CATS design

Rodrigues, Felyppe; Matalonga, Santiago; Travassos, Guilherme Horta

Published in:
Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing - SAST

DOI:
10.1145/2993288.2993293

Published: 19/09/2016

Document Version
Peer reviewed version

CATS Design:
A Context-Aware Test Suite Design Process

Felyppe Rodrigues
Universidade Federal do Rio de Janeiro/COPPE/PESC
Rio de Janeiro, Brazil
felyppers@gmail.com

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

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

***Abstract* — Context-awareness is the ability of a system to gather information from its context and adapt its behaviors. This feature makes the testing of Context-aware software systems more challenging. Based on this assumption, this research proposes an approach to design functional test cases for context-aware software systems, the CATS Design. In its development, ideas from other domains presenting similar issues were tailored to the problem of testing context-aware software systems. This paper presents the design process of CATS Design, and its two empirical evaluation stages: 1) through a proof of concept, and 2) with an observational study involving eight undergraduate students in Uruguay. The results suggest that it is feasible to apply CATS Design for designing test cases for context-aware software systems.**

Keywords — Software Testing; Test Criteria; Context-Aware; Context-Awareness; Context-Aware Software Systems; CASS; Ubiquitous Systems.

I. INTRODUCTION

One of the most important aspects of ubiquitous systems is the possibility to interact with several actors simultaneously to support the user in the completion of its tasks in a non-intrusive way. These actors can be other users, other components or even other systems. Ubiquitous systems make use of sensors to gather the environment's data, which they use to adapt their behavior. This ability is called context-awareness [5]. Being able to interact with several different actors at once makes the quality assurance of such systems more difficult when compared with non-context-aware systems.

The technical literature presents several studies stating the difficulty to assure the quality of ubiquitous systems, as well as context-aware systems, and why they should be handled differently from other software systems [2, 11, 16]. These authors argue that these types of systems can lose efficacy and efficiency when treated as traditional software systems. We have undertaken a *quasi*-Systematic Literature Review [14], in which we observed that current test approaches handle the problem of context variation either by fixing context values at test time or relying on simulation to reveal failures [12]. Therefore, we argue for the need to develop test technologies that do not impose artificial constraints to the Test Item during the test execution.

This research aims at developing a specific way to assure the quality of context-aware software systems. Specifically, we have developed and empirically evaluated a process for designing test cases for context-aware software systems. We have named this process "Context-Aware Test Suite Design (CATS Design)". CATS enables the development of Test Suites including Test Cases which consider the variation of context during the test design and execution.

To develop the CATS Design process, we started with a process from the resilience practice in natural systems and interactively tailored it to Software. The CATS Design was evaluated in two steps. First, with a proof of concept, where we applied CATS Design to a context-aware requirements specification document. In this proof of concept, the feasibility of context-aware software systems to design test cases that account for context variation was successfully evaluated. Secondly, to validate the viability of applying CATS Design by other persons not involved in its design, we performed an empirical observation study involving eight undergraduate students. All subjects were able to design test cases considering context variation after following the CATS process. This paper presents the work we have undertaken for designing the CATS Design process, and the two evaluation activities previously mentioned.

Section II presents the research background and motivation. We discuss issues related to software testing, ubiquitous computing, and context-aware software systems. Section III present challenges related to Context-Aware Software Systems, followed by Section IV, which presents the CATS Design process. Section V presents both validations. In Section VI, we discuss limitations and threats to validity. Finally, in Section VII we present our conclusions and future work.

II. BACKGROUND AND MOTIVATION

A. The Testing Criteria Issue

Software testing is a practice used to verify that the software behaves as it was specified. The main concern is not to correct the defects, but to find them. Therefore, the activity

of testing aims to reveal failures and, afterward, find defects in the software.

The testing of all possible usage situations, distinct inputs, and all different environmental settings are not just a difficult task; it is unfeasible. This observation was made early in the evolution of the software development practices by Goodenough and Gerhart [7]. These authors started researching on software testing by raising the question: "what is a test criterion?" i.e., what constitutes a good decision, capable of assuring the greatest test coverage with minimum test effort.

The observation by Goodenough and Gerhart [7] has remained valid over the years. As we will discuss in the following section when the software is context-aware, the situation worsens.

B. Ubiquitous Computing

In 1991, Mark Weiser envisioned that "the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." [20]. Weiser believed that computers should not be limited to desktops or machines with direct user interaction. He defended the idea that a truly powerful technology should be able to support the user in its everyday activities, even without its perception. Weiser defined this concept as invisibility and this type of system as ubiquitous. Nowadays, ubiquitous systems have become a reality. Smart houses, GPS selecting a better path with less amount of traffic, mobile applications, wearable devices, digital cameras capable of focusing the image by themselves, are just a few examples [19].

