Testing context-aware software systems
Matalonga, Santiago; Travassos, Guilherme Horta

Published in:
Proceedings of the 31st Brazilian Symposium on Software Engineering

DOI:
10.1145/3131151.3131190

Published: 21/09/2017

Document Version
Peer reviewed version

Link to publication on the UWS Academic Portal

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the UWS Academic Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact pure@uws.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Testing context-aware software systems: Unchain the context, set it free!

Santiago Matalonga
Facultad de Ingeniería
Universidad ORT Uruguay
Montevideo, Uruguay
smatalonga@uni.ort.edu.uy

Guilherme H. Travassos
Universidade Federal do Rio de Janeiro
PESC/COPPE
Rio de Janeiro, Brazil
ght@cos.ufrj.br

Abstract— Background: In the era of digitalization, context awareness has become more important to allow software systems adaptation. Therefore, the quality assurance of such systems must consider the variation of context. However, there is a lack of software quality technologies doing so, which can increase the risk of failure of these systems.

Objective/Aim: To evidence recent advances regarding the testing of context-aware software systems (CASS), focusing on the abstraction that context should freely vary during test execution as it does in production environments.

Method: Based on knowledge acquired with quasi-Systematic Literature Reviews, we evaluate current testing approaches and discuss benefits and limitations of applying the observation above about CASS.

Results: It was not possible to observe any software testing technology supporting the unconstrained variation of context during testing a CASS. The practitioners, meanwhile, can use three main evidence-based strategies to test CASS: to assure functional correctness before turning to testing context-aware requirements; to design test cases to target context variables and to take advantage of automatic testing tools as much as possible.

Conclusions: To allow context to vary during testing freely, seems plausible. It provides a new frame of thought to enable the design of novel technologies to improve our capacity of testing CASS.

CCS Concepts
Software and its engineering → Software creation and management → Software verification and validation

Keywords. Software Testing, Context-aware, Context-aware software testing.

Introduction
Context-aware software systems (CASS) are a subtype of ubiquitous systems. They use context information to aid users in achieving their tasks. This capacity of sensing the application context has found useful in several application domains. Platform providers like Google are taking advantage of the myriad of sensors available in smartphones to provide contextual APIs to software developers (https://developers.google.com/awareness/). In turn, application providers are using these devices in domains such as smart traffic routing like Waze, health care, athletes training, and performance enhancement, among others. The IoT and manufacturing industry are also adopting context-aware software systems to optimize production and reduce costs [1].

As context-aware applications become pervasive and more ambitious, so does the cost of their failures. Imagine the remote-monitoring health care system described in [2], dispatching an ambulance with a doctor for kilometers across India for mistakenly interpreting a push-up for fainting. Likewise, imagine a CASS designed for optimizing a horizontally-integrated supply chain as the one envisioned in [3]. In this system, the factory capacity and inventory are managed by software, signaling work orders downstream, misjudging any variable can result in costly inefficiencies either for excessive inventory or additional waiting times.

These examples show the importance of assuring the quality of CASS. In particular, in this work we are interested in the current practices and technologies available for testing CASS, which is particularly challenging. Traditionally, designing a suitable set of test cases to verify a software system requires that the test designer makes tradeoff decisions between the test suite comprehensiveness and its execution effort. In CASS, context becomes a factor and the test designer capacity to plan a possible test suit worsens.

The definition of context is elusive [4][5]. To the best of our knowledge, the most used context definition comes from Dey and Abo [6]: “Context is any information that can be used to characterize the situation of entities that are considered relevant to the interaction between user and application.” At design time, context can be abstracted to context variables and inputs from sensors will be regarded the representation of such variables. At run-time, context can take many forms, and even be represented by values not foreseen for the application [7], [8].

Current testing approaches for CASS primarily rely on two strategies. Either they exploit extensive computation resources to simulate possible contextual input, or they artificially constrain the context values to make testing decisions manageable. However, both strategies impose artificial limitations and threat the effectiveness of testing, since these conditions are not present in the intended production environment of context-aware software systems.

