COPPE
UFRJ
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia

# CATS#: A TESTING TECHNIQUE TO SUPPORT THE SPECIFICATION OF TEST CASES FOR CONTEXT-AWARE SOFTWARE SYSTEMS

Andréa Cristina de Souza Doreste

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Guilherme Horta Travassos

Rio de Janeiro
Dezembro de 2021

CATS#: A TESTING TECHNIQUE TO SUPPORT THE SPECIFICATION OF TEST CASES FOR CONTEXT-AWARE SOFTWARE SYSTEMS

Andréa Cristina de Souza Doreste

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Guilherme Horta Travassos

Aprovada por: Dr. Guilherme Horta Travassos

Dr. Claudio Miceli de Farias

Dr. Santiago Matalonga Motta

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2021

Doreste, Andrea Cristina de Souza

CATS#: A Testing Technique to Support the Specification of Test Cases for Context-Aware Software Systems / Andrea Cristina de Souza Doreste. – Rio de Janeiro: UFRJ/COPPE, 2021.

IX, 107 p.: il.; 29,7 cm.

Orientador: Guilherme Horta Travassos

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 107-108

1. Software Engineering. 2. Software Testing. 3. Context-Aware Software System. I. Travassos, Guilherme Horta. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

*“E ali logo em frente, a esperar pela gente, o futuro está”*

(Aquarela – Vinicius de Moraes e Toquinho)

# Agradecimentos

Gostaria de agradecer a Deus por todas as oportunidades e bênçãos ao longo de todo o caminho.

Agradeço também a todas as pessoas que trabalharam, se esforçaram e cooperaram para que enfrentássemos a pandemia e chegássemos até aqui. Agradeço também a todas as pessoas que acreditaram na ciência e, tendo a opção, escolheram ficar em casa. A todos os cientistas brasileiros que continuaram fazendo ciência no meio do caos, minha eterna admiração.

Gostaria de agradecer a minha mãe, Márcia Souza, e ao meu tio, Nilton Ferreira, por todo apoio, suporte, compreensão e parceria. Eu não teria chegado até aqui se vocês não estivessem ao meu lado. Obrigada por apoiarem, desde o início, a minha decisão de fazer mestrado. Obrigada por tudo, especialmente por estarem sempre ao meu lado.

Agradeço também a toda a minha família e a todos os familiares que a vida me deu, que me apoiam e torcem tanto por cada passo que eu dou.

Obrigada aos meus amigos por tanto! A minha vida é muito melhor com vocês.

Em especial gostaria de agradecer a Débora Pina. Citando uma cantora que nós duas gostamos "long live all the mountains we moved, I had the time of my life fighting dragons with you". A gente escolheu segurar a mão uma da outra no dia em que viramos engenheiras, passamos por muitos momentos incríveis e estressantes juntas e eu espero que muitos outros ainda venham pela frente. Eu tenho muito orgulho de ser sua amiga, da mulher e cientista que você está se tornando e estarei aqui pra te aplaudir ou te acolher a cada passo.

A Juliano Marinho, obrigada por toda a paciência e compreensão que você teve comigo, como sua co-orientadora. Como sua amiga, obrigada por fazer parte da minha vida e por essa parceria tão incrível que a gente vem construindo ao longo dos anos. Ser sua co-orientadora de TCC foi um enorme presente pra mim e não tinha uma forma melhor de fechar o mestrado do que dividir esse caminho contigo.

Ao meu ex-colega de apartamento e eterno amigo, Brian Confessor, obrigada pela paciência, pela amizade e pelas diversas conversas interessantes e fora da caixa. Que você tenha muito sucesso na sua nova jornada e, se precisar de mim, estarei aqui (mas, por favor, se for pra revisar texto ou dar opinião como pesquisadora, inclua a tradução pro inglês ou português que japonês eu aida não aprendi).

Ao meu amigo, Marcos Filho, por sempre ter estado ao meu lado (mesmo quando estávamos geograficamente distante). Você, inúmeras vezes, trouxe significado e alegria pra minha vida quando eu não conseguia por conta própria e fazia o dia a dia parecer uma aventura. Foi um prazer dividir essa jornada contigo e, como eu já disse algumas vezes, espero que um dia você enxergue o potencial gigante que você tem e que alcance tudo que eu sei que você pode alcançar. Eu estarei sempre torcendo e vibrando por você!

Agradeço a minha psicóloga, Wanessa Lisbôa, por todo o apoio, paciência e competência. Obrigada por me ajudar a entender melhor meus próprios pensamentos e a lidar com a ansiedade no meio de uma pandemia que abalou o mundo.

Gostaria também de agradecer aos meus companheiros da linha ESE. Talita, Hélvio, Luciana, Taísa, Hilmer, Valéria e Danyllo, vocês são uma grande inspiração pra mim. Rebeca, eu espero que um dia, "quando eu crescer", que eu seja igual a você. Victor Vidigal, obrigada pelo suporte, parceria e oportunidades, eu aprendi muito contigo. Bruno e Alessandro, foi um enorme prazer dividir o mestrado, as disciplinas e o caminho com vocês, obrigada pela amizade, pelo companheirismo e parceria.

Agradeço aos professores Claudio Miceli e Santiago Matalonga por aceitarem o convite para participar da minha banca.

Por fim, gostaria de agradecer ao meu orientador, Guilherme Horta Travassos. Obrigada pelo carinho, pela compreensão, pelas broncas, pela confiança e por sempre me tornar uma versão melhor de mim mesma. Sob a sua orientação e tutela eu me tornei engenheira. Sob sua orientação, eu dou mais um passo no caminho de virar cientista. Por causa da sua orientação, eu me tornei (e me torno) uma profissional e um ser humano melhor a cada dia. Obrigada por cada voo que o senhor me permitiu dar, por cada conversa, cada conselho, cada acolhimento e por essa parceria que eu espero, honestamente, que não se encerre com o final do mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

# CATS#: UMA TÉCNICA DE TESTE PARA APOIAR A ESPECIFICAÇÃO DE CASOS DE TESTE PARA SISTEMAS DE SOFTWARE SENSÍVEIS AO CONTEXTO

Andréa Cristina de Souza Doreste

Dezembro/2021

Orientador: Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

Sistemas de Software Contemporâneos (CSS - Contemporary Software Systems) apresentam características distintas daquelas usualmente encontradas em Sistemas de Software Convencionais. Uma delas é a sensibilidade ao contexto, que é quando o contexto e sua variação afetam o comportamento do sistema de software de modo imprevisível e impensado. Dessa forma, é essencial garantir o correto funcionamento de Sistemas de Software Sensíveis ao Contexto (CASS - Context-Aware Software Systems). No entanto, percebe-se na literatura uma ausência de tecnologias e estratégias que apoiem o teste desse tipo de sistema. Com base nisso, esse trabalho apresenta uma técnica que visa apoiar a especificação de casos de teste para CASS chamada CATS#. CATS# evolui a técnica CATS (Context-Aware Test Suite) Design e apresenta um conceito adaptado de caso de teste que leva o contexto em consideração e oferece um template de teste que possibilita a captura (e representação) da variação do contexto durante a execução do caso de teste. A técnica CATS# foi aplicada em um projeto conduzido por estudantes de graduação e os resultados indicam sua viabilidade inicial.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

CATS#: A TESTING TECHNIQUE TO SUPPORT THE SPECIFICATION OF TEST CASES FOR CONTEXT-AWARE SOFTWARE SYSTEMS

Andréa Cristina de Souza Doreste

December/2021

Advisor: Guilherme Horta Travassos

Department: Computer Science and Systems Engineering

Contemporary Software Systems (CSS) bring distinctive characteristics compared to conventional systems to the table. One of them is context-awareness when the context and its variation affect the software system's behavior in unthinkable (sometimes unpredictable) ways. Therefore, it is essential to ensure the correct functioning of this type of system. However, as far as it could be investigated, there is a lack of software technologies to support these systems' testing. This work presents CATS#, a testing technique to support the specification of test cases for Context-Aware Software Systems (CASS). CATS# evolves the CATS (Context-Aware Test Suite) Design technique by adapting the test case concept to include the context and offers a test template capable of capturing (and representing) the variation of context that can influence the system's behavior during test execution. CATS# was applied in a project by undergraduate students. The results indicate its initial feasibility to support the specification of CASS test cases for situations not covered by conventional testing techniques.

**INDEX**

# 1 Introduction

## 1.1 Motivation

Nowadays, it is possible to observe the emergence of many Contemporary Software Systems (CSS), such as the Internet of Things (IoT), Cypher-Physical Systems (CPS), Smart Cities, Self-driving cars, and others. Most of these software systems interact closely with the real world, with animals, nature, and human beings.

Faqaha et al. [1] describe three problem domains where IoT is useful: nursing home patient monitoring, eating disorders, and in-door navigation systems for blind and visually impaired people. Martini et al. [2] describe technological resources for indoor agriculture, creating a Smart Farming application. Andrade et al. [3] show a Smart Research Building as an example of a smart environment, and Priyadarshini et al. [4] show the application of CPS in the healthcare industry.