To better understand the characteristics of ubiquitous systems, Spínola et al. [15] conducted a systematic review to find the features that characterize ubiquitous software systems. This work [15] identified ten characteristics that define a software system as ubiquitous. Spínola et al. claim that the scope of ubiquitous systems is related to the following features; however, the absence of some aspects does not imply that a system is not ubiquitous. The ubiquity in a system can be met wholly or partially according to the presence or absence of these aspects:

- Service omnipresence: it allows the users to move around with the sensation of carrying the computing services with them;
- Invisibility: the ability to be present in objects of daily use, weakening, from the user's point of view, the sensation of explicit use of a computer;
- Context sensitivity: ability to collect information from the environment where it is being used to improve the user's experience;
- Adaptable behavior: ability to, dynamically, adapt itself and offer services according to the environment where it is being used, respecting its limitations;
- Experience capture: ability to capture and register experiences for later use;
- Service discovery: pro-active discovery of services according to the environment where it is being used. The application has to interact with the environment and allow the user to do the same, to find new services or information to achieve some desired target;
- Function composition: the ability of (based on essential services) creating the services required by the user;
- Spontaneous interoperability: the ability to change partners during its operation and according to its movement;
- Heterogeneity of devices: it provides mobility among heterogeneous devices. That is, the application could migrate among devices and adjust itself to each of them;
- Fault tolerance: the ability to adapt itself when facing environment's faults (for example, on-line/off-line availability).

C. Context-Aware Software Systems (CASS)

According to the previous characteristics, context-aware software systems can be defined as a subtype of ubiquitous systems. Dey & Abowd [6] define context as "any information that can be used to characterize the situation of an entity". Where an entity can be a person, place, or object that is considered relevant to the interaction between an actor and the application, including the user and applications. In addition to this, a system is context-aware when it uses the context to provide relevant information and/or services to the user, where relevancy depends on the user's task and perspective.

Dey & Abowd [6] suggested organizing the context information into five dimensions, so answering the following questions the system can be aware of the context:

- **Who** – related to the actor identity. To adapt activities based on the presence of other people in the environment.
- **Where** – related to the actor's location and its impact on the user action.
- **What** – related to the identification of actor activities, embedding the interpretations of human activities to provide useful information.
- **When** – related to the temporal context, at which time or at which moment the action is happening and between which other activities.
- **Why** – related to the information that led to certain user actions. The challenge is to understand why the user is executing an action instead of just realizing what action the user is executing.

For instance, a mobile application can be seen as an example of a CASS. A context in which the user (**who**) is at the office (**where**) during a meeting (**what**) at the working

time (**when**) because it is a user's routine (**why**) can be recognized by the system.

The system then can gather the contextual data by the user profile, user calendar, clock time and other available resources. Based on the presented context, the system can turn on airplane mode to avoid unnecessary disturbance or detect that there is an incoming call and automatically avoid it.

III. Challenges in Testing Context-Aware Software Systems

In our previous work, we undertook a quasi-systematic literature review [14] to understand how context-aware software systems were being tested. Our results show that the challenges for testing these types of systems can be abstracted into the following categories [12]:

- Context Variation: This group of problems deals with issues associated with the alteration of one or more context variables, i.e. variables that can have their values updated without a direct user intervention. For instance, changes in location, in connected external devices, among others.
- Device hardware and network constraints: These challenges deal with the problem that hardware for context-aware are resource scarce.

Our research has not been able to find strategies to cope with the last challenge. In contrast, a few alternatives are available to deal with context variation:

- Identify context variables in the requirements specification to evaluate where the execution may lead to defects. The rationale behind this implies that context-aware software systems are more likely to fail where context changes [18]. Therefore, test case design can pay particular attention to these points in the software system.
- Use of dynamic test case generation and execution tools. The use of tools that automate the execution of the software system has also been established as useful. Even though these tools cannot assure conformance to specifications – since typically, they stimulate the software system to random inputs. These tools help by enhancing the robustness of the software systems. The works [3, 9, 10, 18] present few examples of these tools.

There is empirical validation as to the usefulness of these strategies to help with the quality assurance of context-aware software systems. Nonetheless, we argue that they do not address the whole problem of testing context-aware software systems. Because they provide artificial environments to execute the testing instead of a real scenario where the context can variate as in a production setting. In the following section, this argument is further developed.

IV. CATS Design: A Context-Aware Testing Approach

As discussed in the previous sections, the context can change before, during and after a user acting, and the system must be able to handle these variations. We call this context variation. During the execution of context-aware software systems in their intended environment, no restrictions can be imposed to the context variation. We argue that, during test execution, the test process must allow the context to change without restrictions as well.

Figure 1 presents how we conceptualize a test case that enacts the previously described abstraction. It shows how context affects both the Test Case and the Test Oracle. In the Test case, because context variables modify how the software behaves. In the Test Oracle, because, a modification in the value of a context variable may result in a different expectation for the output of the Test Case.