A recent proposal to guide the design and execution of CASS test cases observed that context should freely vary during test execution as it does in production environments [7]. Therefore, during the design and execution of test cases, the test designer should not treat context variables as regular test input ones. First, because test input variables are deterministic, that is, the test designer has control over the test input variables and can define their values at design time. Context variables, by their nature, are not deterministic.
Dourish [5] makes a case for a phenomenological view of context for context-aware software. The author argues about a dual nature of context. In this view, an application can be designed to be aware to a context variable (and to the changes in its values) - deterministic. Yet, the same software system can occasionally react to a context variable - phenomenological. We argue that this phenomenological view of context represents an adequate framework when dealing with testing CASS. For instance, a self-driving car can be designed to turn left upon detection of a road cross (deterministic). However, designing a test case for this scenario, can not account for all possible hurdles and obstacles that can appear at any crossroad - phenomenological.

Therefore, to set fixed testing values for context variables during testing, leads to artificial constraints, limiting testing samples and trials regarding the CASS under test. These artificial constraints are enacted by setting context variables to predefined values (for instance location [9]) or simulating those values through a software infrastructure [10]. Secondly, while test input variables impact only on the system under test, context variables have an impact both on the system under test and on the test oracle.

The CASS under test can adapt its behavior after sensing a variation in the context. Therefore, for testing purposes, the test oracle must also be aware of the variations of context to provide an appropriate expected result regarding the test case. Having performed two quasi-systematic literature reviews on different aspects of testing [7] [8], we claim that there is no available technology that can support the variation of context during the testing of CASS.

Therefore, this paper discusses a new perspective for dealing with the testing of CASS and present current recommendations for those practitioners looking to test this sort of software system by comparing and contrasting current approaches against our proposed abstraction (i.e. the free variation of context during the testing). Furthermore, practical and methodological limitations of the approach are presented, aiming at to motivate researchers to work with these software technological challenges.

This paper is organized as follows. First, we discuss practical and methodological motivations for our novel approach. In section – “A new abstraction for testing context-aware software systems” - we describe the elements of the new approach and how it address the methodological motivations. We then go on to discuss envisioned attributes and limitations of the approach. Finally, we close this paper with some conclusions and reflections.

Testing Context-Aware Software Systems without imposing artificial context constraints

The research results discussed here comes from the CNPq CAcTUS project. The goal of this project is to understand how software engineers are currently assuring the quality of CASS. The CAcTUS project follows an evidence-based approach to software research. The results of two quasi-systematic literature reviews (qSLR) [7], [8] support our propositions. This section will first describe the implications for the industry and then discuss the methodological motivations arising from the acquired evidence.

Practical contributions

Through our observations of current approaches used for testing CASS, we abstract the following practical contributions that the industry can currently apply when testing such type of software system[12]:

P1. Assure functional correctness before turning to testing context-aware requirements.

When the contextual input is considered for designing test cases, the test input space grows. Therefore, before tackling the problem of verifying how the software system behaves under a changing context, verify the system for its functionality in constrained contextual environments.

P2. Design test cases to target context variables.

Context variables are the designed parts of the software system where context is sensed. Therefore, the test designer should take care of such variables, as explicitly recommended in [13].

P3. Take advantage of automatic testing tools as much as possible.

The most popular approach to deal with the unforeseen variation of context is to use platforms that can simulate pseudo-random input for context variables (for instance [10], [14], [15]).

Practitioners can use these previous recommendations to test CASS because they do not require any special infrastructure (see in section “Practical limitations of the proposed approach”, PL3). The current software testing technologies can enact them.

Nevertheless, we argue that they do not capture the phenomenological nature of context. Therefore, these approaches increase the risk of not achieving appropriate coverage of the system under test.

Methodological motivations

This section argues for the development of a technology that enables the variation of context during testing.

