

# Towards a Perspective-Based Usage of Mobile Failure Patterns to Focus Quality Assurance

Konstantin Holl<sup>1</sup>(✉), Frank Elberzhager<sup>1</sup>, and Vaninha Vieira<sup>2</sup>

<sup>1</sup> Information Systems Quality Assurance, Fraunhofer Institute for Experimental Software Engineering IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany  
{konstantin.holl,  
frank.elberzhager}@iese.fraunhofer.de

<sup>2</sup> Computer Science Department, Fraunhofer Project Center for Software and Systems Engineering at UFBA, Federal University of Bahia,  
Av. Ademar de Barros, 500, Ondina, Salvador-BA 40110-170, Brazil  
vaninha@dcc.ufba.br

**Abstract.** The use of mobile applications for business tasks calls for effective quality assurance during development to prevent potential failures of the mobile application and the consequential costs. Essential activities of quality assurance are to inspect the requirements specification and to test the realized mobile application. Both activities ideally benefit from the knowledge of typical failure patterns in order to guide quality assurance engineers and to focus the quality assurance, i.e., to direct the attention of quality assurance on these failure patterns. For this purpose, a mobile-specific failure pattern classification was derived in previous work. In this paper, we introduce an initial quality assurance approach, which considers inspection and testing in a combined way and methodically uses the classification within the mobile context. The developed method FIT4Apps, which is based on the flexible, efficient reading technique perspective-based reading, allows purposeful use of the proposed failure pattern classification. As proof of concept, we developed a tool prototype, which generates basic perspective-based scenarios considering the derived failure pattern classification in order to support inspection and testing activities.

**Keywords:** Quality assurance · Mobile testing · Inspection · Perspective-based reading · Failure pattern classification

## 1 Introduction

The market for mobile devices is rapidly growing [1]. Furthermore, new mobile applications are continuously developed and shipped. Especially in the context of mobile business applications, several advantages are expected, e.g., higher availability, quick distribution of new information, and faster reaction time. However, as it is true for software development in general, a high quality of such applications is a necessity due to the risk of severe consequences, such as lost revenue or reduced efficiency. For example, consider a mobile application for production process control, which offers (i) information about the current status for monitoring the production, and (ii) decision support, like the collection and interpretation of environment parameters. An application failure

could lead to wrong decisions or unintended production effects, controlled by the mobile application. In this example, the usual consequences are very costly due to the high run-up time of production processes.

During the last decades, several quality assurance methods, techniques and tools were developed and are usually applied during software development. For instance, different inspection [2] and testing techniques [3] support, e.g., quality assurance engineers, inspectors and testers, to ensure the high quality of a software. However, as the mobile trend is relatively new, such quality assurance techniques are barely adapted to the new needs and challenges, such as:

- Mobile applications are developed in short, often agile, development cycles, and a fast time to market is expected [4].
- The set of requirements as the basis for the implementation and also for testing is changing during the development [5].
- Quality assurance is often unfocused due to unknown failure patterns in the mobile domain [6].

Consequently, compared to traditional quality assurance, mobile quality assurance has to be adapted to these new challenges by an approach, which aims eventually at being more efficient to be suitable for short development cycles, and being more effective by coping with a changing test basis and by focusing on typical failure patterns. Failure patterns represent the relation of fault aspects and their corresponding failures.

In an earlier publication [7], we presented how we derived a mobile-specific failure pattern classification. In this paper, we show a combined quality assurance method, which uses the failure classification. Furthermore, we demonstrate our method partially via a proof-of-concept using a tool prototype. The research question we address is the following:

RQ: How to use the derived mobile-specific failure pattern classification to focus quality assurance of mobile applications?

The structure of the paper is as follows: Sect. 2 describes the related work, where we provide an overview of quality assurance aspects regarding mobile applications. Furthermore, the section explains the failure pattern classification, which can be used to focus quality assurance, i.e., to direct the attention of quality assurance to these failure patterns. Section 3 presents our combined quality assurance method, which is tailored for mobile development. It also describes a detailed example, our tool prototype, and a discussion regarding the overall approach. Finally, Sect. 4 concludes our paper and gives an outlook on future work.

