Prioritized Use Cases as a Vehicle for Software Inspections

Thomas Thelin, Per Runeson
Dept. of Communication Systems,
Lund University
Box 118, SE-221 00 LUND, Sweden
{thomas.thelin, per.runeson}@telecom.lth.se

Claes Wohlin
Dept. of Software Eng. and Computer Science
Blekinge Institute of Technology
Box 520, SE-372 25 Ronneby, Sweden
claes.wohlin@bth.se

Principles from software inspections, use cases and operational profile testing are combined into the usage-based reading *technique (UBR). The goal is to provide an efficient reading technique for software inspections, which takes the user viewpoint on the software and the faults it may contain. The user reads, for example, a design document guided by prioritized use cases. An experimental evaluation shows that the UBR method is more effective and efficient in finding faults, critical to the user, compared to checklist-based methods.*

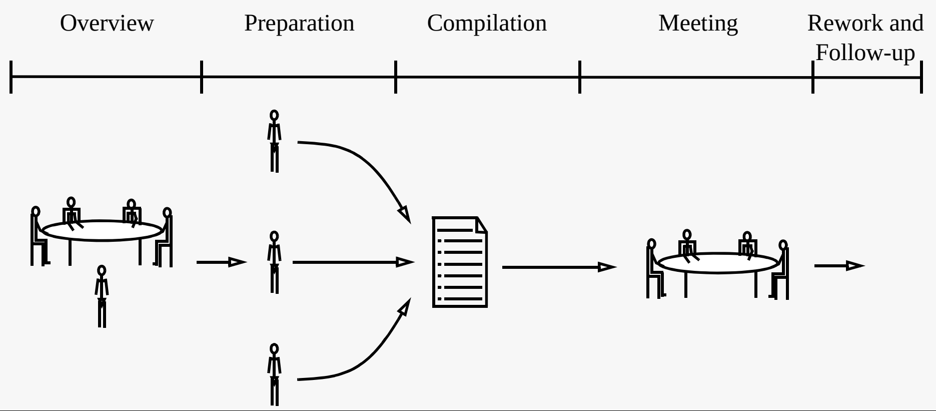
INSPECTIONS AND USE CASES

Introduction

Software inspections have achieved quite some interest, since its first presentation by Michael Fagan in 1976 [1]. Inspection is a static verification and validation technique, which can be applied to any kind of artefact prepared in software development. There is no need for dynamic execution of the artefact. The steps of an inspection process are typically 1) *overview* of the artefact to be inspected, 2) individual *preparation*, where each reviewers reads the artefact, 3) *compilation* of the findings into one list of issues, 4) inspection *meeting*, and finally 5) *rework and follow-up* (See *Figure 1* and [2]). Inspections are considered industry practice in domains where software quality is of importance for the delivered products. Despite the fact that much empirical research is conducted on inspections [3], the traditional checklist approach to inspections is still dominating in industry [4].

Another trend in software engineering seen during the last two decades is the turn towards object-oriented design methods, and the application of use cases in this context [5]. Use cases and scenarios (see box *Use Cases and Scenarios*) help the development turn user-focused instead of being solely technology-focused. Use cases and scenarios can, for example, be documented in terms of UML use case diagrams and sequence diagrams respectively, or they can be documented using structured text. Use cases are generally developed as a means for the specification of a system,

Figure 1. Inspection process



Use Cases and Scenarios

Scenario: A sequence of steps describing an interaction between a user and a program

Use case: A set of scenarios tied together by a common user goal

Source: M. Fowler and K. Scott, *UML Distilled*, Addison Wesley, 2000.

but can also be used for inspection purposes later in the development.

Furthermore, usage-based testing, or operational profile testing has contributed to setting the user and usage of a system in focus of the test activities [6]. The basic principle is that test cases are selected according to the frequency or probability of operational use. Hence, the most important functions, from the user viewpoint, are tested first and most.

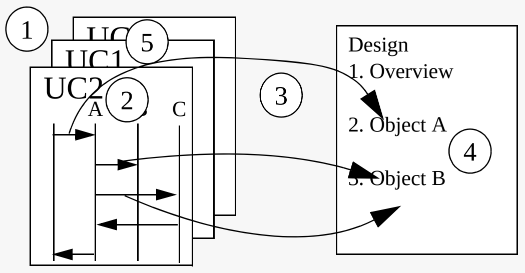
Usage-Based Reading

We combine inspections, use cases/scenarios and principles of operational profile testing into *usage-based reading* (UBR) to provide an efficient reading technique for application in software design inspections [7][8]. The technique takes into account the fact that different faults may have different impact on the user perception of the product. The focus here is primarily on design inspections, while code inspections guided by use cases have been discussed elsewhere [9].