M1. Context influences the test cases [8].

This influence is explicit in two test case elements: the test item and the test oracle. Since the Test Item is context-aware, when executing the test case, the Test Item can sense the context and react by adapting its behavior. The Test Oracle must match the Test Item behavior change to provide a suitable expectation of results to evaluate the Test Case.

M2. The application context, the test input, and the test environment are different entities and should also be distinguished during testing [7].

The ISO/IEC/IEEE 29119 Standard for Software and Systems testing [15] defines Test Input as: “the input to which the Test Item will be stimulated.” Moreover, the same standard defines the Test Environment as “the facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform the software testing.” Therefore, for a Test Case, both Test Inputs and Test Environment should be completely deterministic. The test designer should be able to define the test inputs (variable types and values), and the test environment (hardware, operating systems, database, etc.). In contrast, for context, only the variable type can be determined during the test planning. Pre-setting context variable values leads to fixating context, which leads to observing the test item in an artificial environment. Therefore, limiting the possibility of identifying context-related faults (which results in M3).

M3. Context must be left free to change during the test case execution [7].

In a production environment, a CASS is not constrained by the values that it can accept from its context sensors. For the execution of a Test Case, the CASS should not be imposed by artificial constraints that could limit its interaction with the context.
Back to the self-driving car example of the introduction, this car can be programmed to accept the full range of values included in the sensors specification. However, during testing, it is not feasible to design scenarios for all possible values. Therefore, higher abstraction oracles must be set.

M4. Fuzzy Test Oracles.

As a corollary of the previous two results, we argue that Fuzzy (non-deterministic) Test Oracles are needed to evaluate the test cases results. Since context has influence in the Test Case, then the Test Oracle cannot be deterministic. Therefore a behavior expectation should be sufficient to pass judgment on the suitability of the Test Item behavior modification given the current context and Test Inputs.

Figure 1 Abstraction of the elements of a test case allowing the variation of context

A new abstraction for testing context-aware software systems

Figure 1 depicts the relationships between the methodological motivations and the elements of a test case. In this figure, Context – represented by a cloud – is not completely within the scope of the Test Case. This relates how the test case cannot control all possible contextual inputs (M3).

The arrows for the context towards the system under test and the test oracle, convey the influence that the context has on both of these items of a test case (M1). As per this influence, the test oracle must adapt to the contextual changes to give timely and relevant expectations to evaluate the test output (M4).

Finally, the context, the test input, and the test environment are different entities that must be differentiated (M2). Test input and context are different in nature. The test input is deterministic. Moreover, though some inputs can be generated outside of the scope of the test case, these are known to the tester (hence, we represent it as an arrow not as a cloud). Also, the context and test environment since the latter one should be defined in the system under test requirements. And though some environments are not feasible to reproduce during test time, this brings on a different problem than that of the variation of context.

Limitations of the proposed approach

This section presents some limitations that must be considered when generalizing the arguments from the previous section. This section is also divided into the discussion of the current practical applications and methodological limitations.

Practical limitations of the proposed approach

PL1. The proposed abstraction is in its early stages, and it is not completely clear how it can contribute to improving the quality of CASS.

The proposed abstraction was introduced in [7]. In [17], the abstraction supported the proposal of a test suit design
process, which has only been evaluated \textit{in vitro} with students. There is no discussion about how this differentiation applies to the unit testing/code level.

**PL2.** There are no available testing strategies to support practitioners on designing test cases for CASS, which consider the behavior changes regarding the Test Items and Test Oracles.

In [17], the authors compare which test types (as defined by ISO/IEC 25010) are being used to design test cases for CASS. Their results show that the current solution is to adapt traditional test design strategies to overcome the context variation.

**PL3.** There is currently no computerized infrastructure to support the proposed approach.

To the best of our knowledge, no technical infrastructure enables the practical implementation of this abstraction yet. Even whether practitioners design test cases using the proposed abstraction, no tools nor approaches allow them to plan or execute CASS testing considering the variation of context during testing execution.