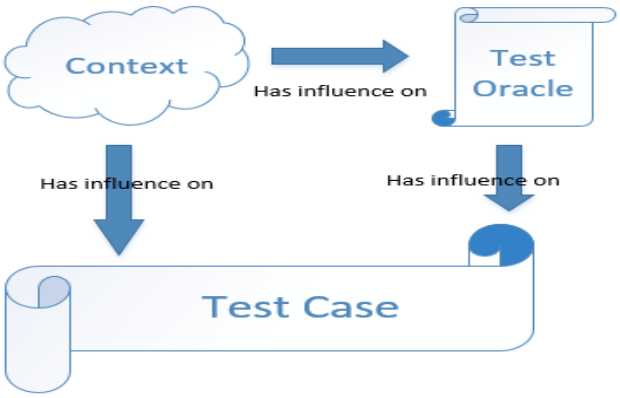

Figure 1 - Software testing execution process considering context.

As we have mentioned, we argue that the strategies presented in the previous section do not address this abstraction. Most of them rely on revealing errors in the execution, instead of checking the output against a Test Oracle.

We exemplify this abstraction with the Android OS integrated camera application. For instance, this application has the precondition of not executing with low battery (less than 10%) to save the remaining battery. Now, suppose the scenario where the user wants to take a picture of a specific moment, maybe during a lecture or a concert. To be sure that no time will be lost opening the application, the user lets the application running and wait for the perfect moment. When the application started, the device had more than 10% of battery left; however, the device is continuously consuming battery while the user waits for the specific moment to take the picture. When the moment comes, the device has now 8% left, so when the user tries to take the picture, the application freezes.

This type of failure during the context changing indicates that this variation was not considered while selecting the test coverage criteria, i.e. the perspective adopted is still to handle context variables as inputs. In fact, trying to execute the application with more than 10% of battery is possible and less is not feasible, but the transition remains an issue.

A. Designing CATS Design

The CATS design process is part of our ongoing effort to understand how to assure the quality of CASS systems. The Resilience domain presents similar issues when compared to the CASS domain. For instance, a resilient system can be defined as a system that can absorb disturbances and still retain its basic functions and structure [17]. It means that the system can receive several inputs and, being resilient, always return the expected output. CASS, must also be able to handle variation of special types of inputs (context variables) and still provide service to the actors. Table 1, extracted from [13], presents a comparison between the resilience practices and CASS.

Table 1 - Comparison of concepts between CASS and Organizational resilience

	Organizational Resilience	**Context-aware software systems**
Definition	"The ability of a system to absorb disturbances and still retain its basic function and structure." [17].	A system with the dynamic property of adapting its behavior according to the context (Adapted from CAcTUS Project).
Objective	Maintain the system general resilience (system identity).	Be able to adapt its behavior in order to keep the system working as expected based on the contextual variances (Adapted from CAcTUS Project).
How it calls the factors that affect the system?	Threshold	Context
Factor definition	"Disturbance capable of changing the system identity." [17].	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 its relevant relations for the actor-computer interaction (Adapted from CAcTUS Project).
How it anticipates these factors in order to achieve the proposed objective?	Thresholds avoidance techniques	Software testing and Verification and Validation techniques
What are the limitations of the way used to handle the factors?	The techniques help the thresholds identification, but not assure coverage. By knowing them, controller actions can be performed beforehand.	Lack of coverage, no evidence was found to support the existence of a technique to test CASS considering the context variance.

Although the idea of resilience provides a good level of maturity for a system, making a system resilient is not that simple. For example, every system can suffer disturbs. Disturbs that do not interfere with the system output, in the resilience domain, are simple disturbs. However, when such disturbs can change the expected output, we call them thresholds, i.e. a threshold is a disturbance capable of changing the system identity.

To keep the system working as expected, Walker and Salt [17] proposed a process for thresholds identification. If the system thresholds were known beforehand, it would be possible to take actions to avoid them. The proposed process can be observed below:

1. List the known thresholds: In the first step, the known thresholds are listed. They might be known for tacit knowledge, previous experience or just guessing.
2. Enumerate the thresholds of potential concern: Based on the thresholds found in the first step, look for thresholds that can affect the entire system directly.
3. Reproduce the system in a conceptual model: The third step creates a model to show how the system is supposed to behave, i.e. each possible system's state or usage situation is listed in separated boxes. These boxes are connected with two types of arrows, gray and black. The gray ones represent passive transitions, i.e. transitions happening without an actor intervention, as the battery consumption for instance. The black ones represent active actions, i.e. an actor intervention, like a user pressing a button. This model is very similar to State Machines usually used in Software Engineering [11].
4. Reproduce the system in an analytical model: The fourth and last step consists of listing every action the system can do and describe its impacts to the system, if possible with the aid of experts in the area. By analyzing these two generated mental models, it is possible to find unknown thresholds to be considered from now on.

