Towards Supporting the Specification of Context-Aware Software System Test Cases

Andréa Cristina de Souza Doreste and Guilherme Horta Travassos

Systems Engineering and Computer Science Program
COPPE
Federal University of Rio de Janeiro, Brazil
{doreste, ght}@cos.ufrj.br

Abstract. In Software Engineering, context can be understood as the overall set of information used to characterize the situation of an entity. A software system is context-aware if it uses the context to provide relevant information or services to the user. Nowadays, different types of software systems with a profound impact on the user's life can be considered context-aware (e.g., ubiquitous and Internet of Things software systems). Therefore, it is crucial to guarantee that they behave correctly. Software Testing is the primary process for verifying system behavior. Because of the particularities of Context-Aware Software Systems (CASS), conventional testing methods are not enough to test such systems. Based on that, we propose CATS#, a technique to support software engineers with the specification of CASS test cases and capture not only the context itself but its variation as well. CATS# is an evolution of the CATS (Context-Aware Test Suite) Design technique. It introduces two relevant evolutions: test case models and a new test template. The test case models were proposed after we searched for context-aware applications through the literature and realized that the context could affect the test process in different manners. The test template was proposed to capture the variation of context by specifying which variation should occur during the test case and when it should occur. We selected one example application to apply our technique, deriving different types of test cases and showing its relevance in these initial steps.

Keywords: Context-Aware Software System; CASS; Software Testing

1 Introduction

Over the last few years, many contemporary software systems having a profound impact on society are emerging. Such systems include solutions regarding Smart Cities, the Internet of Things, Cyber-physical System (CPS), Self-driving cars, and many other modern systems susceptible to the context and its variation. Moreover, like any other software system, they must adequately behave after their deployment. A common feature among such systems is their dependency on the variation of context. Some recently reported accidents with autonomous vehicles and jet airplanes have shown the damage is enormous when the variation of context cannot be appropriately verified.

However, it is crucial to validate the behavior of context-aware software systems (CASS) regarding the variation of context before their deployment.

Software Testing is the primary process for verifying the behavior of a software system. Due to the particularities of CASS, conventional testing methods are, usually, not sufficient. There is a lack of software technologies to test CASS [1]. Despite this fact, these systems still need to be adequately tested because, like any other system, they fail [2]. Therefore, these fails must be revealed, and the defects identified and fixed before the system reaches the final users. However, a lack of testing technologies for such software systems does not contribute to mitigating their failure after deployment. That is why it is essential to propose new software technologies to support testing strategies for CASS.

The CATS (Context-Aware Test Suite) Design technique [3] resulted from the CAcTUS (Context-Aware Testing for Ubiquitous System) Project, which investigated test strategies for ubiquitous systems. CATS has the purpose of identifying the context variables and thresholds (THR) to describe the test oracle and, consequently, the test cases. The test oracle would be the combination of context variables, identified thresholds, and expected behaviors once reaching the thresholds.

CATS Design was our first step towards understanding how to test CASS. It covers most of the issues regarding the specification of test cases for CASS. However, it does not include the perception that the variation of context can affect the test cases in different ways. Therefore, we propose CATS# as an evolution of CATS Design. Based on what we observed on working in the development of and testing with CASS, we introduce two new features: the test case models and a new test template. Section 3 depicts the reasons why these features contribute to evolving the CATS Design.

Besides this introduction, there are five more parts in this article. Section 2 has the definitions; Sections 3.1 and 3.2 present the modifications we are proposing; Section 3.3 shows one example application; Section 4 shows our final considerations and next steps.

2 Basic Definitions

2.1 Context-Aware Software Systems

In our work, context is the overall set of information used to characterize the situation of an entity considered relevant to the interaction between an actor, which can be a user, and an application [4]. Furthermore, while the context is, in some way, abstract, substantial, and continuous, we can discretize it through representative context variables (CVs). Each CV represents a piece of specific information about the context [3], as can be seen in Fig 1.

