Challenges in Testing Context Aware Software Systems

Santiago Matalonga¹, Felyppe Rodrigues², Guilherme H. Travassos²

¹Universidad ORT Uruguay. Cuareim 1471, 11100, Montevideo, Uruguay
²Universidade Federal do Rio de Janeiro/COPPE/PESC. Rua Horácio Macedo, 2030 Cid. Universitária Rio de Janeiro–RJ 21941-450

¹smatalonga@uni.ort.edu.uy, ²{felyppers,ght}@cos.ufrj.br

Abstract. Context aware software systems (CASS) are becoming pervasive in our lives. Nevertheless, it is not clear whether traditional (no-context aware) software testing techniques are adequate for testing CASS. Therefore, a quasi-systematic literature review was used to identify 11 relevant sources that mentioned 15 problems and 4 proposed solutions, which were analyzed and classified into 3 groups of challenges and strategies for dealing with CASS testing. Additionally, some recommendations for testing such software applications with the currently available testing technologies are presented. However, we argue that new context aware testing techniques need to be developed in order to assure the quality of CASS.

1. Introduction

Context aware systems are becoming even more pervasive in our lives, from recommendation software using our information to select songs to wearables, monitoring our biometric information and acting in response to it. At this work, context is “any piece of information that may be used to characterize the situation of an entity (logical and physical objects present in the system’s environment)”, and context aware is the “dynamic property representing a piece of information that can evolutionarily affect the overall behavior of the software system in the interaction between the actor and computer” [Mota 2013]. Therefore, when one or more behaviors of a software application are affected by information it can sense from the context where it is being executed, such application is defined to be context aware software application (CASS).

Testing allows the observation and validation of software application's behaviors. However, one of the issues when designing software tests is always concerned with the tradeoff decision between the testing set completeness and the resources available to execute the software testing process. When context awareness features come into play, and at the very least, the available test space from which test cases are selected can significantly increase, consequently increasing the testing partiality and in some sense making unfeasible the using of traditional testing technologies (those ones used to test no context aware software).

In order to identify the available knowledge on testing CASS, a quasi-systematic literature review (qSLR) had been undertaken under the umbrella of CNPq - CAcTUS project. The qSLR protocol execution revealed 11 technical papers that provided
evidence regarding challenges and solutions on testing CASS. Analyzing these selected sources, a set of 15 issues and four proposed solutions were identified. Furthermore, by means of applying qualitative analysis techniques, 15 problems were abstracted intending to represent two distinct challenges: (1) How to deal with CASS testing within “Device and hardware constraints” and (2) How to deal with all possible combinations of “context variations” when designing test cases. These two abstractions enabled us to observe that three out of the five proposed solutions have been experimentally evaluated deal with the challenge of mitigating the “context variations” problem. No experimental results dealt with other aspects of context aware like “Device and Hardware constraints”.

Therefore, upon analysis of these results, we claim that new alternatives for testing CASS need to be developed in order to properly evaluate the quality of such software systems. Current identified proposals tend to fix the value of context variables during test case design. We argue that these strategies are limited in light of the identified challenges (especially the problems associated with “Context Variation”), since the tester will always be faced with trade off decisions between coverage and test effort/cost. Therefore, we envision the emergence of Test Design and Execution techniques especially targeted at CASS, where the evaluation of the unit under test is performed under the unconstrained variation of the context.

This paper is organized as follows. Section 2 presents the research method, with an initial analysis of the selected technical literature in section 2.1. A summary of the selected sources is provided in section 3. Section 4 presents issues practitioners have to take into account when applying the identified solutions to their environment. In section 5 we present recommendations for testing CASS. Section 6, presents our vision for context aware software testing. Finally, in section 7 we conclude this paper and call for new directions of research for context aware software testing.

2. Research method

In order to acquire the state of the art in challenges on testing CASS, a quasi Systematic Literature Review (qSLR) has been undertaken. A qSLR follows the same formalism of a Systematic Literature Review, but due to the maturity of the field under study it is still not possible to aggregate data through comparative meta analysis [Travassos et al. 2008]. We make this distinction when the technical literature of the phenomenon under study cannot provide a comparison baseline, yet the rigor and completeness of the secondary study are not affected. A key strength of this research strategy is its capacity to acquire evidence-based knowledge. Even though, for the sake of brevity and simplicity, we do not go into the qSLR protocol details [Rodrigues et al. 2014]¹, this section describes the highlights.