In CASS domain, the thresholds can be seen as a context variation not expected by the system, or simply a context variation not considered during the test process. In this way, the process proposed by Walker and Salt [17] was adapted to the CASS domain.

B. The Context-Aware Test Suite Design Process – CATS Design

We set out to adapt the process by Walker and Salt [17] iteratively. Our vision was that, after each iteration, we could reflect on the process applicability to designing test cases, and also to see if the abstractions of thresholds and context variables were a good match for our domain.

A restriction, we had to circumvent, was the lack of available Software Requirement Specification Documents – except Castellanos [4] – therefore we created a few toy projects during the research process. Therefore, the three design iteration were carried out against play specifications we described to exemplify the concepts.

At each iteration, the lead author of this paper, generated test cases by following the current version of the process against the "Problem Set." This set of test cases was evaluated, as it was the process execution. A retrospective was then held among the researchers to discuss the lessons learned and to assess the iteration execution. Table 2 presents the three design iterations summarized and how they impacted in the final version of CATS Design process.

Table 2 - CATS Design development trials.

Iteration	Problem set	Lessons learned
1	Smart House example	- Propose to remove the concept of "threshold of potential concern". - Identification of Test Case template to document thresholds. - Group steps into phases.
2	Smart Camera example	- Remove context variables of potential concern. - Validation of proposed Test Case template
3		No further changes

The following numbered list describes the final phases and activities of the CATS Design process and implements the main lessons learned from the design process.

1. Context Variables Identification
   - Step 1: Identify context variables from the requirements
   - Step 2: Identify additional context variables
2. Thresholds Identification
   - Step 3: Create a conceptual model
   - Step 4: Find thresholds in the conceptual model
   - Step 5: Create an analytical model
   - Step 6: Find thresholds in the analytical model
3. Test Suite Generation
   - Step 7: List test oracles
   - Step 8: Create test cases
   - Step 9: Package the test suite

The process was separated into three phases: Context Variables Identification, Thresholds Identification, and Test Suite Generation. The first one finds the context variables in the requirements and by tacit knowledge. The second phase uses the requirements documentation together with the collected context variables to reproduce the system in mental models, allowing the thresholds discovery. Finally, the last phase uses the models to generate the test oracles and test cases. The process execution can be observed in the proof of concept presented in the next section.

V. VALIDATING CATS DESIGN

We performed a two-step validation of the CATS Design process. First, a proof of concept validation against a complete Software Requirement Specification (SRS) of a CASS. Then an observation experience involving undergraduate students using the same SRS. This section presents both validation steps.

A. Proof of Concept

The CATS Design process was initially evaluated through a proof of concept. The objective of this first validation was to execute CATS against a complete Software Requirement Specification Document. A bachelor's undergraduate project called CAUS (Context-Aware University System) [4] was selected for this proof of concept. The documentation for CAUS was created following the IEEE 830:98 Recommended Practice for Software Requirements Specifications [8].

The CAUS system aimed at applying the ubiquitous computing concepts into a university environment. By adapting the work of Abowd et al. [1], which had already proposed a context-aware university system, yet with no formal requirements specification. Based on this, Castellanos defined the requirements of CAUS, a mobile application that recovers information from QR Codes, Wi-Fi signal, and other sensors to provide contextual information to the user.

The CAUS seeks to provide information with minimal user intervention. The system supports the management of some aspects of interaction among students, staff and university entities, such as classrooms, offices and other users.

The system's communication approach uses the mobile device camera for QR code reading. Also, the system is supposed to be functional only inside the university dependencies, characterizing, even more, the concept of context-awareness. The features incorporated into the system include the localization of the users and offices, management

of user and offices, request for users and offices availability and messages exchange.

1) The Proof of Concept applied on CAUS

As presented in section IV, the first phase of CATS Design is to find relevant context variables for the system, i.e. the variables that have their values updated even without a direct request from the user, and once this value is updated, it can generate impacts on the system behavior. Therefore, Step 1 found context variables from the requirements documentation and Step 2 from tacit knowledge of the process applied.

Phase 1 - Context Variables Identification

- *Step 1: Identify context variables from the requirements*
  - University Wi-Fi
  - QR code (location/users and workshops info)
- *Step 2: Identify additional context variables*
  - Application Focus
  - Server Availability (offline/number of users)
  - Internet Connection
  - Information update

The requirements specification explicitly mentions the existence of the University WI-FI, which is the university local Wi-Fi connection on which the user must be connected to access the application. The QR Code is mentioned as well since all information that can be retrieved by the application relies on the QR Code scan.