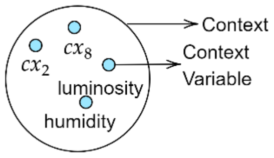


Fig 1. Context and Context Variables

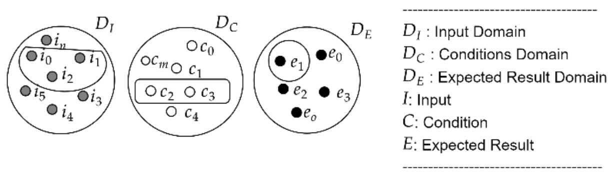
A software system is context-aware, whether it makes use of context variables to provide relevant information or services to their actors. For the sake of simplicity, there are two types of CASS. The first one (T1) only uses the information regarding the context to support its behaviors (e.g., a system responsible for sensing the temperature and display it to the user). The second one (T2) has its behavior influenced by the variation of context (e.g., the same previous system using the temperature to determine, automatically, whether turning the air-conditioner ON or OFF) [4].

Nevertheless, depending on the type of the software system, being context-aware is not all. Context is something dynamic; it can change anytime, influencing the T2 system´s behaviors on the fly. Thus, the CASS must be ready to respond and adapt itself appropriately to the variation of context. As so, their testing.

2.2 Software Testing

Software testing is responsible for verifying software system behaviors and revealing failures [3]. A Test Case (TC) is composed of input values (test input), constraints under which the test item must be executed (conditions), and the expected behaviors from the software under those constraints (expected results) (Fig. 2) [5].

Test Scripts would use the Test Cases during the Test Process for the testing of a Test Item into a predefined Test Environment to produce a set of Outputs. In the end, the obtained outputs (behaviors) should be compared with the expected results specified in the test cases (Oracle) to verify the success or failure [3]. Obtained and expected results are predictable and directly comparable in this case.



$$I = \{i_i : i_i \in D_I\},\ C = \{c_i : c_i \in D_C\},\ E = \{e_i : e_i \in D_E\}$$

$$TC = \{(I, C, E) : I \subset D_I,\ C \subset D_C,\ E \subset D_E\}$$

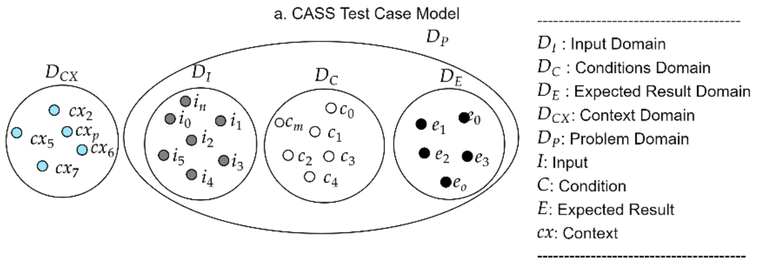
Fig 2. Conventional Test Model

3 Some Issues Regarding CASS Testing

The variation of context can affect the test process in two different ways: the input and/or the conditions of the test cases, which naturally affect the expected results. Consequently, sometimes, the variation of context will occur during the test execution by varying the environmental conditions. Therefore, to prepare the testing for this situation correctly, it is necessary an evolved test template. It should not just capture the variation of context (as it is in CATS) but also when such a variation should occur as well. It is not an easy task to abstract it in general scenarios, but we believe it can be done in particular cases. To support our discussions and represent these differences, we present a conceptual model in Section 3.1 and a test template in Section 3.2.

3.1 Testing Models

Section 2 illustrates a conceptual testing model for conventional software systems, but when the variation of context matters, a new set of information must be taken into consideration, as well as its influence in the test process. It has been said that the variation of context could affect the test inputs and conditions and, consequently, the expected results. Thus, when generating a model for testing CASS, it must be taken into consideration that each information on the test input and test conditions will represent and be attached to the context.
In this way, as Fig 3 shows, in CASS Testing Models, the input should be composed by a tuple (*i*, *cx*), where *i* is an element from the Input Domain, and *cx* is the context it represents. The same occurs with Test Conditions (*c*, *cx*) and with the Expected Results (*e*, *cx*) where *c* and *e* will be elements from the Conditions Domain and Expected Results Domain, respectively.