These applications deal closely with lives. Therefore, software engineers must assure the quality of such software systems. However, compared with Conventional Software Systems, CSS has specific characteristics, such as autonomy, high connectivity, a deeper necessity of interoperability, and context-awareness, among others [5] [6].

Context-Awareness is the ability to sense the context in which the software system is immersed, taking advantage of it to provide relevant information or services to the actors that use it [7].

The context itself is abstract, infinite, and dynamic. It englobes all possible information about the software system, the hardware, the environment, and the users. Therefore, it can change at any time. Moreover, while entirely capturing the context is impossible, a failure may occur if a Context-Aware Software System (CASS) does not adapt its behavior when the context varies [8].

CASS failure may result in profound damage since these systems deal closely with the real world, as was mentioned before. Therefore, their adequate behavior must be assured [9].

In Software Engineering, there are different ways of verifying a system's quality. One of them is software testing. Software testing is the activity performed during (and after) the development cycle responsible for verifying whether a software system behaves adequately. The main objective of software testing is to reveal these failures, ideally, before they affect the users [10].

There are many techniques and strategies to test conventional software systems. However, these strategies are not able to reveal failures regarding context-awareness. It is because they do not consider the context or that it varies. Based on that, it is possible to conclude that new testing strategies should be proposed to test CASS, considering its context and variation [11].

The Context-Awareness Testing for Ubiquitous Systems (CAcTUS) project was started in 2015 to investigate this research topic. The main goal of the CAcTUS project was to understand and create strategies to test Ubiquitous Systems regarding the context-awareness property. As a result of the project, Silva [8] created a Context-Aware Test Suite (CATS) Design.

CATS Design searched the inspiration to create a testing strategy focused on the context and its dynamicity in different domains. It was the initial step towards understanding how to test CASS. However, since the knowledge about the context evolved, it was necessary to create a new technique.

In this work, a new testing technique for CASS is presented. While this technique was inspired by CATS Design, it evolves the conceptual background entirely, introduces new elements, and evolves its process.

This new technique will be presented through the next chapters and the entire research investigation and evaluation. This first chapter presents the problem being addressed, the methodology used to conduct this research, and how this dissertation is organized.

## 1.2 Problem and Objectives

As mentioned in the previous section, verifying the correct functioning of a context-aware software system is not a trivial task. These software systems can sense the context where they are immersed and adapt their behavior accordingly. Nevertheless, the context is abstract and dynamic. It cannot be entirely captured, and it can change at any time.

A testing strategy for CASS must capture the context and, particularly, its variation. Suppose the manner a software system behaves after being affected by the variation of context is not tested during the development phase. In that case, it is not possible to assure the system will respond accordingly. Thus, while a conventional software system failure is manageable, it can cause real damage in CASS, such as a car crash between autonomous vehicles or jet airplane accidents.

Conventional software testing strategies were not designed to capture the context. Therefore, new testing strategies must be proposed, investigated, and evaluated, focusing on testing the context-awareness property.

Based on that, an investigation was conducted to collect information about how the context affects the behavior of real-life applications. With this knowledge, it would be possible to propose a testing technique to support the specification of CASS test cases, which is the main goal of this research.

The following questions guided this work:

- How to test a Context-Aware Software System?
- Why is the testing of CASS different from Conventional Software Systems?
- How does the context influence test activity?

Besides proposing a testing strategy, the secondary objectives of this research were investigating how the context usually behaves and affects a software system.

While searching the literature, it was noticeable that different authors have different interpretations of the context. In this work, the context is mathematically represented, considering the inaccuracy regarding this concept in the literature.

Furthermore, using a universal language such as Math and its models may facilitate communication about this topic. Therefore, it was the chosen approach to conduct this research.

## 1.3 Methodology

The methodology presented in Figure 1, considering the research goal and the questions presented in the previous section, was followed during this work.

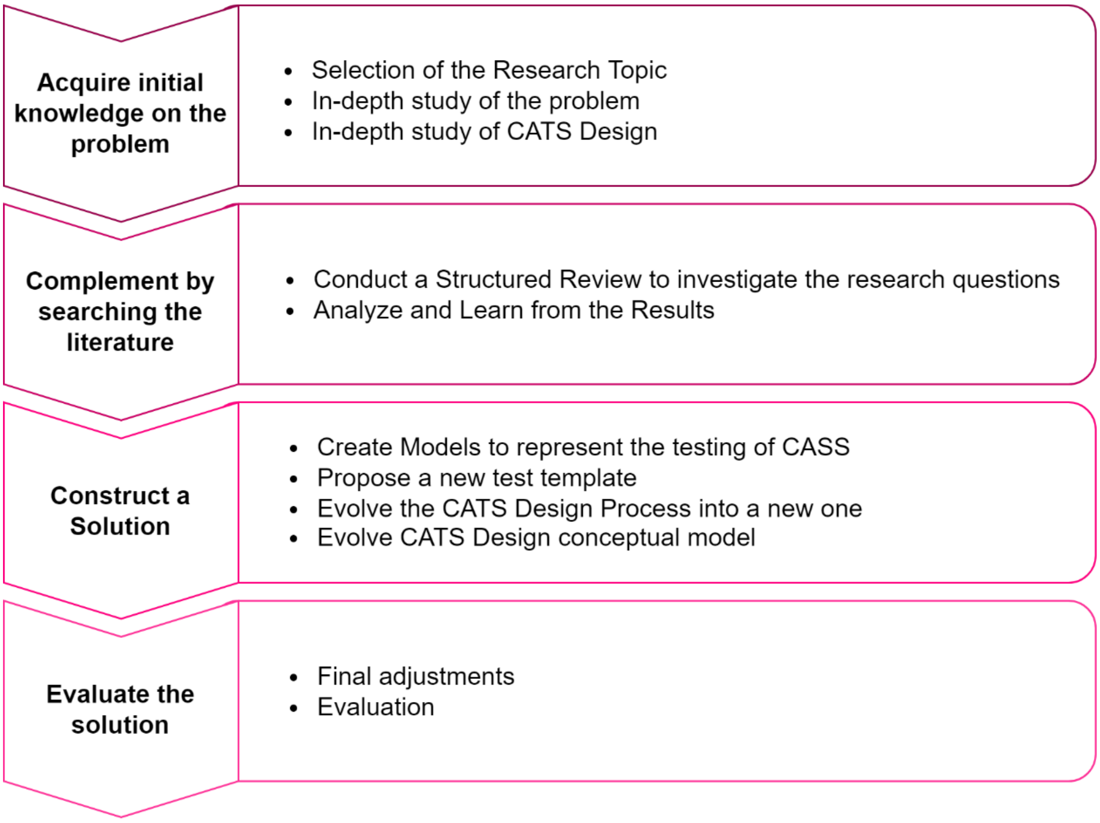

Acquire initial knowledge on the problem
Selection of the Research Topic
In-depth study of the problem
In-depth study of CATS Design
Complement by searching the literature
Conduct a Structured Review to investigate the research questions
Analyze and Learn from the Results
Construct a Solution
Create Models to represent the testing of CASS
Propose a new test template
Evolve the CATS Design Process into a new one
Evolve CATS Design conceptual model
Evaluate the solution
Final adjustments
Evaluation

*Figure 1. Methodology*

The research had the four main phases, explained below:

- **Acquire initial knowledge about the problem**: The research problem was defined in this phase. Additionally, the main concepts on which this research would be based were identified and studied. The CATS Design technique was also studied in detail. At this phase, the necessity of observing how the context behaves in real-life applications arose.

- **Complement by searching the literature**: At this phase, a strategy was defined to find context-aware applications in the literature. A structured review was conducted reusing the data from Amalfitano et al. [11]. The selection procedure had the following steps: inclusion based on title, inclusion based on abstract, and inclusion based on the application described in the articles. The lessons learned during this stage inspired the proposed solution.

- **Construct a solution**: After the results obtained in the previous phase, models representing how the context affects the testing activity were created. Based on these models, the construction of a solution began. The entire CATS Design process was tailored, evolved, and new elements were included to encapsulate the observations made at the previous phases. Finally, the solution was built using an iterative process composed of four steps: Learn, Build/Adapt, Measure, and Proofs of concepts.

- **Evaluate the Solution**: After three iterations and some adjustments, the proposed solution was evaluated.

## 1.4 Contributions

As direct results and contributions of this work, it is possible to cite:

- A discussion about the importance of considering the context while testing CASS;

- A discussion about what makes CASS testing different from Conventional Software Systems;

- Mathematical models to represent the context and how it influences the testing of CASS, and;

- A process for supporting the designing of CASS test cases.