The application focus was inferred since the CAUS is a mobile application and nowadays it is possible to run several mobile applications simultaneously and prepare the application to behave differently while running in the background. The server availability was inferred as well since no information about the server capacity over the number of users was available. Also, it is possible that the server goes offline, turning the QR Code functionality unavailable.

The internet connection was suggested because being Wi-Fi connected does not imply having an internet connection, and some of CAUS application's features require an internet connection. Finally, the information update was inferred due to the real-time system nature. A user can access an office status just before the office owner updates the status, making the user receive non-updated information. The user must be aware of the updates in real-time.

By using these six context variables, the thresholds identification phase started. Firstly, the conceptual model was created, and four thresholds identified by analyzing the model, as shown in steps 3 and 4.

Phase 2 - Thresholds Identification

- *Step 3: Create a conceptual model*
  - See Figure 2
- *Step 4: Find thresholds in the conceptual model*
  - Loss of University Wi-Fi Signal at any stage
  - Loss of Internet Connection during Play Store or Send Mail stages
  - Loss of Application Focus at any stage
  - Information Updates during information exhibition

The conceptual model (Figure 2) was significant to observe how and when the system sensed the context variables. However, it is worth noticing that using the models do not guarantee the complete coverage of the context-aware features. For instance, different testers might have different perceptions on the context variables.

- *Step 5: Create an analytical model*
  - Consult Table 3

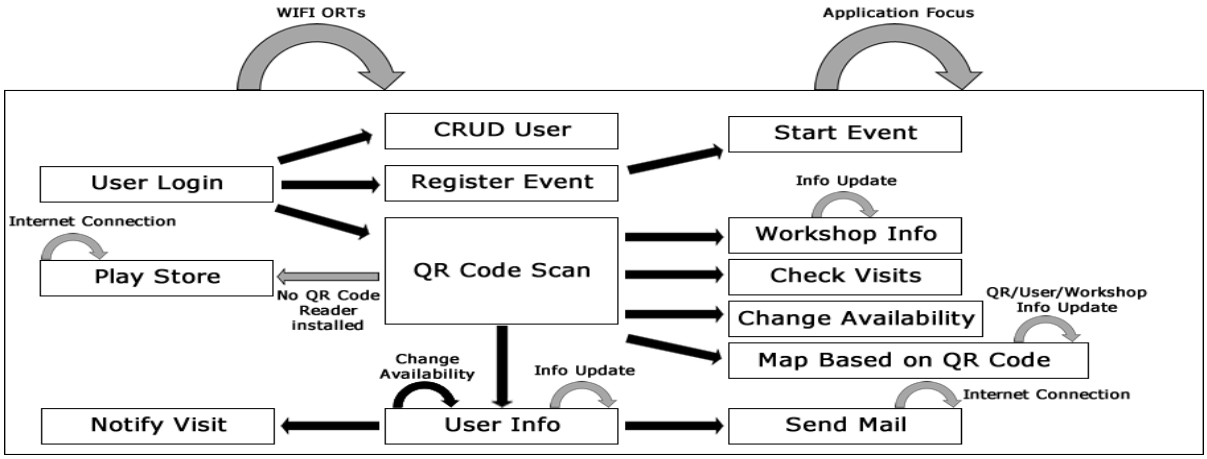

Figure 2 - CATS Design conceptual model applied to the CAUS.

Also, it was possible to observe that the University Wi-Fi signal, the internet connection, and the application focus are context variables that can change at any time, and the system must be ready to handle these variations. The information update might happen while some users are visiting the updated page, making the users lose the updates.

Since the information update is not a major context variance while the user is, for instance, sending an email, the identified thresholds were also described with the information of when they are important to be handled. Continuing with the thresholds identification phase, Steps 5 and 6 are responsible for the creation and interpretation of the analytical model. These steps allowed finding three additional thresholds to be incorporated into the test suite.

- *Step 6: Find thresholds in the analytical model*
  - Server goes offline
  - Server limit is exceeded
  - QR code is unreadable

By analyzing the context variables expected behaviors into the system on Table 3, it is possible to observe the issues concerned with the server availability.

Table 3 - CATS Design conceptual model applied to the CAUS.

Context Variable	Effect
University WI-FI	Grant the user access to the application.
QR Code	Return information of users and workshops.
Application Focus	If on focus, the user is interacting with the application. If not, the user is interacting with other application, but the one is still under execution in the background.
Server Availability	If the access to the database is lost or if the server gets too crowded, the needed information might not be available.
Internet Connection	Functions like the QR Code Reader Download and Send Mail require an internet connection.
Information Update	If the user is accessing some info which suffers an update in the meantime, the user must be notified.