a. CASS Test Case Model
D_P
D_{CX}
D_I
D_C
D_E
cx_2 cx_5 cx_p cx_6 cx_7
i_n i_0 i_1 i_2 i_5 i_3 i_4
c_m c_0 c_1 c_2 c_3 c_4
e_1 e_0 e_2 e_3 e_o
D_I : Input Domain
D_C : Conditions Domain
D_E : Expected Result Domain
D_{CX}: Context Domain
D_P: Problem Domain
I: Input
C: Condition
E: Expected Result
cx: Context

$$TC = \{(I,\ C,\ E) : I \subset D_i \times D_{CX},\ C \subset D_C \times D_{CX},\ E \subset D_E \times D_{CX}\}$$

$$I = \{(i_i,\ cx_i) : i_i \in D_I,\ cx_i \in D_{CX}\},$$
$$C = \{(c_i, cx_i) : c_i \in D_C,\ cx_i \in D_{CX}\},$$
$$E = \{(e_i,\ cx_i) : e_i \in D_E,\ cx_i \in D_{CX}\}$$

b. No Context $I = \{i_i : i_i \in D_I\}$ $C = \{c_i : c_i \in D_C\}$ $E = \{e_i : e_i \in D_E\}$	c. Context influences the Input $I = \{(i_i,\ cx_i) : i_i \in D_I,\ cx_i \in D_{CX}\}$ $C = \{(c_i : c_i \in D_C\}$ $E = \{(e_i,\ cx_i) : e_i \in D_E,\ cx_i \in D_{CX}\}$
d. Context influences the Conditions $I = \{i_i : i_i \in D_I\}$ $C = \{(c_i, cx_i) : c_i \in D_C,\ cx_i \in D_{CX}\}$ $E = \{(e_i,\ cx_i) : e_i \in D_E,\ cx_i \in D_{CX}\}$	e. Context influences the Conditions and the Input $I = \{(i_i,\ cx_i) : i_i \in D_I,\ cx_i \in D_{CX}\},$ $C = \{(c_i, cx_i) : c_i \in D_C,\ cx_i \in D_{CX}\},$ $E = \{(e_i,\ cx_i) : e_i \in D_E,\ cx_i \in D_{CX}\}$

Fig 3. CASS General Test Model

It is important to note that the context element in E will always be the same one in I or C because a specific expected result would be a consequence of the context. Equations (1) and (2) demonstrate these relations:

$$\text{I} = \{i_0,\ i_1\},\ \text{C} = \{(c_2,\ cx_3),\ (c_4,\ cx_0)\},\ \text{E} = \{(e_0,\ cx_3),\ (e_2,\ cx_0)\} \quad (1)$$

$$\text{TC} = \{(\{i_0,\ i_1\},\ \{(c_2,\ cx_3),\ (c_4,\ cx_0)\},\ \{(e_0,\ cx_3),\ (e_2,\ cx_0)\})\} \quad (2)$$

Thus, this model can represent four situations, as summarized in Table 1:

- When there is no context (D_{CX} = { }), the model gets back to the Conventional Test Case Model (Fig. 3.b);
- When the context influences just the input, and the conditions remain constant (Fig. 3.c), the variation of context does not happen during the execution of testing. The only consequence will be increasing the number of test cases;
- When the context influences the conditions, and the inputs remain constant (Fig. 3.d), we need a new test strategy able to capture the variation of context during the test execution, and;
- When the context influences the input and the conditions simultaneously (Fig. 3.e), we have the complete and general CASS Test Case Model. In this case, there is a need to use test environments supporting the variation of context during the test execution. As far as we are aware, there is no such environment available yet [8].