## 2 Related Work

The related work is divided into a general mobile quality assurance part, which describes recent publications about issues in this field, and into an outline of the recently developed failure pattern classification, which is fundamental to lead over to the development of the method to focus quality assurance of mobile applications.

## 2.1 Quality Assurance of Mobile Applications

Launching a mobile application development project often leads to questions regarding how the methods, techniques, and tools differ from classical software development projects, e.g., in the field of desktop applications. Methodological questions arise regarding the peculiarities of aspects, such as the application's architecture and security, and phases like requirements engineering and quality assurance. Those questions can usually be answered by using established approaches for non-mobile development projects. Also, existing tools are basically the same used in classical software development projects. For instance, recording and playback tools, which are popular in the area of mobile applications, are based on the already established concepts.

Differences in developing and using mobile applications are often related to technological aspects. The main peculiarities of applications on mobile devices such as smartphones and tablets are, according to Muccini et al. [8]:

- limited resources (e.g., battery)
- user interface (e.g., touchscreen)
- context awareness (e.g., mobile connectivity)
- diversity (e.g., devices and operating systems).

Muccini et al. [8] answered and confirmed their research questions, whether mobile applications are different from traditional ones and whether they require different and specialized new testing techniques, by discussing the peculiarities of mobile applications. They concluded with regard to mobile applications testing that there are many challenges “related to the contextual and mobility nature of mobile applications” and furthermore that “performance, security, reliability, and energy are strongly affected by the variability of the environment where the mobile device moves toward”.

Dantas et al. [6] considered mobile applications so specific that the derivation of testing requirements is necessary to enable efficacious and productive quality assurance. One of the conclusions drawn from the conducted interviews was that “most companies do not have a specific testing process for mobile applications” and that “tests are not executed systematically, and are defined based on previous experience or intuition of developers and testers”. Furthermore, it is possible to achieve effective quality assurance in spite of the given challenges by helping “the testers to find specific errors on mobile environment”.

Franke and Weise [9] claimed that the challenges of mobile applications development “make great demands on software and ask for specific approaches and methods for quality assurance” and that quality assurance methods “vary between different kinds of software, in particular between software for mobile and desktop applications”. Their contribution describes the provision of a software quality framework based on finding and defining key qualities of mobile applications.

Overall, existing methods and insights regarding quality assurance in mobile application development projects – which are typically conducted using ad hoc, classical (e.g., regression testing within the waterfall model), or agile state-of-the-art approaches (e.g., test first within Scrum) [4] – have increased in importance as a key discipline, but lacks specialized methods, expertise, and environment [10] and are often not effective due to the missing focus on the peculiarities of mobile applications.

## 2.2 Focus Quality Assurance on Mobile-Specific Failure Patterns

Due to the problem that there is no established information about typical mobile application failure patterns, our previous contribution [7] described how to collect and to classify frequently mentioned fault aspects of mobile application failures. Furthermore, approaches for creating classifications of defects respectively failures found in the literature were described like the schemes of Li et al. [11] and Mauser et al. [12], which were encouraged, among others, by the ODC [13] and the IEEE Standard Classification for Software Anomalies [14]. Considering the experiences, made in multiple mobile application development projects, the previous contribution explained and classified types of typical mobile application faults and failures.

The project experience comprised four mobile applications, which have been developed within six months, in 2013. The quality assurance team analyzed the failure reports of this project with the intention to create a classification for failures and typical fault aspects. In our previous contribution [7], we used the term *fault* as the origin of a failure, and the term *fault aspect* as the focus of a test case that leads to a specific failure of the mobile application, which is called fault class (as part of the failure pattern classification) in the current contribution.

The state of the art regarding typical mobile failures and typical fault classes was captured based on publications by a literature review according to Kitchenham [15]. The review was done for the time span between the year 2006 and the contribution date in 2014, what resulted in 26 publications, which could be identified to support the development of the classification (see Fig. 1).