As indirect results, it is possible to mention the following publications:

- **Doreste, A. C. S.**, Travassos, G. H. Towards Supporting the Specification of Context-Aware Software System Test Cases. In: XXIII Ibero-American Conference on Software Engineering (CIbSE), 2020, Curitiba.
- **Doreste, A. C. S.**, Amaral, I. D., Gonçalves, T. G., Travassos, G. H. Digitalizando o Microscópio Óptico: a solução do Parasite Watch. In: Anais do XIX Simpósio Brasileiro de Computação Aplicada à Saúde. SBC, 2019. p. 324-329.
- Amalfitano, D., Matalonga, S., **Doreste, A.**, Fasolino, A.R., Travassos, G.H. A Rapid Review on Testing of Context-Aware Contemporary Software Systems, 2019. https://www.cos.ufrj.br/uploadfile/publicacao/2910.pdf.
- Souza, B. D., **Doreste, A**., Xexéo, G., Reis, C. Utilizando o Framework MDA para Avaliar a Estética de um Jogo: Um Estudo Preliminar sobre a Percepção de Estudantes de Graduação. In: Anais do Simpósio Brasileiro Games (SBGames), 2018, Foz do Iguaçu

## 1.5 Dissertation Organization

This dissertation is organized into six chapters as follows:

- **Chapter 1. Introduction**: This first chapter introduces this work. It presents the motivation. The problem addressed the objective of this research and the methodology and the contributions.
- **Chapter 2. Concepts and Definitions**: This chapter presents the basic concepts used as foundations of this research, such as the definition of the context, the context-awareness property, and software testing.
- **Chapter 3. Testing CASS - A Structure Review:** This chapter presents the Structured Review to investigate how the context and its variation usually affect the context-aware applications and their results.
- **Chapter 4. CATS# - Towards Evolving CATS Design:** This chapter presents all the evolution from CATS Design until the final version of

CATS#. It presents in detail both CATS# versions 1 and 2 and their internal evaluation.

- **Chapter 5. CATS# - Final Version:** This chapter describes the final version of the CATS# technique and presents all the needed information for everyone interested in using it.
- **Chapter 6. Assessment Study:** This chapter presents the realized assessment study using an application named COVID Safe classroom. In this study, CATS# and CATS Design were used to create a test plan and compared to observe if there would be any advantages of using CATS#.
- **Chapter 7. Conclusion:** This chapter concludes this work. It presents the main contributions, limitations, and open items for future work.

# 2 Concepts and Definitions

## 2.1 Introduction

*“Humans are quite successful at conveying ideas to each other and reacting appropriately. This is due to many factors: the richness of the language they share, the common understanding of how the world works, and an implicit understanding of everyday situations. When humans talk with humans, they are able to use implicit situational information, or context, to increase the conversational bandwidth. Unfortunately, this ability to convey ideas does not transfer well to humans interacting with computers.”* (Dey and Abowd, 1999)

In 1999, Dew and Abowd wrote a paper explaining the importance and context-awareness property. According to them, humans are, naturally, context-aware, but not necessarily computers are. This statement can be applied to a classical computer machine from 1999. Since then, computers have evolved. Nowadays, an increasing number of software systems and applications are influenced by the context (even if they are not fully aware of it) [7].

Paradigms such as the Internet of Things (IoT), Cyber-Physical System (CPS), and Smart Cities, among others, deal with computational information and sensors, actuators, and hardware limitations. These systems use information captured from the “real world” to make decisions that can affect the users, their lives, and the environment they are immersed in [12].

Considering how much these context-aware systems can affect users' lives, it is primordial to ensure they behave correctly. One way of doing this is through software testing, conducted to evaluate whether a software system behaves as it should [13].

However, as was mentioned before, there are considerable differences between a classical computer machine from 1999 and the modern context-aware software systems. Therefore, it is important to investigate whether a test strategy applied for conventional software systems is suitable to evaluate CASS's correct behavior.

Due to all the previous issues, this chapter presents:

- The main concepts of software testing for conventional software systems
- The notions of context and context-aware software systems
- How the concepts of software testing can be adapted to CASS

These definitions were used as a foundation for this research. Understanding them is the first step towards understanding how to test CASS.

## 2.2 Software Testing for Conventional Systems

The ISO/IEC/IEEE 29119:2013 is a series of international standards for software testing. Their purpose is to support the software testing activities in different scenarios and provide a common vocabulary that can reference organizations worldwide.

According to the ISO/IEC/IEEE 29119:2013, software testing is a process conducted to evaluate the properties of one or more test items (product/functionality under test) with the following goals:

- Provide information about the quality of the test item
- Find defects before the test item reaches the production phase
- Mitigate the risks the test item with poor quality can present to the stakeholders

The test process generally applied for conventional software systems is presented in Figure 2. A test item is performed according to a test script, considering a predefined Test Environment. A test script is a procedure that must be followed during manual or automated testing. The test environment is the facilities, hardware, software, firmware, procedures, and documentation used to perform the test [13].

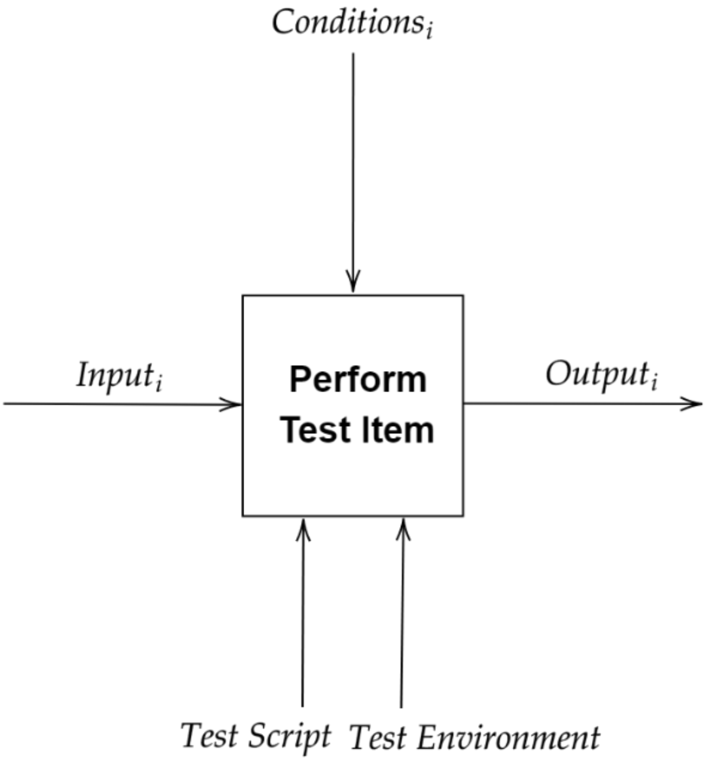
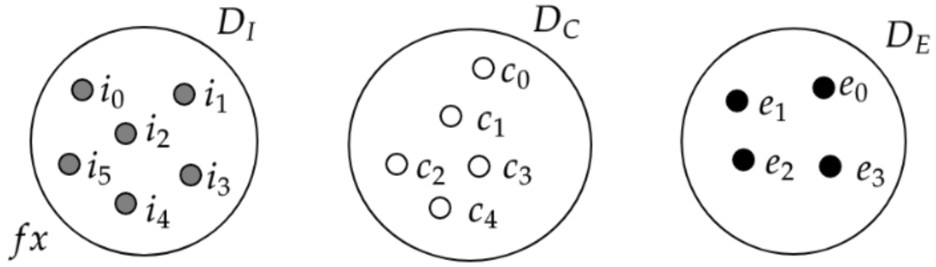


*Figure 2. Conventional Test Process*

During the process, a set of input values (test input) stimulates the test item under very specific environmental configurations (test conditions), producing a set of behaviors as the response (test output).

After executing this process, the test result will indicate whether the process failed or passed. Therefore, the test passes when the test output is like the set of expected behaviors from the software system (expected results). When the test output is different from the expected results, the test fails. In other words, there is a failure when the system does not behave as expected. If the test fails, an investigation must occur in the software system to determine the cause of that failure.

The combination of test input, test conditions, and expected results defines a Test Case (TC) presented in Figure 3 [14]. The test case is the most basic element of a test process. Therefore, its input and conditions should be chosen considering the test item that must be stimulated.



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

$$CT = \{(I, C, E) : I \subset D_i,\ C \subset D_C,\ E \subset D_E\}$$

| | |
|---|---|
| $D_I$ : Input Domain | $I$ : Input |
| $D_C$ : Conditions Domain | $C$ : Condition |
| $D_E$ : Expected Result Domain | $E$ : Expected Result |

*Figure 3. Conventional Test Case Model*

As was mentioned before, one of the primary goals of testing is revealing defects. However, the test activity shows the presence of defects, not their absence. Therefore, it is not possible to affirm that the software is defect-free, even if the test process does not reveal any defect [10]. The best approach, in this case, is to change the test strategy since the one used does not appear to be detecting them.

While many software systems share a certain amount of characteristics, they will differ in many aspects. Software testing must consider this. In addition, the same testing technique can work for a specific type of software system and not reveal failures when applied to a different one. Therefore, different testing approaches (test strategies) should be applied to different software systems.

For this reason, it is necessary to investigate whether software testing strategies successfully applied in a conventional software system can be used for testing CASS. Following this path, the next step is understanding how the context behaves, the software system's effects, and CASS's basic characteristics. The next section will explore these questions.