Table 1. Test Situations

	Input (I)	**Condition (C)**	**Expected Result (E)**
Conventional Test Case Model	Static Value	Static Value	Static Value
CASS Test Case Model A	Dynamic Value	Static Value	Dynamic Value
CASS Test Case Model B	Static Value	Dynamic Value	Dynamic Value
CASS Test Case Model C	Dynamic Value	Dynamic Value	Dynamic Value

3.2 Testing Template

The first column of Table 2 shows the fields of CATS Design's test template. It has the necessary information used in conventional systems and new fields specific for CASS (*in Italic*). Its objective is to specify every known threshold for each context variable related to the Test Case. Thus, if one of the specified contexts varies during the test execution, the Expected Output should be specified as well.

We intend to cause a variation of context (regarding a specific CV) during the test case execution while keeping all the other CV fixed. In doing so, some fields must be changed, and others added to the CATS template. The result is presented in the second column of Table 2. The field "Fixed Conditions" specifies which CV is static; the "Varying Conditions" field contains how the conditions must variate, crossing the previously established threshold. Finally, the CATS# template gains a structured flow similar to use cases. The tag "ci" is added to the Test Steps, indicating that conditions ci must vary just before the execution of the next test step. A filled template can be seen in Table 5.

Table 2. CATS Template X CATS# Template

CATS Template	CATS# Template
Test Case ID	Test Case ID
Test Objective	Test Objective
Precondition	Precondition
X	*Fixed Conditions*
Test Input	Test Input
Test Steps	Test Steps
Relevant Context Variables	*Varying Conditions (C)*
Known Threshold	*X*
Expected Result for each Threshold	*Expected Result (E)*
Postconditions	Postconditions

3.3 Proof of Concept

Our proposal is at its initial stage. To observe its initial feasibility, we selected an application by Afanasov, Mottola, and Ghezzi [6] to use CATS#. It is a Wildlife tracker where battery-powered nodes from a WSN (wireless sensor network) are embedded in collars and attached to animals. Each node has a GPS sensor, two low-power short-range radios working as proximity sensors, and small solar panels to prolong the node lifetime. The GPS sensor captures the pace of the animal's movement, and it may be disabled if the battery is running low. A proximity sensor is responsible for tracking an animal's encounter with other animals and logging the data. Another sensor is responsible for sending the collected data to the base-station whenever they are near. More information about the application can be found in [6]. After applying CATS#, we obtained the results presented in Table 3.

Table 3. Results after applying CATS# for Wildlife Application

Context Variable	Effect	Threshold	Test Situations
Battery Level	> threshold, GPS status = ON	Battery Level going down the threshold	CASS - Model B
	< threshold, GPS status = OFF	Battery Level going up the threshold	CASS - Model B
Animal Proximity	= YES, collect data from encounters	Getting closer to an animal	CASS - Model A
Base-Station Proximity	= YES, send data to the Base-Station	Getting closer to a Base-Station	CASS - Model A

We selected three test cases, one from each model, to show how they will differ. As Tables 3 and 4 show, in TC01, the "battery level" will be tested as a condition (CASS

Model B) related with CX0 (Bat. Level ≥ *THR*) and CX1 (Bat. Level < *THR*), and the Expected Results are “GPS Available” and “GPS Disabled,” respectively.

In TC02, "BS Proximity" will be tested as an input (CASS Model A) and is related to CX0 (BS Proximity = NO) and CX3 (BS Proximity = YES). The Expected results related to CX3 is "Send Data to BS" and to CX0 is "Log in GPS information."

Finally, to show the difference, TC03 specifies a conventional test case, with no variation of context.

Table 4. Wildlife Test Cases

Id	Input	Context	Conditions	Expected Result
TC01	Animal location	CX0	Bat. Level≥ *THR*	GPS status = ON
		CX1	Bat. Level< *THR*	GPS status = OFF
TC02	BS Proximity = NO	CX0	Bat. Level ≥ *THR*	Log Data in the node
	BS Proximity = YES	CX3		Send Data to BS
TC03	Animal location	CX0	Bat. Level ≥ *THR*	Log in GPS information

As we said before, conventional strategies can be used to test the CASS Test Case Model A (see Table 1) and, consequently, TC02. Therefore, we selected TC01 to use our proposal test template (Table 5). It shows that, after the third step, the Battery Level should variate as indicated in c1, and the Expected Result for this variation is “The system disables the GPS.” The template also shows that the variables Animal and BS proximity should be kept constant. Finally, the “GPS status” in Precondition and Post-condition changes because of the variation of context occurring during the test execution.