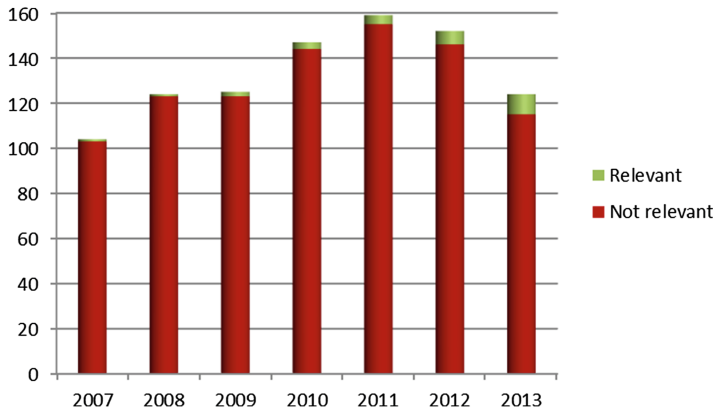


Fig. 1. Distribution of publications containing related work

In our earlier publication, the fundamental research question “How to classify detected mobile-specific failures?” was answered based on project experiences and related work regarding failure classifications. The ensuing research question “Which typical fault aspects exist and how to integrate them into the failure classification?” was answered based on project experiences and the literature review regarding mobile failures [7]. We concluded that the “work on both research questions resulted in overlapping

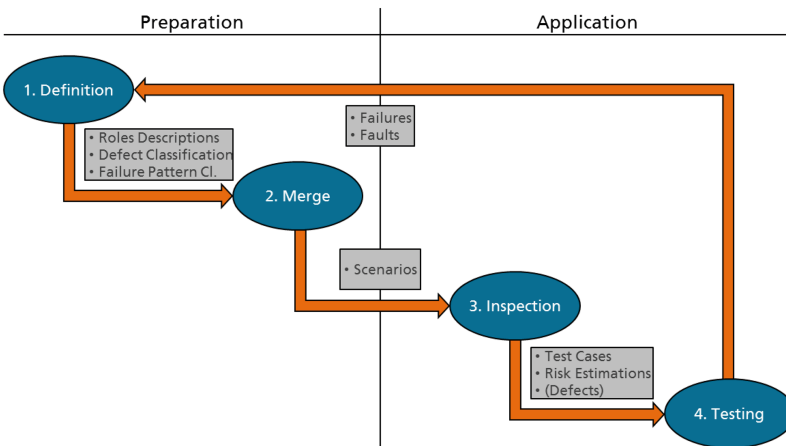
information on the one hand and complementary information on the other hand. Combination with the knowledge of other classification schemes enabled the creation of a basic mobile-specific failure classification including categorized typical fault aspects [...]. This classification could be successfully evaluated and optimized by applying it to multiple mobile application developments.” [7] Due to the defined relations between failures and fault aspects, we call this classification for short: failure pattern classification.

### 3 A Method to Focus Quality Assurance of Mobile Applications

The creation of the failure pattern classification led to the question how to use it within quality assurance. This section explains the developed method, gives an example of its application and describes a first prototype for a partial automation.

#### 3.1 Description of the FIT4Apps Method

A quality assurance method that addresses the main reasons for the insufficiency of common approaches can be approached by combining testing activities with an inspection technique, both influenced by the mobile-specific failure pattern classification. Therefore, the method FIT4Apps (Focus Inspections and Tests for Mobile Applications) was developed.



**Fig. 2.** The process model of the FIT4Apps method.

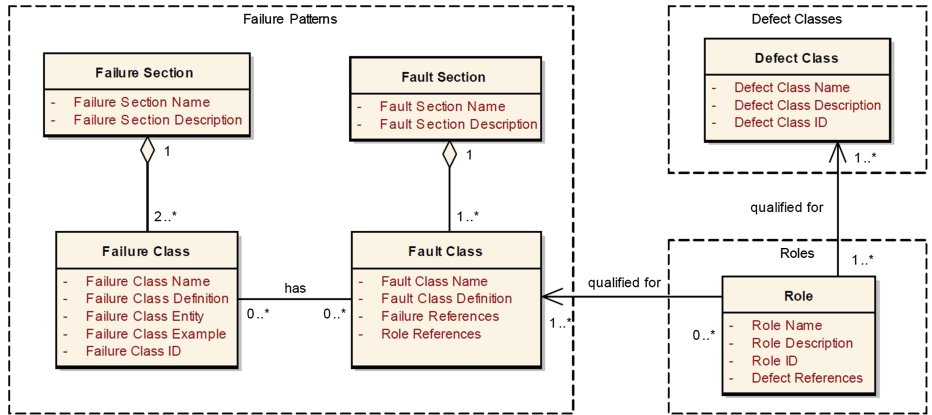
FIT4Apps contains two preparation steps, *Definition* and *Merge*, which enable the application of the method: *Inspection* and *Testing* (see Fig. 2).