## 2.3 Context-Aware Software System

### *2.3.1 Context*

According to the quote that opens this chapter, context is informally described as "implicit situation information" such as "the common understanding of how the world works" or "implicit understanding of everyday situations." Although it gives an idea about the context, the term "implicit information" is too vague, especially when considering software systems. Thus, a more precise and formal definition is necessary. This work uses an adaption version of the formal definition proposed by Dew and Abowd in 1999 [7]:

*"Context is the overall set of information used to characterize the situation of an entity. An entity can be a person, a place, an application, a thing (in case of IoT), or any other type of logical or physical objects, including the system itself."*

The context itself is as infinite and abstract as the implicit knowledge of how the world works. However, in everyday situations, a specific set of information helps humans deal with a specific problem in a specific moment: look for traffic information to choose the better route to work, look for the forecast to decide whether to take an umbrella and so on. Information that does not have much value in some situations can be extremely important in others. The same occurs with software systems.

In this way, although the context cannot be entirely captured, it is possible to capture specific pieces of information, named Context-Variable (CV). Additionally, according to the above-presented definition, the “set of information” will characterize the situation of an entity. Therefore, a finite set of Context-Variable will characterize a Context-Situation, as shown in Figure 4.

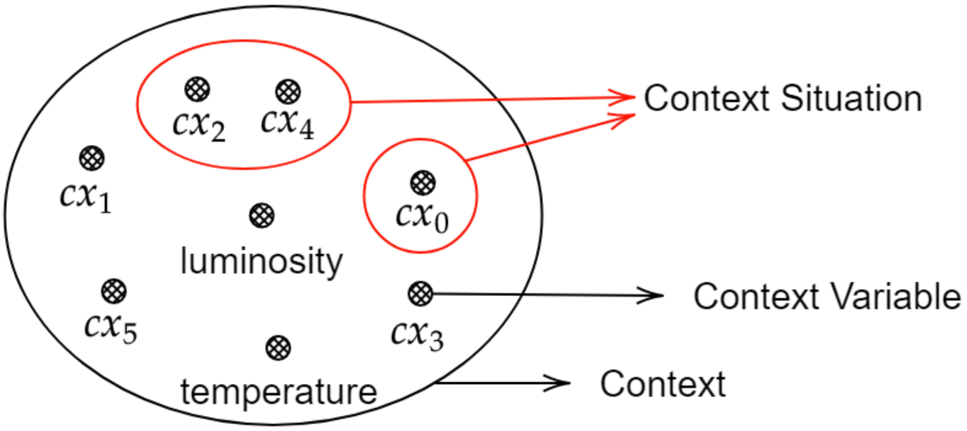


*Figure 4. Context Representation*

Another relevant characteristic of the context is its dynamicity. The context is always changing, always varying. It is possible to observe in everyday situations: a sunny sky can become cloudy; a warm day can become cold and rainy. The traffic on an avenue can become intense after an accident. All these changes can occur at any time. Therefore, a context variation can occur at any time and should be considered while designing solutions for engineering such software systems.

### *2.3.2 Context-Awareness*

Context-Awareness is the software system's property of capturing the context variables to provide relevant information or services to the different actors interacting with the system [15]. A system with this property is called Context-Aware Software System (CASS).

As far as it was investigated, there are at least two types of CASS:

- **Type 1 (T1):** Captures the context to make the context information available for their users or actors but does not have its behavior affected by the context.
- **Type 2 (T2):** Captures the context and uses the context information to decide. In this case, the software system has its behavior affected by the context

An example of Type 1 is a software system that collects the temperature of a room to display it to the users through a dashboard. An example of Type 2 is the previous software system that automatically uses the temperature to determine whether to turn the air-conditioner ON or OFF.

Types 1 and 2 are not mutually exclusive. A software system can be partially T1 and partially T2 (such as a software system that displays the temperature in a dashboard and uses it to turn the air-conditioner ON or OFF). The main difference is that T1 uses context information as regular input. In contrast, in T2, the context information can affect the whole functionality of the system. T1 behaves similarly to conventional systems. T2 must respond according to the context.

As was mentioned in section 2.2.1., the context can vary at any time. Therefore, T1 will not be affected. However, T2 must be prepared to deal with the variation of context. In other words, when the context changes, T2 must adapt and keep its behaviors consistent with the context.

While there are many techniques to help construct conventional software systems that can be successfully applied in T1, there is a lack of software technologies and techniques to help the engineering and verification of T2 [11].

T2 Software systems must be specified, designed, and verified considering the context and variation. However, the conventional practices used to engineer the software systems do not consider the context [11].

Based on this lack of technologies, a software testing technique (which considers T2 systems) is presented in this work. The next session explains why CASS testing (especially T2) must be different from conventional test strategies.

## 2.4 Software Testing for CASS

### *2.4.1 Literature Review*

After understanding the basic concepts related to the Software Testing and Context-Awareness area, it is necessary to look into the literature to understand how researchers are investigating the issue of testing CASS and how far they have gone with their investigation. The following paragraphs illustrate what we found out.

In a study published in 2017, Matalonga et al. [16] conducted a quasi-Systematic Literature Review (qSLR) to investigate the existent methods used to test CASS and how efficient they were. They used the same definition as us, proposed by Dey and Abowd [ref], and focused on Ubiquitous Systems. They selected 12 technical articles and analyzed them. As a result, they concluded that the existent test methods for CASS were not completely context-aware since they were based on selecting a specific context variable (location, for example) and assigning a fixed value for it, without any context variation during the testing.

A few years later, Luo et al. [17] surveyed to collect context simulation methods for testing mobile context-aware applications. The authors argued that conducting real-world tests for mobile context-aware applications can be laborious and time-consuming. Considering this scenario, using simulated context data to test the applications would be an alternative. The authors presented a comparison of the most relevant context simulation techniques. They concluded their work arguing that more research is needed to support the testing of mobile context-aware applications.

Continuing with the mobile application domain, Almeida et al. [18] performed a systematic mapping to identify the Android testing tools and the Android context-aware testing tools in the technical literature. As a result, they identified 80 general Android testing tools and 10 Android context-aware testing tools: five specifically to test context-awareness and five with a generic approach that includes context-awareness testing.

Matalonga et al. [19] executed a Rapid Review focusing on Context-Aware Contemporary Software Systems (CACSS) outside the mobile domain. The authors define CACSS as Contemporary Software Systems (such as Ubiquitous Systems, Internet of Things, Industry 4.0, among others) that are also context-aware and argue that they are mainstream in our society. The Rapid Review had the goal of investigating how the industry was dealing with the testing of CACSS regarding the context variation. However, the results indicate that even if it is possible to find some techniques and strategies to test CACSS, they are mostly focused on improving the test suite. Therefore, the authors could not find studies regarding testing techniques and strategies managing the context variation during the testing lifecycle process.

Lastly, Siqueira et al. [20] performed a Systematic Literature Review (SLR) and a thematic analysis of studies to characterize state of the art in Adaptive and Context-Aware Systems. Hence, they selected 102 studies and concluded that while there are some trends (such as model-based testing and hybrid techniques), some issues like uncertainty and prediction of changes are little investigated).

### 2.4.2 Discussion

Section 2.2 presented how software testing occurs in conventional software systems. However, when it comes to CASS, the context should be considered. When a software system is in the production phase, dealing with the real world, the variation of context happens at any time, and the system should adapt. To observe the same behavior during the test phase, the variation of context must affect the test process during the test execution, as Figure 5 presents.

A context variation takes the software system from one situation (situation 1) to the other (situation 2). For example, suppose the variation of context occurs after the ending of the test process. In that case, just the first situation (situation 1) is evaluated by the test. On the other hand, if the variation of context occurs before the beginning of the process, just the second situation (situation 2) is evaluated.

The test process must capture the software system response to the variation of context, as would happen in real life [9]. Therefore, the context must vary during the test process execution.

Capturing the context during the test execution is not an easy task. The test process itself was not designed for that. Nothing happens once the input is submitted, and nothing changes until the output is generated. Therefore, the conventional test process can be considered static, and a more dynamic test process is necessary to test CASS.

However, it is possible to observe from section 2.4.1 that, when looking into the literature, there is a gap of testing techniques helping the software engineer to capture the context during the test execution.

Therefore, new test strategies with a more dynamic approach should be proposed.