Although the conceptual model describes how the states of the system, generated by changes in the context variables, interact with each other, the server availability feature is only possible to be observed by analyzing it separately. Also, the unreadability of the QR Code is only seen in this step. It is important to observe this to understand that the conceptual and analytical models are complementary.

Phase 3 – Test Suite Generation

Starting the third and last phase of the CATS Design process, the context variables and thresholds are already known. Also, the conceptual model, the analytical model, and the requirements specification are available artifacts. By analyzing the list of thresholds, the seventh step describes how the system is supposed to behave when each one of the thresholds is faced, i.e. the test oracles concerning the thresholds. The result of this analysis is that not all thresholds trigger functionality defined in the SRS. Therefore, these areas are documented to improve the requirement documentation. Also, as there is no expectation for the behavior, Test cases for this combination of inputs and context can not be defined.

- Step 7: List Test Oracles
  - See Table 4.
- Step 8: Create test cases

Once the test oracles are known, step 8 is intended to generate the test cases aiming at covering the identified contexts that are relevant to the system. Those contexts are the transitions considered in the conceptual model and the thresholds presented in the test oracles list. Each test case must describe the relevant context variables, the relevant thresholds and the test oracles for the listed thresholds.

Table 5 presents an example of generated test case. Remembering that if the SRS do not specify how to handle a threshold, the correct approach is to handle these faults first. Notice that no alternative flow has been described, for instance providing invalid credentials for the login feature. This kind of coverage can be obtained by other means of software testing, the focus of the CATS Design is the context variation, and this is why it is intended to be complementary to other testing techniques.

Table 4 - CAUS test oracles found with CATS Design

Feature	Context	Expected Output
Send mail	Loss of internet connection	Not specified
Play Store	Loss of internet connection	Not specified
QR Code Scan	QR Code unreadable	Cannot proceed transition
User Info, Workshop Info, Map based on QR Code	Information updates during the user access	Not specified
Any transition	Loss of University Wi-Fi connection	Cannot proceed transition
Any transition or state	Loss of focus	Proceed as if the focus was not lost
Any transition or state	Server goes offline	Not specified
Any transition or state	Server limit is exceeded	Not specified

Table 5 - Example of a context-aware test case generated by CATS design.

Test Case ID	**CAUS – TC001**
Test Objective	Verify the login feature
Precondition:	-
Test Input:	A valid user and password
Test steps:	1. User connects to University Wi-Fi 2. User executes the application 3. User provides credentials to access the application 4. The system shows the menu for the user
Relevant Context Variables:	1. University Wi-Fi 2. Application Focus 3. Server Availability
Known Thresholds:	a. Loss of University Wi-Fi connection b. Loss of focus c. Server goes offline d. Server limit is exceeded
Test Expected Outputs for each Threshold:	a. Notify the user and close the application b. Continue the application from where it stopped c. Not specified d. Not specified

- *Step 9: Package the test suite*

The package containing the test cases, the models, the test oracles, the context variables and the thresholds is called Context-Aware Test Suite and composes the step 9. The next section presents a discussion regarding the results of the CATS Design evaluation process through the proof of concept.

2) Discussion of results of the proof of concept

The first phase of the CATS Design, concerned with the context variables identification, indicated that the steps 1 (identifying context variables from the requirements) and 2 (identifying context variables using tacit knowledge) are complementary. Since there is no guarantee that the first step can recover all possible context variables, having complementary steps increase the strength of the context variables identification phase.

The second phase, responsible for generating the models, has indicated to be useful for thresholds identification. In this proof of concept, the models revealed to be complementary, when the conceptual and analytical models separately could not list the impact of all context variables found in phase one. However, using them together was possible to identify how each of them influences the system.

By the third and final phase, the process of creating test oracles based on the context variables, thresholds and requirements have revealed to be useful and helped to find faults from the requirements as well. It indicates that the CATS Design approach not only deals with the testing aspect, but also with verification of the requirements documentation. Although the aim of the test process is not to point omissions on the requirements specification, it is interesting to observe the verification ability of the CATS Design regarding the completeness of the requirements documentation for CASS.

The final version of the test case template can address scenarios without freezing the context, i.e. without treating the context variables as inputs. Instead, all possible relevant contextual changes are described together with the test scenario, and the Testers have the freedom to model the test environment as they wish and modify the context variables values at any time, knowing how the system is supposed to behave at any relevant configuration identified by the CATS Design process.

Considering the coverage, the test suite generated by the CATS Design process was able to cover factors that a non-context-aware test suite (Castellanos, 2015) did not cover, i.e. context variables changing in real-time, not being treated as simple inputs. The non-context-aware test cases are presented together with the requirements documentation online (Castellanos, 2015).

Nevertheless, the non-context-aware test suite covered factors that the CATS Design was not able to cover. These results indicate that the CATS Design process is complementary to the traditional test techniques, covering the context variance aspect only, and does what it was intended.