---

¹ See the full protocol available at CACTUS portal http://lens.cos.ufrj.br/caactus
The main research question is "What are the existing methods for testing context aware software systems?". The search string used to retrieve the primary sources from the search engines was structured using the PICO strategy [Pai et al. 2004]. Population was defined to be “Sensibility to context”. Intervention was defined to be “Software testing”. No Comparison is available. Outcome was defined to be “Methodology”. The resulting set of keywords was:

"context aware" OR "event driven" OR "context driven" OR "context sensitivity" OR "context sensitive" OR "pervasive" OR "ubiquitous" OR "usability" OR "event based" OR "self adaptive" OR "self adapt" AND "software test design" OR "software test suite" OR "software test" OR "software testing" OR "system test design" OR "system test suite" OR "system test" OR "system testing" OR "middleware test" OR "middleware testing" OR "property based software test" OR "property based software testing" OR "fault detection" OR "failure detection" OR "GUI test" OR "Graphical User Interfaces test" AND: model OR "metric" OR "guideline" OR "checklist" OR "template" OR "approach" OR "strategy" OR "method" OR "methodology" OR "tool" OR "technique" OR "heuristics"

The search engines were selected due their coverage and previous experience of the researchers. These were IEEEXplore, Scopus and Web of Science. Google Scholar and ACM Digital library where discarded because the implementation of their search algorithm does not favor the repeatability of the results [Gehanno et al. 2013]. A total of 1820 potential sources were retrieved using the aforementioned search string in the search engine. The research method application (filtering through applying the inclusion and exclusion criteria) narrowed them down to the 11 technical sources described in this paper (see section 3). The raw numbers of the filtering process is explained in the fact that a qSLR looks for evidence based in the application of the scientific method. Many relevant and interesting research has been carried out in the topic of testing context aware software systems, yet, our interpretation is that only those 11 primary sources unequivocally report that their research method applies the scientific method.

2.1 Data Analysis

From each of the 11 technical primary sources (Column “Source’in Table 1) we extracted the Problems and Solutions that the authors mentioned context awareness brings to testing. We further distinguished them among those that were part of the experimental setting described in the source from those that were mentioned or cited.

<table>
<thead>
<tr>
<th>Source</th>
<th>Mentioned Problems</th>
<th>Proposed Solutions</th>
<th>Experimentally Observed Problems</th>
<th>Experimentally evaluated Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Alsos and Dahl 2008]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Amalfitano et al. 2013]</td>
<td>Resource scarceness, Variety of running conditions, Integration of devices</td>
<td>Extensive platform testing</td>
<td>Variety of running conditions</td>
<td>Exploit defined scenarios with mutation</td>
</tr>
<tr>
<td>[Jiang et al. 2007]</td>
<td>Devices resource constraints</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Canfora et al. 2013]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Mordes et al. 2006]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Ryan and Gonsalves 2005]</td>
<td>Diversity of hardware, Difficulties in collecting user reactions</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Satoh 2003</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
[Tse et al. 2004] Prohibitive number of possible situations
Behavior changes when context changes

[Wang et al. 2007] Determining and controlling when and how to
feed a series of changing context
Integration with thick middleware
Anticipation of all relevant context changes, when they could impact behavior
Asynchronous communication of context information

unprecedented details to process

[Wang et al. 2014] -

In Table 1, the reader can observe that only three proposals (column “Experimentally Observed Problems”) deal with one of the identified problems (column “Mentioned Problems”). In order to elaborate further on this result, we applied content analysis [Weber 1990]. Through this technique, the extracted data presented in Table 1 was analyzed for proximity and sorted in chunks in order to look for and derive abstractions. The results of this analysis are presented in graph format in Figure 1.

![Figure 1 Relationship between challenges and solutions](image-url)

- Device hardware and network constraints. The star-like layout of connected nodes show that no proposed solution has been experimented for this challenge.
• Context variation; which is the abstraction standing for several identified problems (for instance "Many usage scenarios" or "diversity of hardware") that deal with the problems associated with the impact of variation of one or more context variables (for instance, associations from the problem nodes to the solutions show how the researchers have dealt with the problem).

This analysis also gave way to the possibility of classifying the selected technical literature in one of three categories: Usability, for those sources dealing with testing problems or evaluating CASS usability; Tool validation, for those sources which aims at evaluating tools for testing CASS and; Solve Challenge, for those sources providing experimental data in solving one of the identified problems. The following section presents the selected technical sources description grouped by this categorization.

3. Summary of Findings

3.1 Evaluating usability of CASS

Ryan and Gonsalves (2005) present an evaluation on the effects of device diversity and mobility on the usability of CASS. Their experimental design involves the evaluation of the usability of the same functionality in four different platforms. The technology used for this experiment has already been phased out; therefore, generalization is no longer possible. Their results show that application type was not a factor that influenced user satisfaction, and that location is a factor for the preferences of users regarding platform selection.