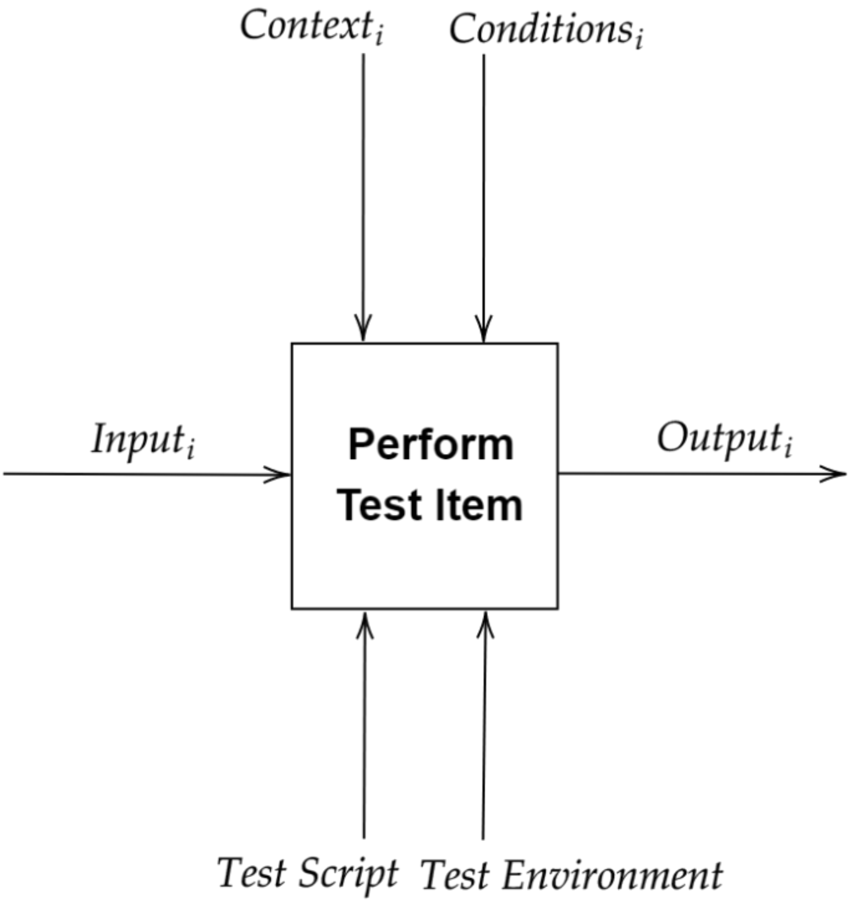


*Figure 5. CASS Test Process*

## 2.5 CATS Design

As mentioned in Chapter 1, CATS Design was created during the CAcTUS project and designed a testing technique for Ubiquitous software systems that focuses on the context-awareness property presented by this type of software system. To do this, the author got inspiration in problems with similar characteristics from domains such as Cybernetics and Organizational Resilience [8].

Although these domains did not have the concept of context, they share similarities. In cybernetics, for example, the goal is to control a complex system autonomously. If a disturbance occurs, the system should respond adequately without impacting its behavior. Moreover, the Organizational Resilience domain states that a system should be resilient. It must have many different manners to handle a disturbance to decrease the likelihood of failure [8].

Inspired by how these domains dealt with the necessity of adapting according to the changes, a process was created and refined. It is presented in Figure 6. It is centered on constructing a test oracle based on three elements: context variable, threshold, and expected result.

As mentioned in section 2.3.1, a context variable represents specific information about the context, such as location, temperature, or network availability. The threshold is the value assumed by the Context Variable, representing the variation from one context to another. The expected result is the behavior expected from the system once the threshold is reached.

The context variable is the temperature, using the situation mentioned in section 2.3.2 as an example; the threshold reaches a previously defined value. The expected result is the software system turning ON the air-conditioner.

With these three elements, the test oracle will be part of a test case. The process, shown in Figure 6, will guide the software engineers through the activities to create these test cases.

The CATS Design process comprises three main phases: Identify the Context Variables, Identify the Threshold, Generate the Test Suite. The following section will expose the activities that must be performed in each of them.

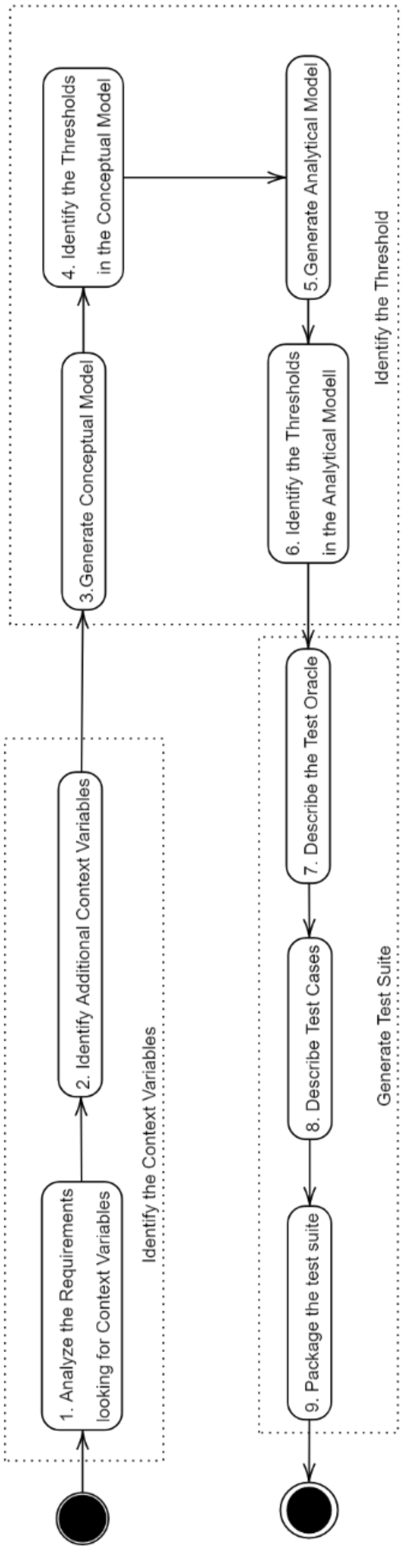

1. Analyze the Requirements looking for Context Variables
2. Identify Additional Context Variables
Identify the Context Variables
3.Generate Conceptual Model
4. Identify the Thresholds in the Conceptual Model
5.Generate Analytical Model
6. Identify the Thresholds in the Analytical Modell
Identify the Threshold
7. Describe the Test Oracle
8. Describe Test Cases
9. Package the test suite
Generate Test Suite

*Figure 6. CATS Design process*

### *2.5.1 Identify the Context Variables*

This phase aims to identify the Context Variables based on the requirements and the software engineer's knowledge about the domain. It comprises two activities: Analyze the Requirements looking for Context Variables and Identify Additional Context Variables. The output of each activity is a list of variables of context, which can be merged in one single list of every identified CV.

### *2.5.2 Identify the Thresholds*

Once the context variables are identified, the goal of this phase is to identify their thresholds. Therefore, this phase is composed of four activities: Generate Conceptual Model, Identify the Thresholds in the Conceptual Model, Generate Analytical Model, and Identify the Thresholds in the Analytical Model to help the software engineers with such an identification.

A Conceptual Model based on the software system behavior will be the output of the first activity. This model will be composed of boxes and arrows, as it is possible to observe in Figure 7. Boxes will represent the system’s states or usage situations, while arrows will represent transitions, both passive and caused by an actor intervention.

Figure 8 shows an example of a Conceptual Model for the Air-Conditioner situation from section 2.3.2. Two usage situations can be observed: air conditioner turned ON or OFF. Both passive situations (like the temperature reaching a THR) or usage intervention (the user manually turning the AC ON or OFF) could cause the system to vary from one situation to another.

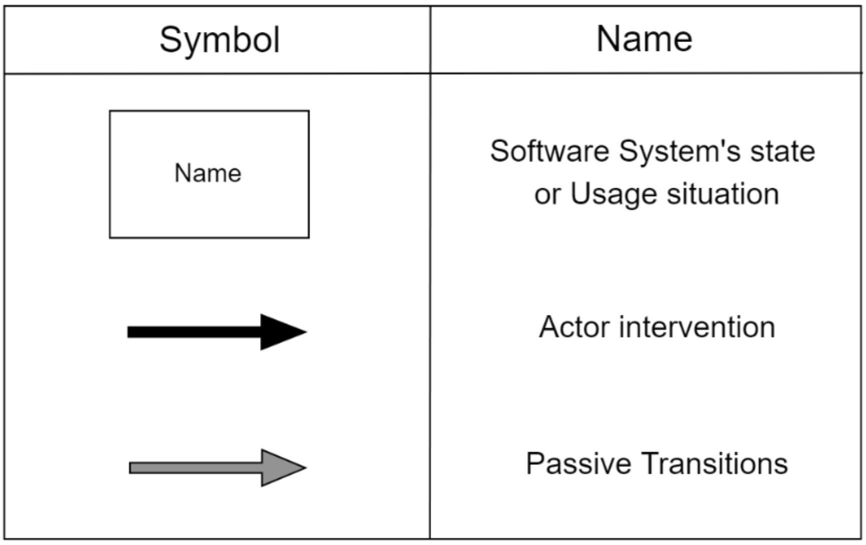


*Figure 7. CATS Design Conceptual Model Elements*

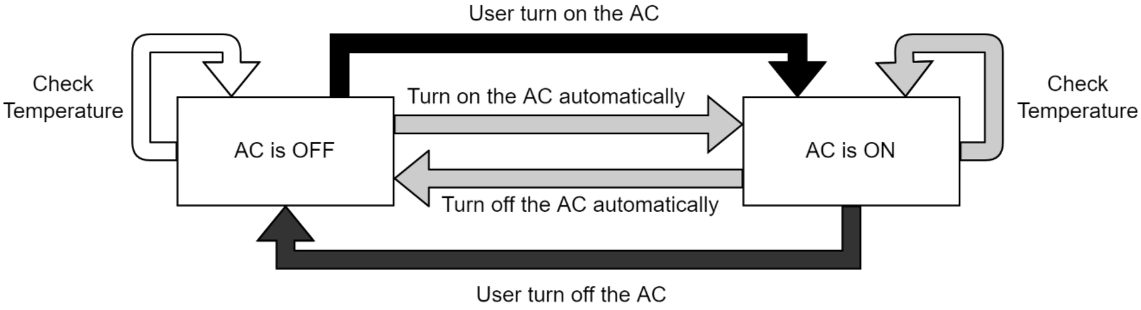


*Figure 8. Conceptual Model for the AC example*

After generating the model, the user should study the artifact carefully to identify every possible threshold. A list of the identified thresholds will be the output of this activity.

