficiency		
Insp.	UBR	UBT-i	Mix	UBR	UBT-i	Mix
1	39.8	34.4	39.5	5.2	4.4	5.2
2	53.4	49.6	53.5	3.5	3.2	3.4
3	68.0	55.9	65.8	2.8	2.4	3.0
4	74,3	61.4	69.9	2.3	2.1	2.3
5	74.1	69.1	74.2	2.0	1.8	1.9
6	76.7	71.0	76.7	1.7	1.6	1.7

6. Discussion

In this section we summarize the empirical results from our experiment concerning effort, effectiveness, efficiency, and team composition. Analyzing the results we derive the following implications for the comparison of UBR and UBT-i.

Effort: The results showed no significant overall difference concerning UBR and UBT-i. One reason might be some kind of a group effect for overall inspection duration, because we set up an upper limit of 300 minutes. Further investigation on the document and use case coverage is necessary. Because UBT-i performed an additional task, finding of equivalence classes according to given guidelines for equivalence class derivation and generation of test cases, only a subset of the prioritized use cases might be considered.

In summary the hypothesis H1, that UBR need less effort than UBT-i must be rejected.

Effectiveness: We define effectiveness as the ratio of matched defects with respect to all seeded defects at the observed defect class (different severity levels) and document location (source code and design documents). We recognize significant differences in observing source code documents and critical defects. Additionally, effectiveness concerning UBR is somewhat higher than UBT-i for all evaluation tasks, but we do not recognize any significant difference.

Therefore, we have to reject our replicated hypotheses H21, that Effectiveness (UBR) > Effectiveness (UBT-i). We imply an equivalent application of UBT-i with respect to UBR including the additional benefit of generated test cases at UBT-i. One possible problem might be the document coverage, because we didn't observe document coverage, i.e. did the inspectors apply all use-cases or did they run out of time.

We also have to reject hypothesis H22, that UBT-i performs better regarding UBR according to document location for critical defects. Because of prioritized use cases inspectors focus on critical and important defects first. Another reason for this effect might be the desk test approach of UBT-i because inspectors do use a compiler (i.e. execute program code) to find defects but have to "compile" mentally, i.e. stimulating program control flow.

Efficiency: Efficiency is the number of defects found regarding predefined time intervals. In our initial study we investigated defects found per hour. Similar to effectiveness we observe a significant difference concerning critical (class A) defects and source code documents. Furthermore, our results don't show any significant differences regarding the hypothesis. Note again, that UBR inspectors achieve a somewhat higher efficiency than UBT-i. Our hypothesis, UBR performs significantly better than UBT-i, according to efficiency, must be rejected.

We suppose similar reasons for these results according to effectiveness. Because efficiency includes time values (i.e. defects found per time interval), the measured value might be supported as well.

Team composition: Because different inspection approaches focus on different document locations, i.e. design specification and source code, we expect a better overall performance concerning effectiveness and efficiency. Therefore, we mixed teams up to 6 team members, using a uniform inspection approach distribution and control groups applying the same inspection approach. We observe the best performance for UBR and the worst overall performance for UBT-i. Our team mix is near to UBR but somewhat lower. Following these results we have to reject our hypothesis H41. Investigating the results of UBR and UBT-i we assume a similar effect on team composition. Another possible reason might be our number of samples (i.e. 10 nominal teams concerning each team composition strategy), which influences team composition and results of effectiveness and efficiency.

7. Conclusion and Further Work

Inspection and testing are important approaches in software engineering practice addressing the reduction of defects in software products. Software inspection focuses on design specifications in early phases of software development and traditional testing approaches focus on implementation phases or later. Therefore, we introduced a new testing variant, UBT-i, integrating benefits of software inspection and software testing. UBT-i is a desk test and – in contrast to testing approaches – doesn't require executable code. Additionally, UBT-i inspectors generate equivalence classes and test cases during inspection proceeding.

This paper presents the basic description of this new testing approach and the results of an initial study, performed at Vienna University of Technology, according to effectiveness, efficiency and some basic ideas on team composition.

The main findings of this study were: (a) no significant differences concerning all matched defects according to effectiveness and efficiency, (b) significant differences for critical defects (class A) in source code documents, and (c) all performance measures are somewhat higher for UBR than for UBT-i, concerning defect severity classes and defect location.

Further work is necessary to investigate inspector capability on UBT-i in the experiment environment and document coverage of individual inspection approaches to achieve deeper knowledge of the impact of additional tasks, i.e. finding of equivalence classes and test cases, applying UBT-i. Furthermore, we have to replicate our initial study to verify the results involving a higher number of participants to improve external validity.

We invite researchers with interest in empirical studies in V&V to share their interests and insights to replicate our initial study.

8. References

[1] Andersson C., Thelin T., Runeson P., Dzamashvili N.: "*An experimental evaluation of inspection and testing for detecting of design faults*", ISESE'03 – International Symposium of Empirical Software Engineering, pp. 174-184, 2003.