Table 5. CATS# Template for a TC from Wildlife Application

Test Id	TC01
Test Objective	Verify the variation of the CV Bat. Level
Precondition	– The BS is out of reach – Battery-Level ≥ *THR* – Solar Panel is deactivated – GPS status = ON
Fixed Conditions	– Animal proximity = NO – BS proximity = NO
Input (I)	– GPS location (lat, long)
Test Steps	1. Starts the node 2. Change the node position 3. The node starts to collect data (c1) 4. Change the node position
Varying Conditions (C)	c1. Bat. Level ≥ *THR* → Bat. Level < *THR*
Expected Result (E)	The system disables the GPS
Post condition	– GPS status = OFF – Battery-Level < *THR*

4 Final Considerations

As can be seen in [1], there is a lack of technologies to test CASS. Also, we realized that the context had been treated as someway static most of the time. Besides, the different ways of how the context affects testing are not even noticed. Thus, it cannot be captured by the test activity. CATS# is an attempt to evolve CATS adopting the learning from the literature. We propose the CASS testing models and the new test template to help software engineers understand the variation of context and adequately plan the testing for CASS. In this way, each fulfilled test template will represent a different context with its expected result and the test script necessary for the future test execution. Also, the template can be used in different levels of testing, in simple scenarios (such as varying one CV) or more complex situations (where the CVs will interact).

The application used in section 3.3 is simple. However, it intends to demonstrate the different test situations involved in the testing of CASS, as well as the influence of the variation of context in the test process.

Finally, CATS# is ongoing research. We intend to modify the entire CATS process to make it simple and easily applicable. Besides, after all the modifications, experimental studies will be conducted to adequately evaluate the behavior of CATS#, specifying test cases for different test situations and applying it in more complex scenarios. We are aware that forcing a variation of context as specified is not always an easy task when performing testing. Thus, in the future, we intend to propose a tool to control the Test Environment and, consequently, setting the context-free during testing [8].

Acknowledgments

The CNPq (Brazilian National Council for Scientific and Technological Development) and CAPES (Brazilian Coordination for the Improvement of Higher Education Personnel) support this research. Prof. Travassos is a CNPq researcher (304234/2018-4).

References

1. Amalfitano, D., Matalonga, S., Doreste, A., Fasolino, A. R., Travassos, G. H.: A Rapid Review on Testing of Context-Aware Contemporary Software System, Technical Report, UFRJ, Rio de Janeiro (2019). http://www.cos.ufrj.br/uploadfile/publicacao/2910.pdf
2. Charette, R. N.: Why software fails. IEEE Spectrum, 42(9), 42-49 (2005).
3. Silva, F. R.: CATS Design: A Context-Aware Testing Approach, Master Dissertation, UFRJ, Rio de Janeiro, (2016). http://www.cos.ufrj.br/uploadfile/publicacao/2596.pdf
4. Dey, A. K., Abowd, G. D.: Towards a better understanding of context and context-awareness, Technical Report, Georgia Institute of Technology, Atlanta (1999), https://smartech.gatech.edu/handle/1853/3389
5. Dias-Neto, A. C.: Uma infra-estrutura computacional para apoiar o planejamento e controle de testes de software, Master Dissertation, UFRJ, Rio de Janeiro (2006). http://www.cos.ufrj.br/uploadfile/publicacao/2704.pdf
6. Afanasov, M., Mottola, L., Ghezzi, C.: Software adaptation in wireless sensor networks. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 12(4), 18 (2018).
7. Mirza, A. M., & Khan, M. N. A.: An Automated Functional Testing Framework for Context-aware Applications. IEEE Access, 6, 46568-46583 (2018).
8. Matalonga, S., Travassos, G.H.: Testing context-aware software systems: Unchain the context, set it free!, Proc. 31st CBSOFT/SBES. ACM (2017).