After modeling the software system behavior and listing the identified thresholds, the next activity generates an analytical software system model. To accomplish this, the user will use the list of CVs from phase one (Identify the Context Variable) to describe how they should interact with the software system. The output of this task will be the description of all context variables and their expected influence over the system. Table 1 shows an example of the AC situation mentioned previously.

| Context-Variable | Effect |
|---|---|
| Temperature | If bigger than THR, turn on the Air Conditioner |

*Table 1. Analytical Model for the AC example*

The final activity of this phase is to identify every threshold from the previously generated analytical model. The output, in this case, will also be a list of every identified threshold.

### *2.5.3 Generate Test Suite*

After identifying both context variables and threshold, the first activity of this phase is using them to describe the test oracles. Each usage situation or software system feature affected by the context variation should be listed, as well as their threshold and the test oracle (known as expected result) for them. The test oracle must be based on the requirements specification, and any situation without a described test oracle must be identified as "Not Specified."

Once all the information regarding the context is available and the test oracle, the next activity describes the test cases considering both the usage situations and transitions.

Each test case must contain the context variable that may influence the software system behavior, the identified thresholds, and the test oracle obtained through the documentation. The described test cases will be the output of this activity, and, finally, the last activity consists of packaging them in a test suite. A package of context-aware test cases will be the output of the entire CATS Design process.

## 2.6 Chapter Considerations

This chapter presents the most important concepts that guided this work: conventional software testing, context, context-awareness, CASS software testing.

It also presented a literature review and CATS Design: a testing technique capable of helping the specification of test cases for CASS. It is worth mentioning that, although CATS Design does not make explicit how to define a variation of context as the transition from one context situation to another, it uses a very close and intuitive concept, represented by the conceptual model.

Understanding the basic concepts and the challenges presented by CASS was an initial step. However, there were knowledge gaps about the context in the literature and test situations CATS Design cannot cover.

We performed a Structured Review to investigate how the context usually affects real-life applications and software test processes to understand them. The next chapter presents all the information about the performed Structured Review and the gaps found.

# 3 Testing CASS – A Structured Review

## 3.1 Introduction

As it is possible to observe in Chapter 2, the conventional software testing strategies do not consider the context element during the execution of the test case (Figures 2 and 3). Also, evidence shows that when the context is not considered during the testing phase, some failures can occur during the production phase, causing accidents [19].

A structured review was executed to understand how the context affects the real-life application and, consequently, the test process. This chapter shows the protocol used to run this study, the obtained results, and our conclusions.

## 3.2 Structured Review Goal

This study guided the necessity of observing how the context and its variation usually affect context-aware applications in the production phase to suggest an adequate test strategy.

Therefore, the goal was to search for context-aware applications in the literature and observe the context influences in their execution.

## 3.3 Methodology

In November 2018, Amalfitano et al. [11] executed a Rapid Review to investigate the Testing Techniques available for CASS. Due to the similarity between the topics of interest, this study reused their primary sources while searching for Context-Aware applications.

### *3.3.1 Search String*

Amalfitano et al. [11] used Scopus[1] as the search engine. The search string was structured based on PICOC (Population, Intervention, Comparison, Outcome, Context) [21], as can be observed below:

**Population:** Contemporary Software Systems

Synonyms: ("Ambient Intelligence" OR "Assisted Living" OR "Multiagent Systems" OR "Systems of Systems" OR "Internet of Things" OR "Cyber Physical Systems" OR "Autonomous Systems" OR "Autonomic Computing" OR "Multi-Agent Systems" OR "Pervasive Computing" OR "Mobile Computing" OR "Distributed Systems" OR "Cooperative Robotics" OR “Adaptive Systems" OR "Industry 4.0" OR "Fourth Industrial Revolution" OR "Web of Things" OR "Internet of Everything" OR "Contemporary Software Systems" OR "Smart Manufacturing" OR Digitalization OR Digitization OR "Digital Transformation" OR "Smart Cit*" OR "Smart Building" OR "Smart Health" OR "Smart Environment" OR “Digital Transformation” )

**Intervention:** Software Testing

Synonyms: (“Test* Management” OR “Test* Planning” OR “Test* Monitoring” OR “Test* Control” OR “Test* Completion” OR “Test* Design” OR “Test* Type” OR “Test* Implementation” OR “Test* Environment” OR “Test* Execution” OR “Test* Reporting” OR “software test*” OR “software validation” OR “software verification” )

**Comparison:** No

**Outcome:** Software Testing Technologies

[1] https://www.scopus.com/

Synonyms: (“Technique” OR “Technolog*” OR “Method” OR “Activity” OR “Tool” OR “Process” OR “Practice” OR “Mechanism” OR “Instrument” OR “Task” OR “Service” OR “Strategy”)

**Context:** (“Variation” OR “Context” OR “Context Awareness” OR “Context Variation”)

Additionally, the articles were limited to Computer Science and Engineering area from 2002 to 2019.

### *3.3.2 Selection Procedure*

One researcher performed the following selection procedure, while a second researcher reviewed the entire process and results:

1. Run the search string
2. Apply the inclusion criteria based on the paper Title
3. Apply the inclusion criteria based on the paper Abstract
4. Apply the inclusion criteria based on the paper Full text

### *3.3.3 Inclusion Criteria*

1. The paper must be in the domain of software engineering;
2. The paper must be in the domain of Contemporary Software Systems
3. The paper must report a primary study
4. The paper must present a system or application influenced by the context
5. The paper must be written in the English language

## 3.4 Extraction Form

In this study, two different extraction forms were used. The first one, presented in Table 2, had the goal of capturing important information about the article in general. The second one (Table 3) aimed to capture important information about the applications themselves. More than one application was presented in the same article in some cases. In cases like these, each application was captured by a different form.

Tables 2 and 3 also present an excerpt of the application extracted from Mirza and Khan [22].

| Paper ID | 12 |
|---|---|
| Bibliography: | Mirza, A. M., & Khan, M. N. A. (2018). An Automated Functional Testing Framework for Context-aware Applications. *IEEE Access*, *6*, 46568-46583. |
| Abstract: | “In the modern era of mobile computing, context-aware computing is an emerging paradigm due to its wide spread applications. Context-aware applications are gaining increasing popularity in our daily lives since these applications can determine and react according to the situational context and help users to enhance usability experience. However, testing these applications is not straightforward since it poses several challenges, such as generating test data, designing context-coupled test cases, and so on. However, the testing process can be automated to a greater extent by employing model-based testing technique for context-aware applications. To achieve this goal, it is necessary to automate model transformation, test data generation, and test case execution processes. In this paper, we propose an approach for behavior modeling of context-aware application by extending the UML activity diagram. We also propose an automated model transformation approach to transform the development model, i.e., extended UML activity diagram into the testing model in the form of function nets. The objective of this paper is to automate the context-coupled test case generation and execution. We propose a functional testing framework for automated execution of keyword-based test cases. Our functional testing framework can reduce the testing time and cost, thus enabling the test engineers to execute more testing cycles to attain a higher degree of test coverage.” |
| General Information: | • “Our proposed framework would automate testing process of context-aware applications which includes generation and execution of context-coupled test cases to evaluate accuracy of context recognition and adaptation.”<br>• “Test cases generated from test model are in the form of abstract test cases, so they are platform and tool independent. Abstract test cases are human readable and can be executed manually. To execute generated test cases automatically, abstract test cases need to be converted according to tool specific test scripts referred as concrete test scripts [44].” |
| Context-Awareness Information: | • “To test context-aware applications, it is important to understand these features and plan test strategy accordingly. Few important features of context-aware applications are context, quality of context, sources of context, context interpretation and reasoning.”<br>• “Context information is retrieved from different sources which can be grouped into two broad categories, physical sensors and data |

| | |
|---|---|
| | sensors. Examples of physical sensors are GPS, heat and proximity sensors which are used to obtain location and temperature of the device as well as proximity to other neighboring devices respectively. Similarly, examples of data sensors include preferred usage profiles, social networking profiles, calendar and task list of a smartphone. However, context information retrieved from both types of sensors can introduce imperfection e.g., ambiguity, imprecision, errors/omissions about the sensed context due to many reasons such as noise or failure of sensors [7]. These imperfections in the context information may cause context-aware application to behave erroneously.” |
| Study Type: | “To validate our framework, we conducted two case studies and results of these case studies are compared with the results of selected contemporary studies.” |

*Table 2. Excerpt of the application extracted from Mirza and Khan*

| Application’s Name | Call-a-Cab App |
|---|---|
| Application Description | • “Our first case study is based on call-a-cab context-aware application [47]. This application allows users to call a cab to their current location. User location can be obtained using GPS sensor or can be fed manually. If application fails to automatically obtain GPS location, then it reverts to manual mode requiring the user to feed the location.”<br>• “we identify three context reconfiguration points (Call-a-Cab-GPS, Call-a-Cab-Manually and Network Available) where application needs to collect current context to carry out further functionality. While calling a cab using GPS, if GPS connection fails then application will fall back to manual mode. Similarly, while calling cab manually, if GPS location is found, application will fall back to automatic mode. After obtaining user location,application needs to send cab request using cellular network. If cellular network is lost, then an error dialog will be displayed otherwise request will be sent.” |

*Table 3. Excerpt of the application extracted from Mirza and Khan*

The extraction form for each selected article will be presented entirely in Appendix A.