- **Methodological limitations of the proposed approach**

**ML1.** Reviewed literature is relatively narrow in comparison to the available technical literature.

Contributions presented in this paper all come from two independently conducted secondary studies. Both qSLRs share the same Population ("Sensibility to context," the same keywords in the search engines). Thus, we could crudely estimate the amount of missed primary sources. Table 1 presents the overlaps between both qSLRs. Using the Lincoln-Petersen estimator for Mark and Recapture method [18], we can estimate that either research did not consider five potential primary sources.

<table>
<thead>
<tr>
<th>Reference</th>
<th>In [8]</th>
<th>In [7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alisos and Dahl [19]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Amalfitano, D., et al. [13]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Canofra, G. et al. [20]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Chan W.K. et al. [22]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Griebe and Gruhn [24]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Merdes, M., et al. [26]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Micskei, Z. et al. [27]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Propp, S., Buchholz, G., and Forbrig, P. [28]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Ryan and Gonsalves [29]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Satoh, I [9]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Tse, H. et al. [30]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Wang, H. and Chan, W.K. [31]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Wang, H., Chan, W.K, and Tse, H [32]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Wang H., Zhai, K. and Tse, H [33]</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Ye, C. et al. [34]</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1 Evidence-based technologies regarding CASS testing**

**ML2.** Industrial experts have not evaluated the proposed abstraction:

The only evaluation of the test cases design process observing context variation was an \textit{in-vitro} observational study. Without proper industrial validation, it is not possible to warrant that this abstraction is anything else than a theoretical construct with a limited perspective of industrial use. However, the initial results did not suggest it.

**On the issue of Quality assurance of context-aware software system, Where do we stand?**

There is a need for developing and deploying CASS. Four out of the 12 worst "software glitches" in 2015 were caused by CASS (see http://www.computerworlduk.com/galleries/infrastructure/top-10-software-failures-of-2014-3599618/). The industry has very few alternatives for assuring the quality of such systems. Available software testing technologies rely heavily on simulation or extensive computation. We believe that this solution is not feasible for all parties involved with the construction of CASS.

New testing technologies dealing with the issues brought by the variation of context and its influence on systems' behaviors are needed. Such technologies need to be experimentally validated to provide feasible solutions for the industry. Since the demand for CASS is continuously growing while such technologies are being developed and validated, the current and viable alternatives for the industry are P1, P2, and P3, as discussed in Section II – \textit{practical contributions}.

**Summarizing remarks**

This paper describes the implications of a novel abstraction for testing CASS, where test input and context variables are differentiated. This proposal aims at dealing with the issue that context variation worsens the test designers' capacity to make trade-off decisions between test coverage and test effort. The authors argue that software testing technologies must evolve to address the context-awareness characteristic of software systems. Currently, the testing approaches deal with the variation of context by either fixing context values or simulating them based on computerized infrastructure. In either case, the test item is not receiving contextual information from the context, but through artificial methods. Thereby, limiting the test case validity. The authors advocate the idea that context should vary freely during test execution as it does in the production environments.

The proposed abstraction can guide the test designer into recognizing the different nature of both types of variables into the test case: deterministic test inputs under the test designer control and context variables, which should capture the possible variations of context. So far, the proposed abstraction has been enacted in a test suit design process, which has been subject to limited in-vitro evaluations.

Further research is needed to validate both the abstraction and the new software testing technologies that must be derived from it. We are currently working with industrial and academic partners to design an appropriate environment to support its experimental evaluation on testing CASS.

**Acknowledgement**

Authors would like to thank the CActUS team members for the thoughtful discussion of the ideas in this paper. The “CActUS - Context-Awareness Testing for Ubiquitous Systems” project partially financed by CNPq – Universal 14/2013 Number 484380/2013-3.

**References**

industrial context,” - Autom., vol. 64, no. 9, Jan. 2016.