[2] Basili V.R., Shull F., Lanubile F.: "*Building Knowledge through Families of Experiments*", IEEE Trans. Software Eng., vol. 25, no. 4, pp. 456-473, July/Aug. 1999.

[3] Biffl St.: "*Software Inspection Techniques to support Project and Quality Management*", Shaker Verlag, 2001, ISBN: 3-8265-8512-7.

[4] Biffl St., Gutjahr W.: "*Influence of Team Size and Defect Detection Technique on Inspection Effectiveness*", Seventh International Software Metrics Symposium, IEEE, 2001.

[5] Biffl St., Halling M.: "*Investigating the defect detection effectiveness and cost benefit of nominal inspection teams*", IEEE Transactions on Software Engineering 29(5) (2003), pp. 385–397.

[6] Biffl S., Winkler D., Thelin T., Höst M., Russo B., Succi G.: "*Investigating the Effect of V&V and Modern Construction Techniques on Improving Software Quality*", Poster presented at ISERN 2004.

[7] Fagan M.: "*Design and Code Inspections To Reduce Errors In Program Development*", *IBM Systems J.*, vol. 15, no. 3, 1976, pp. 182-211.

[8] Fusaro P., Lanubile F., Visaggio G.: "*A Replicated Experiment to Assess Requirements Inspection Techniques*", Empirical Software Eng.: An Int'l J., vol. 2, no. 1, pp. 39-57, 1997.

[9] Gilb T., Graham D.: "*Software Inspection*", Addison-Wesley, 1993.

[10] Juristo N., Moreno A. M., Vegas S.: "*A Survey on Testing Technique Empirical Studies: How Limited is Our Knowledge*", Proceedings of the 1st International Symposium on Empirical Software Engineering, pp. 161-172, 2002.

[11] Karner C., Falk J., Nguyen H.Q.: „*Testing Computer Software*", Wiley, 1999, ISBN 0-471-35846-0.

[12] Laitenberger O., Atkinson C.: "*Generalizing Perspective-based Inspection to handle Object-Oriented Development Artifacts*", Proc. of the Int. Conf. on Software Engineering, 1999.

[13] Laitenberger O., DeBaud J.-M.: "*An encompassing life cycle centric survey of software inspection*", Journal of Systems and Software, vol. 50, no. 1, 2000, pp. 5-31.

[14] Parnas D., Lawford M.: "*The role of inspection in soft-ware quality assurance*", IEEE Trans. on SE, vol. 29(8), August 2003, pp. 674-676.

[15] Porter A., Votta L.: "*Comparing Detection Methods for Software Requirements Inspections: a Replicated Experiment using professional subjects*", Empirical Software Engineering Journal, vol. 3, no. 4, 1998, pp. 355-379.

[16] Porter A., Votta L., Basili V.: "*Comparing Detection Methods for Software Requirements Inspections: a Replicated Experiment*", IEEE Transactions on Software Engineering vol. 21, no. 6, pp. 563-575, June 1995.

[17] Sandahl K., Blomkvist O., Karlsson J., Krysander C., Lindvall M., Ohlsson N.: "*An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections*", Kluwer Academic Publishers, 1998.

[18] Shull F., Basili V., Boehm B., Brown W., Costa P., Lindvall M., Port D., Rus I., Tesoriero R., Zelkowitz M.: "*What We Have Learned About Fighting Defects*", IEEE Metrics 2002.

[19] So S.S., Cha S.D., Shimeall T.J., Kwon Y.R.: "*An Empirical Evaluation of Six Methods to Detect Faults in Software*", Software Testing, Verification and Reliability, 12(3):155-172, 2002.

[20] Sommerville I.: "*Software Engineering*", 6th Edition, Addison-Wesley, 2001.

[21] Thelin T, Andersson C., Runeson P., Dzamashvili-Fogelström N.: "*A Replicated Experiment of Usage-Based and Checklist-Based Reading*", Proceeding of 10th Int. Symp. on Software Metrics, 2004.

[22] Thelin T., Runeson, P., Regnell B.: "*Usage-Based Reading – An Experiment to Guide Reviewers with Use Cases*" Information and Software Technology, vol. 43, no. 15, pp. 925-938, 2001.

[23] Thelin T., Runeson, P., Wohlin, C.: "*An experimental comparison of usage-based and checklist-based reading*", IEEE Transaction on Software Engineering, 29(8), pp. 687-704, 2003.

[24] Thelin T., Runeson, P., Wohlin C., Olsson, T., Andersson, C.: "*How much information is needed for usage-based reading? – A series of experiments*", International Symposium on Empirical Software Engineering (ISESE'02), 2002.

[25] Westland J. C.: "*The cost of errors in software development: evidence from industry*", The Journal of Systems and Software 62, 1-9. (2002).

[26] Winkler D., Halling M., Biffl St.: "*Investigating the effect of expert ranking of use cases for design inspection*", Euromicro Conference, Rennes, France, IEEE Comp. Soc, 2004.