## 3.5 Results

### *3.5.1 Summary of the Findings*

The Search string mentioned in section 3.3.1 returned 492 articles. As Table 4 shows, from the 492, 54 were selected considering the title and 35 considering the abstract.

| **RR Testing Database (Nov. 11th, 2018)** | |
|---|---|
| Total | 492 |
| Selected by Title | 54 |
| Selected by abstract | 35 |
| **Selected Applications** | 2 |

*Table 4. Summary of the Findings*

After a full reading, only two articles were selected considering the inclusion criteria presented in Section 3.3.3. Their name, author's information, and descriptions of the applications they contain are shown below (Table 5).

| **Article Name** | An Automated Functional Testing Framework for Context-Aware Applications [22] | Software Adaptation in Wireless Sensor Networks [23] |
|---|---|---|
| **Authors** | Aamir Mehmood Mirza, Muhammad Naeem Ahmed Khan | Mikhail Afanasov, Luca Mottola, Carlo Ghezzi |
| **Application Name** | Call-a-Cab App | Wildlife Tracking Application |
| **Application Description** | "Our first case study is based on call-a-cab context-aware application [47]. This application allows users to call a cab to their current location. User location can be obtained using GPS sensor or can be fed manually. If the application fails to automatically obtain GPS location, then it reverts to manual mode requiring the user to feed the location. Testing this application requires test cases to include location determination modes, setting valid and invalid location and manipulation of the network connection | "Battery-powered WSN nodes are embedded in collars attached to animals, such as zebras or badgers. The devices are equipped with sensors to track the animals' movement, such as GPS and accelerometer readings, and to detect their health conditions, for example, based on body temperature. Low-power short-range radios are used as proximity sensors by allowing nodes to discover each other whenever they |

| | | |
|---|---|---|
| | to simulate unexpected service loss. (...) While calling a cab using GPS, if GPS connection fails then application will fall back to manual mode. Similarly, while calling cab manually, if GPS location is found, application will fall back to automatic mode. After obtaining user location,application needs to send cab request using cellular network. If cellular network is lost, then an error dialog will be displayed otherwise request will be sent.” | are within communication range, using a form of periodic radio beaconing. A node logs the radio contacts to track an animal’s encounters with other animals, enabling the study of their social interactions. The radio is also used to off-load the contact traces when reaching a fixed base station. Small solar panels harvest energy to prolong the node lifetime [5].” |

*Table 5. Summary of the Findings*

### 3.5.2 Discussion

Based on the initial number of returned articles (492) and the search string, the expectation was to find a higher number of applications. Therefore, the result raises the question: "Why were there just two applications in the final set?" After reviewing the entire process, it was possible to realize that many applications called context-aware for their authors were not context-aware according to our understanding.

As was mentioned before, context is an abstract concept. Consequently, it can be used to represent different situations. Each situation will influence the test process differently and require a specific test strategy considering the testing perspective.

For this reason, the conventional test case model (Figure 3) was evolved to consider the context and to represent how the context can affect the test creation of test cases. Therefore, the next subsection presents the CASS Test Case Model.

Finally, it is important to mention that although it was impossible to capture many context-aware applications, this study was essential to increase the understanding of the context and help notice these different interpretations. Additionally, the knowledge acquired in this step guided the next ones.

### 3.5.3 CASS Test Case Model

After finishing the structured review process, it was possible to notice that, from the testing perspective, the variation of context would either affect the test input or the test conditions.

For example, imagine an application collecting the user's location, usually using GPS, to show restaurants nearby. Suppose that the GPS is disabled (such as the smartphone running out of battery and disabling the GPS consequently). The application should adapt to the new context situation and ask the user to enter his/her location manually. The first case (showing restaurants nearby based on GPS location) would be an example of context manifested as test input (user's location). However, in the second case (low battery), the context information would affect all testing conditions (GPS signal and battery).

The above example shows two different scenarios that the conventional test case model cannot capture the situation, as shown in section 2.2. However, these scenarios and how they will affect the test execution must be captured because they require different test strategies.

Due to the necessity of representing them, a new test case model was created. Figure 9 presents its first version. While the conventional test case model is based only on Inputs (I), Conditions (C), and Expected Results (E), the new one introduces the Context (represented by Context Variables) as the fourth element of the model.

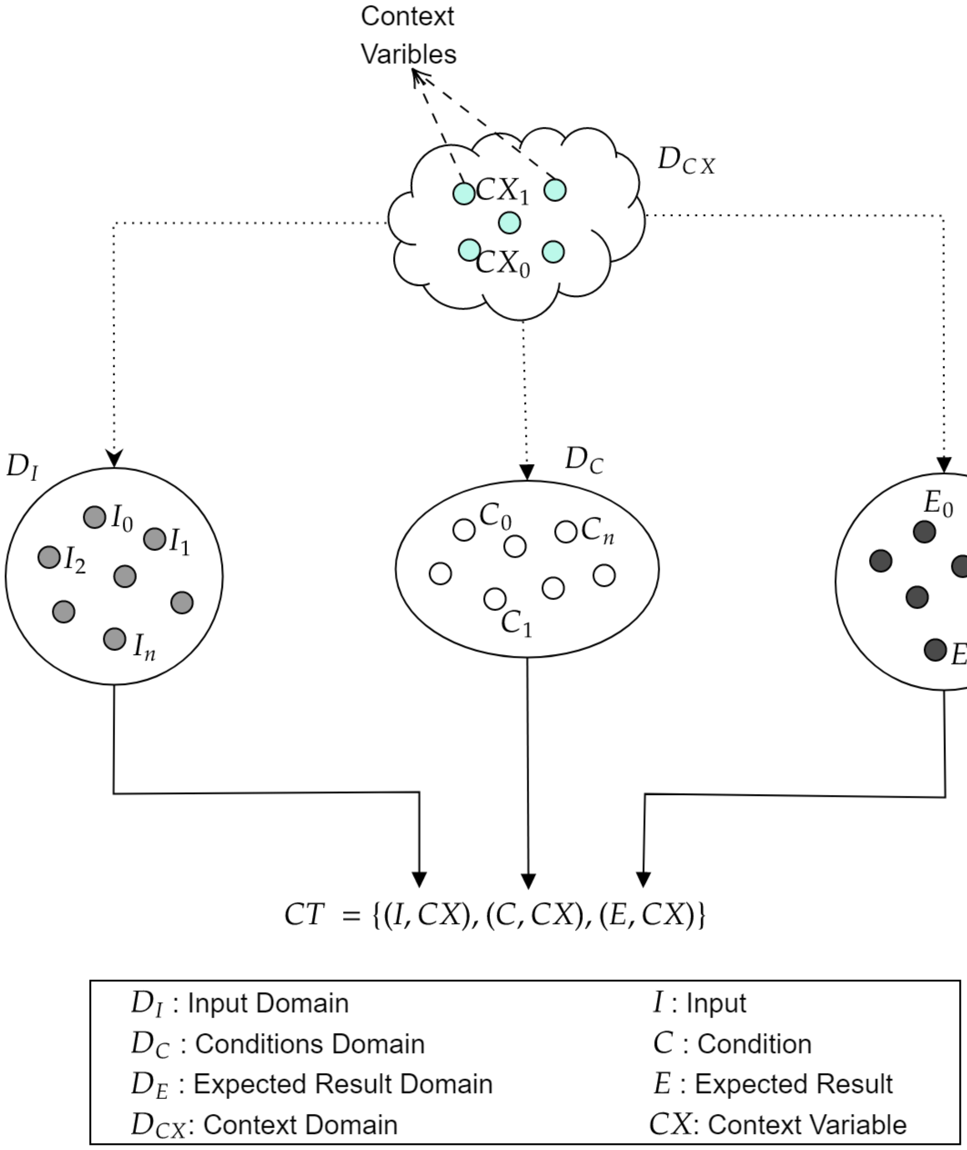


*Figure 9. CASS Test Case Model – v1*

In this way, the Input and the Conditions will combine each one with the Context Variable they are influenced. Consequently, the Expected Result will be attached with the same CV because each context element will produce a different Expected Result.