B. Validation through an empirical observation

To understand how CATS Design behaves and to observe if the practitioners other than ourselves could follow the process; we designed an observational study involving undergraduate students.

The selected students were enrolled in an undergraduate course in software testing. We intervened in the discipline by training the participants in ubiquitous computing, context-awareness, and CATS Design. Participants were then asked to apply CATS to the CAUS SRS. This section describes the results.

1) Preparation of the empirical observation

The aim of the second validation was to evaluate the feasibility that a third party – not related to the CATS Design authors – could find the process and the abstraction useful. This study was carried out in a software testing course in Universidad ORT Uruguay. Eight students were enrolled in this course and participated in the experience.

Since subjects enrolled in this course were expected to have experience in designing test cases, it was decided to train them in CASS and with the CATS Design process. Therefore, two three-hours training sessions were provided to the students.

In the first one, the students were exposed to the concepts of ubiquitous computing, context-awareness and the concepts involved in CATS Design (Thresholds and context variables). In the second one, the participants had to exercise CATS Design with the Smart Camera example.

Finally, the students were presented with the CAUS and were asked to design test cases for it. At the end of the session, the participants were presented with an answer sheet where they could leave their impression of the steps in the CATS Design process. For each process step, participants were asked to answer how valuable they found it on a scale of 1 to 5, with 5 being the highest value. All the answers provided along with a graphical analysis are presented below in Table 6 and Figure 3, and discussed in the next section.

Table 6 - Evaluation of CATS Design steps.

Subject / Step	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9
Participant 1	4	3	3	4	3	3	3	3	3
Participant 2	4	3	4	4	4	3	3	3	2
Participant 3	3	4	3	2	3	3	1	3	1
Participant 4	4	4	2	3	4	4	4	4	4
Participant 5	3	4	4	4	3	3	2	2	2
Participant 6	4	2	4	4	4	4	4	4	
Participant 7	4	4	3	4	2	2	2	3	1
Participant 8	3	1	4	3	3	3	4	3	4
Mean	4	3,5	3,5	4	3	3	3	3	2

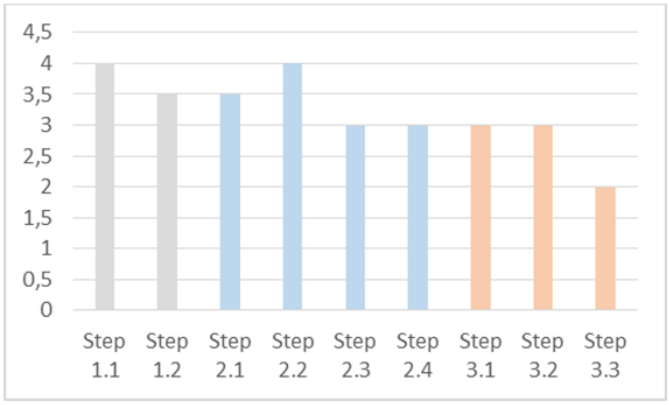

Figure 3 - Graphical analysis of CATS Design process.

2) Discussion of results of the experimental observation

It is interesting to analyze that the participants' opinion about the steps value decreases during the CATS Design process. For instance, none of the third phase steps, which were supposed to be the most important steps of the process, since they are the ones that provide the test oracles and test cases, received more than three for the rate.

Nevertheless, the steps related to the analysis of documentation and context variables/thresholds discovery received higher grades, indicating that the participants felt confident while providing information about context variables or thresholds, even though the grades of the analytical model seemed to be lower than the ones provided for the conceptual model.

It is interesting to observe that apparently the participants did not understand the pre-requisites among the phases and the dependency among them. Another possibility is that maybe the cognitive load of thinking regarding context variables and thresholds, made participants lose track of the main goal of the process that is to design test cases for a software application.

VI. Limitations and Threats to Validity

It is important to observe that even though the CATS Design process provides a useful improvement for CASS testing, due the lack of real projects, the presented results have some threats to validity. The final version of CATS Design was evaluated with a final undergraduate project chosen by convenience, not with industrial projects, a threat to conclusion validity.

The coverage provided by the process is limited by the context variables identified and the thresholds defined. If the models are wrongly built or if the requirements documentation lacks information, the process execution is compromised. Also, the coverage is totally based on the context-aware feature, making the process not confident as the only source of testing.

VII. Conclusions and Future Work

In this work, the concepts of ubiquitous computing and CASS are explored, as well as how these systems particularities can affect the software testing process. To handle these issues, a context-aware test suite design process was presented by adapting ideas from the natural resilience domain to build context-aware functional test cases, the CATS Design. Finally, the proposed testing process was evaluated through a proof of concept and then with an observational study involving eight undergraduate students at Universidad ORT Uruguay.