**1. Definition.** The method's first step is the definition of descriptions of roles, which are taking part in the development, the defect classification regarding potential deficiencies of the requirements specification, and the failure pattern classification. The definition comprises also the relations between these artifacts. The classified **Failure Patterns** are structured by:

- **Failure Sections** (*Behavior, Design and Content*), which categorize the
- **Failure Classes** (e.g., *Interaction Element, Transition, and Static Text*), which comprise by their entities (*wrong, missing and extra*) aspects “in which a system or system component does not perform a required function within specified limits” [14]. The failure classes are related to
- **Fault Classes** (e.g., *Network Disconnect, Low Battery, and Screen Orientation*), which comprise typical fault aspects [7] and belong to several
- **Fault Sections** (e.g., *Connection, Energy, and User Interface*) for the purpose of categorization.

Furthermore, the **Failure Patterns** are related to:

- **Roles** (*Tester, Designer, and User*), which are each qualified for the detection of certain failure patterns and defects of certain
- **Defect Classes** (*Missing Information, Ambiguous Information, Extraneous Information, Inconsistent Information and Incorrect Facts*), which comprise potential deficiencies of requirements specifications [16].



**Fig. 3.** Relation between failure patterns, roles, and defect classes.

The failure pattern classification contains three failure sections, which were split into nine failure classes. Each failure class has three entities, which were related to a maximum of 4 of 21 defined fault classes. Overall, 44 failure patterns (i.e., typical relations between failure classes and fault classes) were defined [7]. As part of the current contribution, the failure pattern classification was related to the typical roles and defect classes to enable a perspective-based usage of the derived classification (see Fig. 3).

**2. Merge.** The defined artifacts of the first step are merged to perspective-based reading (PBR) scenarios structured according to Shull et al. [17]. This represents a convenient reading technique for inspecting the requirements specification in the mobile context, due to its efficiency and flexibility [17]. Each scenario contains:

- an introduction, to understand and, if necessary, to adopt the characteristics of a special role,
- instructions, which describe the tasks the reader has to fulfill while inspecting the requirements specification,
- and questions, which indicate aspects of the requirements specification to be checked by the reader concerning the defect classes and failure patterns.

The merge occurs based on the defined relations. For instances: the *tester* is qualified to detect *missing information* in the requirements specification; a *changing orientation* of the mobile device can lead to a *missing interaction element* in the user interface.

**3. Inspection.** As first step of applying the method, the inspector of each perspective reads every PBR scenario belonging to his role. The definition of typical roles taking part in the project is intended to support the inspection due to the separation of the points of view of the different inspectors. This intention aims at less overlapping results regarding the findings. Typical roles would be, for instance: a user, who is determined to use the mobile application after it has been fully developed, marketed, and installed; or a tester, who operates the mobile application under specified conditions, observing and recording the results and evaluating some aspects of the mobile application. Inspectors of every role are instructed to perform perspective-specific tasks. This means in case of the user's perspective, to conduct risk estimations with focus on given failure patterns. In case of the tester's perspective, it means to derive test cases with focus on given failure patterns. Furthermore, every role has to answer perspective-specific questions to find defects in the requirements specification while following the instructions. The motive of this technique is, to perform an inspection in parallel to those tasks, which have to be performed anyway by this role.

**4. Testing.** As last step of the method, the derived test cases, which have got the focus on the mobile-specific failure patterns, are composed to a test suite which is part of the development project's test plan. According to the test plan's strategy, the test cases are executed. This leads depending to the test level to the detection of faults and failures in the implemented mobile application. The knowledge about the findings are used to enhance the definition of the failure pattern classification for future development iterations. Furthermore, findings of unit tests, which are not part of the quality assurance method, support as well the enhancement of the failure pattern classification.

### 3.2 Exemplified Application of the Perspective-Based Inspection

To illustrate the inspection of the quality assurance method, two scenarios are presented in Tables 1 and 2 for quality assurance performed for a possible mobile application to be used for production process control. The requirements of this application are specified by screen mockups and free text. In this example, the defect classes and

failure classes are underlined to illustrate that these contents are variable according to the merge process and based on the failure pattern classification.