Independently of the inspection method used, there is always a component of reading the artefact under inspection, i.e. individual preparation (see *Figure 1*). The predominant technique in industry is checklist-based reading [4], where a list of check items guides the reviewers on how they should focus their attention when reading.

UBR can be used for inspection of documents developed after the derivation of use cases, e.g. requirements, design, code and test documents. It is assumed that the use cases and/or scenarios are defined earlier in the development process. UBR utilizes the set of use cases as a vehicle for focusing the inspection effort, much the same way as a set of test cases focus the test effort.

Figure 2. Usage-Based Reading



Method Description

The basic steps of the method are as follows and in Figure 2:

Before inspection

1. Prioritize the use cases in order of importance from a user perspective.

In preparation

2. Select the use case with the highest priority.
3. Track the scenarios of the use case through the document under inspection.
4. During tracking, ensure that the document under inspection fulfils the goal of the use case, that the needed functionality is provided, that the interfaces are correct etc. Identify and report the issues found.
5. Select the next use case and repeat from 3. until the time is up, or all use cases are covered.

Method Adaptation

As the UBR method steps only cover the preparation part of an inspection process, it is to be used as an add-on to existing inspection processes. Hence, existing procedures for resource scheduling, meetings and follow-up can be used in conjunction with UBR. The method is easily introduced in a development project where use cases and scenarios are defined at a sufficient level of detail and completeness. If an operational profile is defined for the system, it contains important information for the prioritization activity.

The prioritization can, for example, be performed by pair-wise comparisons, guided by the *analytic hierarchy process* (AHP) [10]. AHP enables determining the consistency of the comparisons if all pairs are compared or can be used to finding priorities without having to explicitly to define the relations between all pairs of use cases.

If the detailed scenarios are available, it is beneficial over having only the use case description, but it is not shown to be worth the effort of developing the scenarios solely for the UBR inspection [11].

EXPERIMENTAL EVALUATION

The Experimental Setting

In order to evaluate the method, we launched an experiment [12]. For a more elaborate description of the experiment, refer to [8]. We wanted to investigate whether UBR is a better approach applied to individual preparation than checklist-based reading (CBR). With *better*, we refer to *more effective,* i.e. finding larger share of the faults, and *more efficient,* i.e. finding more faults per time unit. The new method is compared to the traditional checklist-based reading (CBR) in a student context. 23 students in the last year of a Software Engineering master's program applied the reading techniques when inspecting a design document for a taxi management system. 11 students applied UBR and 12 students applied CBR.

The system is designed to manage a fleet of taxis, dispatching customer orders and follow up the transports made by each taxi driver. The system is an academic product, scaled down from the real application domain, but realistic by being developed under interaction with true stakeholders. The design document is 9 pages long, containing 2300 words and 38 faults. The faults are classified as A, B or C for *crucial, important* and *not important* from a user's point of view. There are 13, 14 and 11 faults of classes A, B and C respectively. The majority of the faults (28) were found during the initial development of the design document and re-inserted before the experiment. The person who developed the system seeded eight new faults and two additional faults from the original development were found during the experiment, summing up to 38 faults. The use case document comprised 24 use cases with defined scenario sequences and alternatives. The use cases and the faults were prioritized by different experts to ensure independent priorities.

The students are considered sufficiently knowledgeable and experienced to be representative for programmers in an industrial context. Many of the students have industrial experience in software engineering and all of them had run a full semester 15-member project with real customers. They were taught both reading techniques before the experiment was launched. A pre-test survey was used to ensure that the two groups were balanced regarding experiences and skills.

Experimental Results

The experimental evaluation shows that UBR is significantly more *efficient* than CBR, i.e. finds more faults per time unit for crucial and important faults (classes A and B). Usage-based reading is also significantly more *effective* than CBR, i.e. finds a larger share of the faults. The data are presented in Table 1 and Table 2, and the time spent on preparation and inspection is presented in Table 3. The differences denoted with an asterisk (*) are statistically significant on 95% level. More details on the data and the analysis can be found in [8].

Table 1: Efficiency data (faults per hour)

	Mean		**Std Dev.**	
	UBR	**CBR**	**UBR**	**CBR**
All Faults*	5.6	4.1	2.0	2.0
Class A Faults*	2.6	1.3	1.0	1.1
Class B Faults	2.1	1.4	1.2	0.7
Class C Faults	0.9	1.4	0.4	0.8
Class A+B Faults*	4.7	2.8	1.8	1.7

Both groups spent about the same time on their tasks. The UBR reviewers spent on average 6.5 minutes less in preparation and 4 minutes less in inspection. Nevertheless, they found more faults.