The proof of concept provides information that increases the belief in the applicability of the CATS Design approach in CASS settings. In addition to this, the generated models also provide a better understanding of the impact of context variation during the system execution.

The observational study indicated that CATS Design process could be executed by subjects other than the authors. It provided a first validation of the process in the field. However, the results show a decreasing level of understanding by the participants in the last steps of the process, probably indicating that the process might need more supporting material in its activities, to be more intuitive for novice testers. Even though, all participants were able to apply CATS Design entire process.

Several interesting observations can be drawn from these activities which might guide future research. For instance, during the step of test oracles identification based on the thresholds, it was observed that the CATS Design approach could be used not only for testing but also for verification of completeness of the requirements documentation regarding context-awareness. Therefore, if a test oracle identified by the process does not have an expected output, it is clear that this information is missing from the requirements.

Furthermore, on coverage, the level of coverage obtained by the CATS Design is based on the contexts revealed by the models generated, i.e. the coverage is concerned exclusively with the context-aware feature. In this way, our hypothesis is that CATS Design can provide a good coverage improvement regarding the context-aware feature. We argue that the CATS Design approach can be a complementary technique, which allows the test to deal with the context variance, which is a feature not handled in conventional software testing.

Finally, we are working to execute CASS in a live CASS project, since CATS Design is focused exclusively on the design of the test cases, but not with the test environment preparation and test execution yet.

Acknowledgment

This work was supported by “CAcTUS - Context-Aware Testing for Ubiquitous Systems” project partially financed by CNPq – Universal 14/2013 Number 484380/2013- 3. Prof. Travassos is a CNPq Researcher.

References

[1] Abowd, G.D. et al. 1996. Cyberguide : A Mobile Context-Aware Tour Guide. *Baltzer Journals*. 3, (1996), 1–21.

[2] Advisory, I.S.T. and Istag, G. 2005. Ambient Intelligence : from Vision to Reality. *Orientations*. (2005), 45–68.

[3] Amalfitano, D. et al. 2013. Considering Context Events in Event-Based Testing of Mobile Applications. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* (2013), 126–133.

[4] CAUS - Context Aware University System - SRS: 2013. *http://fi.ort.edu.uy/innovaportal/file/2231/9/esre_prototipo_caus.summary.pdf*. Accessed: 2016-06-27.

[5] Dey, A.K. 2001. Understanding and Using Context. *Personal and Ubiquitous Computing 5 (1)*. (2001), 4–7.

[6] Dey, A.K. and Abowd, G.D. 1999. Towards a Better Understanding of Context and Context-Awareness. *Computing Systems*. 40, (1999), 304–307.

[7] Goodenough, J.B. and Gerhart, S.L. 1975. Toward a theory of test data selection. *ACM SIGPLAN Notices*. 10, 6 (1975), 493–510.

[8] Ieee 1998. *IEEE Recommended Practice for Software Requirements Specifications*.

[9] Jiang, B. et al. 2007. MobileTest: A tool supporting automatic black box test for software on smart mobile devices. *Proceedings - International Conference on Software Engineering* (2007).

[10] Lu, H. et al. 2006. Testing context-aware middleware-centric programs. *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering - SIGSOFT '06/FSE-14* (New York, New York, USA, 2006), 242.

[11] Malik, N. et al. 2007. Future challenges in context-aware computing. *Proceedings of the IADIS International Conference*. (2007), 306–310.

[12] Matalonga, S. et al. 2015. Challenges in Testing Context Aware Software Systems. *Systematic and Automated Software Testing* (Bello Horizonte, 2015).

[13] Rodrigues, F. 2015. *CATS Design: A Context Aware Testing approach*. Universidade Federal do Rio de Janeiro.

[14] Rodrigues, F. et al. 2014. *Systematic literature review protocol: Investigating context aware software testing strategies*.

[15] Spínola, R.O. and Travassos, G.H. 2012. Towards a framework to characterize ubiquitous software projects. *Information and Software Technology*. 54, (2012), 759–785.

[16] Tang, L. et al. 2011. Supporting rapid design and evaluation of pervasive applications: Challenges and solutions. *Personal and Ubiquitous Computing* (2011), 253–269.

[17] Walker, B. and Salt, D. 2012. *Resilience Practice. Building Capacity to Absorb Disturbance and Maintain Function*.

[18] Wang, Z.W.Z. et al. 2007. Automated Generation of Context-Aware Tests. *29th International Conference on Software Engineering (ICSE'07)* (2007).

[19] Wei, J. 2014. How Wearables Intersect with the Cloud and the Internet of Things : Considerations for the developers of wearables. *Consumer Electronics Magazine, IEEE*. 3, 3 (Jul. 2014), 53–56.

[20] Weiser, M. 1991. The Computer for the 21st Century. *Scientific American*. 265, 3 (Sep. 1991), 94–104.