**Table 1.** Scenario “User Perspective”.

<b>Introduction</b>	You are determined to use the mobile application after it has been fully developed, marketed, and installed. The end goal of the mobile application is to be useful to the consumer.
<b>Instruction</b>	Identify and mark parts of the requirements specification with an estimated high failure impact with “high risk” in the case of a <u>missing interaction element</u> .
<b>Question</b>	Are there parts of the specification with <u>extraneous information</u> ?

A user, or an inspector taking the user’s perspective, reads the introduction to adjust his mind to this perspective. Then he follows the instruction and answers the question shown in Table 1. The failure class *missing interaction element* is linked to a typical fault class, which will be used in the next scenario. The result in this example of the production process control application is that the inspector marks a designated status bar, which has to show critical warnings regarding the production process, with *high risk*. He does not find any extraneous information and quits the scenario.

**Table 2.** Scenario “Tester Perspective”.

<b>Introduction</b>	You operate a mobile application under specified conditions, observing and recording the results and evaluating some aspects of the mobile application.
<b>Instruction</b>	Derive test cases considering the mark “high risk” due to a <u>missing interaction element</u> . Focus on failures that could be related to <u>changing orientation</u> or <u>screen resolution</u> .
<b>Question</b>	Are there parts of the specification with <u>missing information</u> ?

A tester of the development project, respectively an inspector taking the tester’s perspective, reads the introduction and follows the instructions of the scenario shown in Table 2. He considers the *high risk* mark for the derivation of test cases. Hence he knows that the marked part of the requirements specification is critical in case of the failure *missing interaction element* and that typical fault classes causing this failure are: a *changing orientation* or *screen resolution*. Consequently, he derives a test case, which will be focused on the typical failure pattern (shown in Table 3).

**Table 3.** Focused test case.

<b>Precondition</b>	Main screen viewed in horizontal orientation.
<b>Action</b>	Change orientation to vertical.
<b>Postcondition</b>	Status bar information completely visible.



While deriving the test case (Table 3), the inspector in the tester’s role answers in parallel the question whether there is any *missing information*. In this example, it is the absence of information regarding how the application has to react to an orientation change. Several questions will arise depending to the experiences of the inspector: Is it allowed to cut interaction elements off? Is there a priority of displaying elements? Should vertical orientation be deactivated? Should the screen be scrollable?

The defects found in the requirements specification will be sent to the requirements engineers or designers and the test case will be performed as part of the system test by the tester, who thus has a focused test case to detect failures in the implemented mobile application.

### 3.3 Tool Prototype

In the previous example, the PBR scenarios were created manually. In order to enhance the efficiency of applying the method, the tool FOCUST (Focus and Control by Updated Scenarios and Tests) was initially developed. FOCUST represents a partial proof of concept by realizing the merge step of the method (see Sect. 3.1) in an automated way.

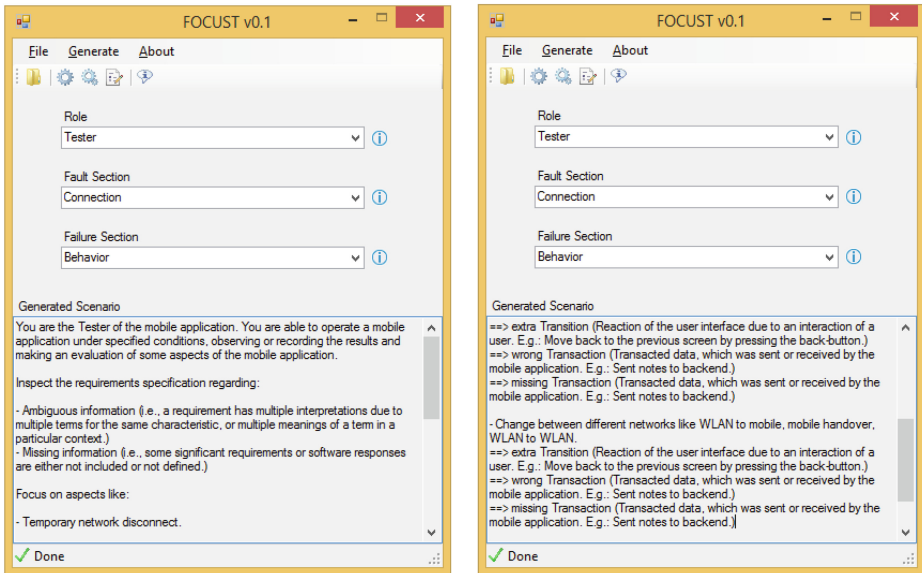


Fig. 4. Output example of the tool prototype for the tester’s perspective.