Regarding *efficiency,* UBR reviewers found twice as many crucial faults per hour (2.6 vs. 1.3 class A faults) compared to CBR reviewers. For the important faults, reviewers

Table 2: Effectiveness data (share of faults)

	Mean		Std Dev.	
	UBR	**CBR**	**UBR**	**CBR**
All Faults (38)	0.31	0.25	0.09	0.14
Class A Faults (13)*	0.43	0.24	0.17	0.21
Class B Faults (14)	0.31	0.24	0.15	0.13
Class C Faults (11)	0.18	0.30	0.08	0.21
Class A+B Faults (27)*	0.37	0.24	0.12	0.16

Table 3: Preparation and inspection time (minutes)

	Mean		Std Dev.	
	UBR	**CBR**	**UBR**	**CBR**
Preparation	53	59	20	15
Inspection	77	81	18	19
Total	130	140	15	12

using UBR found about 50% more faults per hour (2.1 vs. 1.4 class B faults) than those using CBR.

Regarding *effectiveness*, reviewers using UBR identified on average 21% more faults than reviewers using CBR (0.31 vs. 0.25 for all faults). For the crucial faults, UBR identified 75% more faults than CBR did (0.43 vs. 0.24 class A faults). CBR on the other hand identified 63% more of the non-important faults (0.30 vs. 0.18 class C faults). Note that the percentages of the efficiency and effectiveness may not sum to 100 due to rounding.

Based on the data, we can conclude that the UBR method succeeds in directing the inspection effort to find the problems that are considered most important from a user's point of view, instead of wasting the effort on searching for less important issues.

SUMMARY AND CONCLUSIONS

Usage-based reading (UBR) provides support to the individual preparation in the inspection process. It is based on the use cases and scenarios often prepared as a part of specification and analysis efforts. Prioritized use cases guide the reviewers in their search for issues that may cause problems later in the development.

To evaluate the effectiveness and efficiency of UBR, an experiment was launched where 23 students applied UBR and checklist-based reading. It is concluded from the experiment that UBR is more *efficient* and *effective* in identifying faults, which are crucial and important from a user's point of view. The usage-based reading method guides the inspection effort towards finding the faults, which are of most importance for improving the user perception of the product.

The UBR method is based on information that is available in many development projects, i.e. use cases. The additional effort needed is the time to prioritize the use cases to guide the inspection effort to identify the most important faults. The method is easily integrated in an existing inspection process since it supports the preparation part of the process without requiring to change the compete process. Hence, we hope that the method will provide a support in software engineering, in striving towards better quality and more efficient engineering methods.

REFERENCES

[1] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", *IBM System Journal*, 15(3):182-211, 1976.

[2] A. F. Ackerman, L. S. Buchwald and F. H. Lewski, "Software inspections: an effective verification process", *IEEE Software*, 6(3):31-36, May 1989.

[3] A. Aurum, H. Petersson and C. Wohlin, "State-of-the-art: Software Inspections after 25 Years", *Software Testing Verification and Reliability*, 12(3):133-154, 2002.

[4] O. Laitenberger and J-M. DeBaud, "An Encompassing Life Cycle Centric Survey of Software Inspection", *Journal of Systems and Software*, 50(1):5-31, 2000.

[5] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, USA, 1992.

[6] Musa, J. D., "Operational profiles in software-reliability engineering", *IEEE Software*, 10(2):14-32, 1993.

[7] T. Thelin, P. Runeson and B. Regnell, "Usage-Based Reading – An Experiment to Guide Reviewers with Use Cases", *Information and Software Technology*, 43(15):925-938, 2001.

[8] T. Thelin, P. Runeson and C. Wohlin, “An Experimental Comparison of Usage-Based and Checklist-Based Reading”, to appear in *IEEE Transactions on Software Engineering*, 2002.

[9] A. Dunsmore, M. Roper and M. Wood, “Further Investigation into the Development and Evaluation of Reading Techniques for Object-Oriented Code Inspection”, *Proceedings of the International Conference on Software Engineering*, pp. 47-57, 2002.

[10]T. L. Saaty and L. G. Vargas, *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*, Kluwer Academic Publishers, Netherlands, 2001.

[11]T. Thelin, P. Runeson, C. Wohlin, T. Olsson and C. Andersson, “How much Information is Needed for Usage-Based Reading? – A Series of Experiments”, *Proceedings of the International Symposium on Empirical Software Engineering*, pp. 127 - 138, 2002.

[12]C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers, USA, 2000.