Alsos and Dahl (2008) present the design and construction of a simulated environment for usability testing in the Healthcare Information Technology (HCIT) solutions. They argue that generating controlled environments for testing usability of HCIT solutions is expensive and their experiment is oriented towards identifying usability problems in the laboratory or in the intended deployment environment. The only context aware variable concerned with the experiment is the indoor location within the healthcare facility.

Canfora et al. (2013) present the design and development of a platform designed to automate the usability evaluation of devices in the field. Their area of application is a proprietary brand of embedded software, and their study aims at identifying and understanding the variation of usage scenarios for these proprietary devices running in the field. Their results show how their solution can identify more user experience problems than using survey-based tools for data collection.

3.2 Tools for testing CASS

Merdes et al. (2006) present the design of a similar framework for testing CASS. This proposal is designed around the testing of resource utilization and constraints of mobile devices. They define test cases by constraining the access to shared resources of CASS running in smart devices.

Jiang et al. (2007) present the design and evaluation of a black box testing tool for the automation of software testing for context aware applications in smart devices. The
implementation is provided by the Symbian platform—which is no longer relevant in the smart phone market. Nonetheless, their results show that there is benefit in developing such a tool to automate black box testing.

Wang and Chan (2009) propose the metric of Context Diversity as a proxy for the coverage of CASS under test. This metric was evaluated against a controlled set of Test Suites designed by the authors. In Wang et al. (2014), their hypothesis is further evaluated with a huge test suite generated by applying mutation algorithms. The results of both papers sustain the hypothesis that Test Suites ranking higher in context diversity also attain more coverage.

3.3 Proposed solutions for the challenges on testing CASS

Satoh (2003) is concerned with the location awareness of devices. To solve this issue, a sandbox environment (emulator) had been created. Satoh presents a case study of the proposed emulator by describing a scenario where the main use case is the acquisition of remote printing capabilities.

Tse et al. (2004) is concerned with the context variation problem. Their strategy is to dynamically generate test cases via mutation. In addition to this, Tse et al. (2004) provide a mechanism for defining the expectation of the generated test cases output by abstracting relationships between inputs and outputs regarding a predefined set of test cases.

Wang et al. (2007) present another framework for testing CASS. This framework is based on the idea of CAPPs. By identifying these CAPPs the authors suggest the definition of test cases by using structural analysis testing. These CAPPs are identified in the application code, so that the tool can exploit several pre-defined strategies to generate new test cases by navigating the call structure. This approach is similar to [Tse et al. 2004] and [Amalfitano et al. 2013]) in that it first relies on human-generated test cases, and then use computing power to dynamically generate more test cases in order to mitigate the challenge of“Context Diversity’.

Amalfitano et al.(2013) design test cases taking the context into account. They propose a two-step process on which the tester first designs test cases using context variables as input, and then applies mutation testing techniques to dynamically generate more test cases. They present a case study where they implement this approach to deal with an android device with four apps downloaded from the android store market.

4. Threats to the generalization of the identified approaches.

According to the results presented in the previous section, a practitioner with the need of testing CASS is faced with a limited set of empirically validated options, even then presenting limited coverage.

First of all, the identified strategies for evaluating usability of CASS all rely on the investment in infrastructure. These investments do not appear to be reusable for a third party. For instance, the results by Ryan and Gonsalves (2005) need to be reviewed since
the technology used is no longer relevant. Merdes et al. (2006)'s solution was implemented for proprietary hardware, and Alsos and Dahl (2008)'s approach requires extensive investment in physical infrastructure.

The proposal by Wang and Chan (2009) has been evaluated with the aim of helping practitioners to determine the expected test suite coverage. They do not provide guidance to generate test suites, however the practitioner can use their proposed metric to evaluate whether new test cases enhance the system under test coverage.

Finally, the identified technical literature has proposed and empirically evaluated solutions to the challenge of dealing with the “context diversity” ([Satoh 2003], [Tse et al. 2004], and [Amalfitano et al. 2013]). Despite the importance of these initial evidence, some limitations can be observed. For instance, the approach by Satoh (2003) had been empirically evaluated only for the context variable "location". The other approaches suffer from the following two limitations. Firstly, they rely on software platforms, which can become outdated due to technology shifts; Secondly, they rely on mutation to produce test suites, which represents a software testing technique that might be useful to evaluate robustness, but does not guarantee functional correctness.

5. Recommendations for testing CASS.

This section presents recommendations for testing CASS that can be abstracted from the identified technical literature aiming at to partially mitigate the lack of specialized software technologies to support such software applications testing.