FOCUST has an interface to the failure pattern classification including the relation to roles and defect classes. The user interface of FOCUST offers the selection of a specific role like *Tester* (see Fig. 4). Furthermore, the fault section (here: *Connection*) and the failure section (here: *Behavior*) can be selected. Based on that input, the

prototype generates basic PBR scenarios by listing the matching possibilities considering the given input. In future work, these basic scenarios will be linguistically enhanced according to Sect. 3.2.

### 3.4 Discussion

The development of the method and the partial implementation as a proof of concept revealed on the one side possible benefits due to the application of the method and led on the other side to open issues.

**Benefits.** The main advantage compared to common state-of-the-art approaches is that the developed quality assurance method focuses by different perspectives on typical failure patterns supposed to lead to a higher quality assurance effectiveness (see challenges in Sect. 1). Together with the PBR technique, the failure pattern classification [7] enables multiple benefits by being usable for inspection and testing activities through PBR scenarios. These PBR scenarios are efficient due to less overlapping of the findings of different inspectors [17]. A designer usually has another perspective and hence other findings than a tester or a user. Another benefit of those scenarios is the optimized work process during their performance. For the tester, this means that he derives test cases for those parts of the requirements specification that he is currently inspecting (i.e., checking for defects).

A defect in the requirements specification, such as ambiguous information, can cause a failure in the implemented mobile application. Hence, information about a found defect could be used to adjust the failure pattern classification as part of the fault classes and can be considered during the derivation of test cases. This can result in interaction between the testing and inspection activities and hence in a control loop (see Fig. 5), as both are related to the commonly used failure pattern classification.

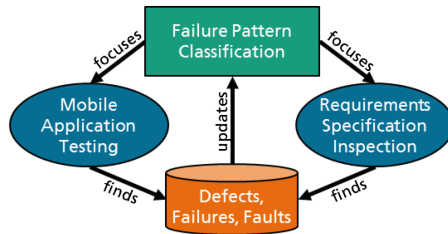


Fig. 5. Quality assurance control loop based on the information flow regarding the findings.

**Open Issues.** A crucial open issue is the missing basis to appraise the method's applicability within mobile application development projects with typical challenges like short iteration-cycles and a changing test basis. Insights for an evaluation could be reached by a first case study using the developed quality assurance method.

Before, several other issues need to be solved. The scenarios cannot be generated completely due to a missing linguistic template. Furthermore, we need a determination of reasonable integration of the scenarios into the working process of the tester,

inspector, etc. That comprises, among others, the question “how to achieve an adequate cognitive load regarding the usage of the scenarios”.

## 4 Conclusions and Future Work

This work resulted in a FIT4Apps method to focus quality assurance of mobile applications and encourages research on several topics as future work.

### 4.1 Conclusion

The research question, “How to use the failure pattern classification to focus quality assurance?”, was answered by the development and exemplified partial application of the proposed quality assurance method, which uses the failure pattern classification presented in our previous contribution [7] (see Sect. 2.2). The combined testing and inspection method FIT4Apps based on the perspective-based reading technique enables the usage of the failure pattern classification. The resulting scenarios aim on focused defect detection by the inspection method and focused derivation of test cases considering typical failure patterns including their risks. The inspection may be applied to changing requirements specifications, working as a control loop, which could enable focused rework of quality assurance activities with reduced effort.

The developed method was partially implemented as a tool to support a quality assurance process through information processing, such as the generation of basic scenarios. These scenarios are actually based on previously defined input, like the derived failure pattern classification of the previous contribution [7].

Overall, we assume that the developed method could enable a higher quality assurance effectiveness, which can reduce the risk of failures remaining in mobile applications and hence their costly consequences.

### 4.2 Future Work

As future work, the integration of a defect causal analysis into the method’s control will be investigated considering the possibilities of defect prevention.

One of the most important further steps, besides the enhancement of the prototype, is the evaluation of the method to get evidence for its effectiveness and to obtain the possibility to compare it with other approaches (such as testing using defect taxonomies [18]). The primary metric will be the detection rates of defects and failures. This will be conducted first by a questionnaire consulting the target group and later by an experiment. One future research question will be about the indications, which exist that such our tailored quality assurance method is more efficient, stable and focused against a changing test basis.

**Acknowledgment.** The research described in this paper was conducted in the context of the Fraunhofer Project Center for Software and Systems Engineering at UFBA, a joint initiative of Fraunhofer Society and the Federal University of Bahia in Brazil, with support from the Bahia State Government.

## References

1. Portio Research: Mobile applications futures 2013-2017, analysis and growth forecasts for the worldwide mobile applications market, pp. 22–28 (2013)
2. Aurum, A., Petersson, H., Wohlin, C.: State-of-the-art: software inspections after 25 years. *Softw. Test. Verif. Reliab.* **12**(3), 133–154 (2002)
3. Juristo, N., Moreno, A.M., Vegas, S.: Reviewing 25 years of testing technique experiments. *Empir. Softw. Eng.* **9**(1–2), 7–44 (2004)
4. Linz, T.: Testing in Scrum: A Guide for Software Quality Assurance in the Agile World (original title: Testen in Scrum-Projekten: Leitfaden für Softwarequalität in der agilen Welt, dpunkt.verlag), 1 edn. (2013)
5. Wasserman, A.: Software engineering issues for mobile application development. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER), pp. 397–400. ACM, New York (2010)
6. Dantas, V.L.L., Marinho, F.G., da Costa, A.L., Andrade, R.M.C.: Testing requirements for mobile applications. In: 24th International Symposium on Computer and Information Sciences (ISCIS), pp. 555–560 (2009)
7. Holl, K., Elberzhager, F.: A mobile-specific failure classification and its usage to focus quality assurance. In: Euromicro Conference Series on Software Engineering and Advanced Applications (2014) (accepted)
8. Muccini, H., Di Francesco, A., Esposito, P.: Software testing of mobile applications: challenges and future research directions. In: 7th International Workshop on Automation of Software Test (AST), pp. 29–35 (2012)
9. Franke, D., Weise, C.: Providing a software quality framework for testing of mobile applications. In: IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), pp. 431–434 (2011)
10. Makarand, T., Buenen, M.: World quality report key findings, executive summary. World Quality Report 2013–2014, Capgemini, Sogeti and HP, Fifth Edition, pp. 6–8 (2013)
11. Li, N., Li, Z., Sun, X.: Classification of software defect detected by black-box testing: an empirical study. In: Proceedings of Second World Congress on Software Engineering (WCSE), vol. 2, pp. 234–240. IEEE (2010)
12. Mauser, D., Klaus, A., Holl, K., Zhang, R.: GUI Failures of in-vehicle infotainment: analysis, classification, challenges and capabilities. *Int. J. Adv. Softw.* **6**, 142–154 (2013)
13. Chillarege, R.: Orthogonal defect classification. In: Lyu, M.R. (ed.) *Handbook of Software Reliability Engineering*, pp. 359–399. McGraw-Hill, New York (1996)
14. IEEE Standard Classification for Software Anomalies, IEEE Std., Rev. 1044-2009 (1994)
15. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report, version 2.3, Software Engineering Group (2007)
16. IEEE Recommended Practice for Software Requirements Specifications, IEEE Std., Rev. 830-1993 (1998)
17. Shull, F., Rus, I., Basili, V.: How perspective-based reading can improve requirements inspections. *Computer* **33**(7), 73–79 (2000). (IEEE Computer Society Press)
18. Felderer, M., Beer, A.: Using defect taxonomies for testing requirements. IEEE Softw. IEEE Computer Society Digital Library (2014)

Software Quality. Software and Systems Quality in  
Distributed and Mobile Environments  
7th International Conference, SWQD 2015, Vienna,  
Austria, January 20-23, 2015, Proceedings  
Winkler, D.; Biffli, S.; Bergsmann, J. (Eds.)  
2015, IX, 133 p. 31 illus., Softcover  
ISBN: 978-3-319-13250-1