**Build quality in.** This recommendation can be enacted with: *Assure the functionality of the software system.* Do it by testing the functional requirements using traditional test cases. This is a pre-requisite in some of the proposals presented in the papers ([Satoh 2003], [Tse et al. 2004], [Amalfitano et al. 2013]). *Incorporate context aware variables in traditional test case techniques.* For instance, subject to the limitations discussed in the following section, context aware variables can be used in Branch Testing design technique [Wang and Chan 2009]. *If possible, automate the test cases definition and execution.* The automation of test cases and their execution, when feasible, will reduce the overhead in regression testing and enable the execution of dynamic techniques like random or mutation testing [Tse et al. 2004].

**Identify context variables in the requirements to evaluate where things could go wrong.** It is highly likely that defects in CASS will come with unforeseen values feed to the system by the context [Wang et al. 2007], therefore identifying these CAPPS could be complemented by the use of defensive coding techniques.

**Use automatic dynamic execution tools.** Tools that automate the generation of execution scenarios—like those that provide Random Testing capabilities—(in spite that they cannot assure the correctness of the application) are useful to evaluate the robustness of the application, and has been observed useful for support the testing of CASS [Amalfitano et al. 2013].
6. Limitations of current approaches and CASS testing perspective

Even though the previous recommendations are applicable at the moment, they do not solve all the issues available in CASS. As seen in Table 1, the experimentally validated solutions only deal with the challenge of context variation. We feel this only partially addresses the reality of testing CASS. Although the technical literature has clearly identified other problems and challenges, we argue that the available identified solutions offer limited relief to the practitioners. From our list of recommendations: the first one (build quality in) only go so far as to assure functional correctness. It does not evaluate how different context variables can interact or vary during CASS execution. The second recommendation, offers some degree of mitigation to the limitation of the first one, but it is limited by the ad-hoc capacity on identifying CAPPS. And finally, automatic testing tools cannot warrant functional correctness, they can only address reliability.

Therefore, we envision the possibility of context aware software testing for CASS. It represents the evolution of test design techniques that do not design test cases to mitigate the challenges context awareness brings to software testing, but in contrast take advantage of the context variation to better cover the CASS under test.

7. Conclusions and on going works

Context aware software systems are mainstream. This means that software organizations are developing and deploying CASS. In order to identify the current available challenges for testing such systems, a qSLR was undertaken with the aim of finding empirical evidence on how CASS can be tested. The results of this review show that practitioners can count with a limited set of options to face the challenges the testing of CASS can bring. This paper has grouped the proposed solutions in three different sets.

First of all, in order to evaluate the usability of CASS, researchers have relied on the construction of expensive infrastructures to simulate selected aspects of the context in which the application is expected to run. Secondly, in order to manage an expected big set of test cases, researchers have built frameworks and infrastructure enabling practitioners to manage and generate massive amount of test cases in order to deal with the huge number of input combinations that context awareness brings into a software application. These strategies have been useful in the context of the presented case studies, but need constant update to keep up with the changes in technology and more research to mitigate the current threats to the generalization of the results. Finally, according to our findings, the variation in context values is the only challenge that has been explicitly tackled by research, although they do not take into account the context variation when performing the test cases. The common strategy from the identified research is to build a convenient set – in size and coverage - of test cases, and take advantage of a computer processing power to dynamically generate more test cases. But the drawback of these automation tools is that they do not necessarily increase coverage, and even so, the increased coverage does not necessary guarantee correctness in functionality.
Based on these challenges, we have abstracted a set of three recommendations that are readily available for practitioners who need to test CASS. First off all, practitioners should not rely only on testing to assure CASS quality (build quality in). Then, they should design the test cases by taking special consideration into the context variables. Finally, if possible, we recommend practitioners to take advantage of automatic dynamic execution tools to evaluate the robustness of the CASS under test.

Nevertheless, we believe that these results are not enough to evaluate the quality of the resulting software. The random addition of test cases, without a competent Test Oracle to verify the results, cannot guarantee neither quality nor correctness of functionality. Therefore, we claim that new software testing techniques need to be developed embracing the challenges concerned with context aware software testing. These techniques should not try to mitigate the challenges of context variation, but to accept the context changes as a reality of such software systems. These context aware software testing techniques should take advantage of this variation in order to ensure the quality of CASS.

8. Acknowledgements

This work was supported by "CAcTUS - Context-Awareness Testing for Ubiquitous Systems" project partially financed by CNPq–Universal 14/2013 Number 484380/2013-3. Prof. Travassos is a CNPq Researcher. Prof. Matalonga is supported by Banco Santander Uruguay grant for young researchers.

9. References