This model can represent different scenarios. For example:

- **Conventional**: in this case, there will be no context (CX = {}). Thus, the test case will be: CT = {I, C, E}

- **Context affecting the Test Input**: in this case, the context will affect the Test Input (I). The context will not affect the conditions during all the test processes. Thus, the test case formula will be CT = {(I, CX), C, (E, CX)}.
- **Context affecting the Test Conditions**: in this case, the context will affect the Test Conditions (C). The Input will not be affected. Thus, the test case formula will be CT = {I, (C, CX), E}.
- **Context affecting both the Test Input and Conditions**: in this case, the context will simultaneously influence the Input and Conditions. Thus, the test case formula will be presented in Figure 9: CT = {(I, CX), (C, CX), (E, CX)}

Table 6 summarizes the presented test scenarios.

| **Test Case Model** | **Input (I)** | **Condition (C)** | **Expected Result (E)** |
|---|---|---|---|
| **Conventional** | Not Affected | Not Affected | Not Affected |
| **CASS Model A** | Affected | Not Affected | Affected |
| **CASS Model B** | Not Affected | Affected | Affected |
| **CASS Model C** | Affected | Affected | Affected |

*Table 6. CASS Test Scenarios*

Although the model can represent the four mentioned scenarios, it does not represent the correct relation between the context variables and the other test case variables. Therefore, it needed to evolve. As a result, the model in Figure 9 evolved throughout this research. These evolutions will be presented in the next chapter.

## 3.6 Chapter Considerations

This Chapter presented the Structured Review to investigate how the context and its variation usually affect context-aware applications in the production phase. The primary sources of Amalfitano et al. [11] were reused due to the similarity between the topics of interest. However, while they searched for testing techniques available for CASS, we were looking for Context-Aware Applications.

After going through the process, just two applications were selected from 492 articles, which raises the question, “why were obtained such a small number of applications?”. While reviewing the process, it was possible to notice that what the authors defined as Context-Aware applications were not Context-Aware according to our understanding of the topic. Therefore, the Conventional Test Case Model was enhanced to capture these differences and how each will influence the Test Process, creating the CASS Test Case Model (which considers the context).

It is worth mentioning that although this study was not able to capture a lot of context-aware applications, it was essential to increase the understanding of the context and to help notice the existence of these different test situations. Thus, the knowledge acquired in this study guided the next steps.

# 4 CATS#: Towards Evolving CATS Design

## 4.1 Introduction

The testing of CASS is the research question that drives this work. While looking for solutions, the two first phases presented in Figure 1 were executed: Acquire initial knowledge on the Problem (described in Chapter 2) and Complement by searching the literature (described in Chapter 3). In phase 2, it was also possible to study the main concepts regarding CATS Design, a technique grounded in evidence to describe context-aware test cases for Ubiquitous Systems. It presented interesting ideas, an intuitive process, and useful characteristics.

The main finding from the structure review (described in Chapter 3) was the necessity of adapting the test strategy according to test scenarios. CATS design addresses many important issues about the test for context-awareness application. However, it does not consider how the context can differently affect the test cases and test process. Therefore, we decided to use it as a start point and use our findings from Chapter 3 to evolve the technique.

As a tribute to the shoulders this work stands on, the technique developed during this research is CATS#. It aims to evolve CATS Design to deal with CASS test scenarios (mentioned in Chapter 3) while evolving the entire process to make it suitable for different types of CASS (not only Ubiquitous Systems, as CATS Design does).

This chapter presents the process of evolving CATS Design to become CATS#. At this phase, the methodology to build the solution was based on Learn, Adapt/Build and Measure, iteratively [24], as presented in Figure 10.

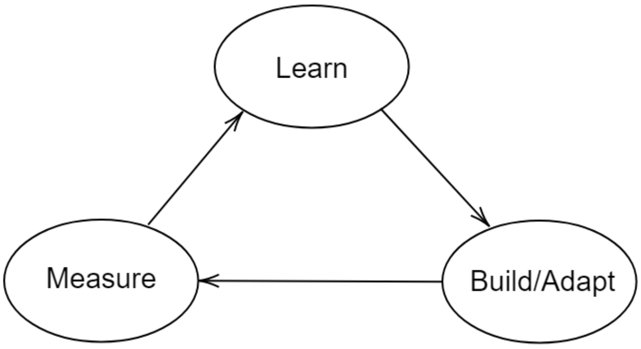


*Figure 10. Methodology for constructing CATS#*

We learned, previously, from CATS Design the advantages of building a first version of the technology and performing an internal evaluation to allow its incremental evolution. As a result, CATS# passed through two intermediary versions before getting into its final form. The intermediary versions (CATS# v1 and v2) and their internal evaluations are exposed in this chapter. The final version of CATS# is presented in Chapter 5.

## 4.2 The first version of CATS#

### *4.2.1 The CATS# v1 process*

At this first version, the main idea was evolving CATS Design to englobe the knowledge acquired with the Structured Review (see Chapter 3). The resulting technique (CATS#) should conduct software engineers to specify CASS test cases considering the different test scenarios. Furthermore, the intention was also to evolve the process to facilitate its application.

Figure 11 presents the first version of CATS#. It contains eight steps: two exactly equal to CATS Design (Generate analytical model and Describe the test oracle), five that were modified at some level (Extract Variables, Identify Context Variables, Generate Conceptual Model, Identify the thresholds, and Describe Test Cases), and one new (Identify Test Scenario). The new step aims to include the test scenarios in the process. Four of the five modified steps had the goal of evolving the CATS Design process itself. The exception is the Describe Test Cases step which proposes a new test template to consider the specific pieces of context variation.

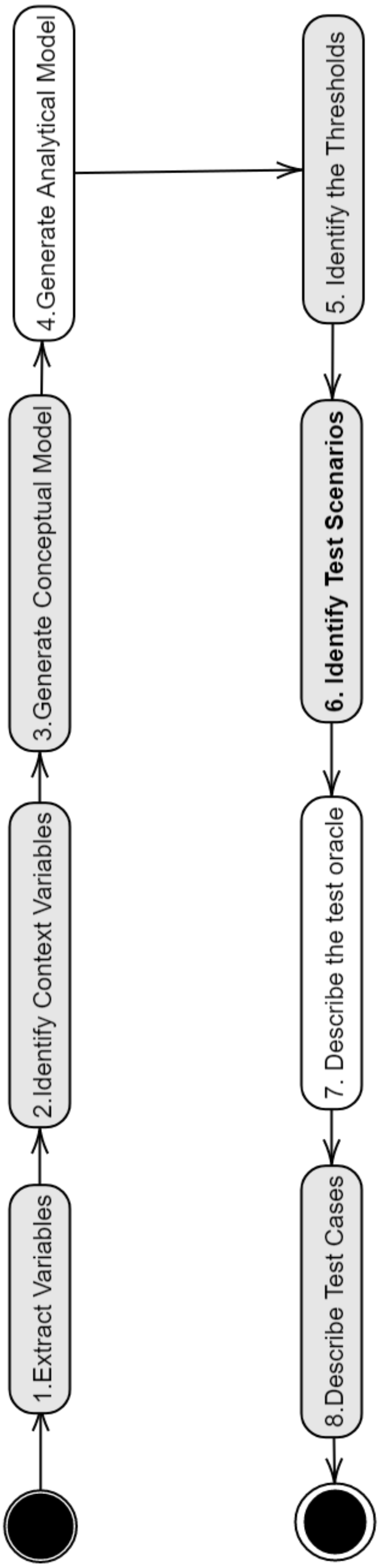

1.Extract Variables
2.Identify Context Variables
3.Generate Conceptual Model
4.Generate Analytical Model
5. Identify the Thresholds
6. Identify Test Scenarios
7. Describe the test oracle
8.Describe Test Cases

*Figure 11. CATS# Process*

### 1. Extract Variables

This step condenses the two initial steps from CATS Design. Then, it extends them to all possible variables presented in the Requirements Document (or any other artifact describing how the software system is supposed to work). The rationale is that the variables that do not represent the context itself could be affected by its variation. Therefore, they can help analyze how the context affects the software system's behavior.

### 2. Identify Context Variables

Once all possible variables are listed, the intention is to identify those representing the context. It is not necessary to create a new list at this step. The idea is that using the list from step 1, the user marks the context variables with any graphical element (such as underlining, highlighting, and so on) that differentiates them from the regular ones.

### 3. Generate Conceptual Model

This step remains almost the same as CATS Design. The only modification was regarding the arrow representing the transitions. The ones from CATS Design were fulfilled and colored. The one used by CATS# was simple in this first version. The arrows used by both techniques are presented in Figure 12. The idea of changing this graphical element was to simplify the model and make it easy to be designed in different tools.

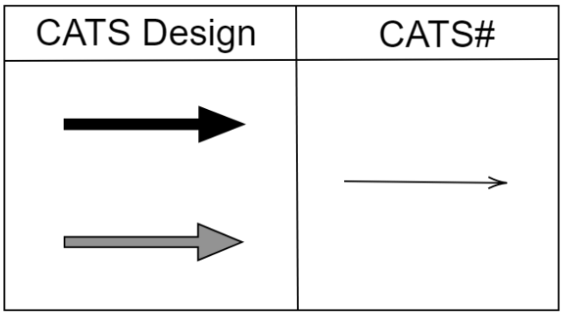


*Figure 12. Difference Between Arrows*

### 4. Generate Analytical Model

This step remains almost the same as CATS Design. The model preserves the two fields that must be filled out: Context-Variable and Effect. Nevertheless, while evolving the process, it became clear that, since the Effect will be a consequence of a CV reaching a THR, the THR itself should be included in this model since it is a part of the effect.

Table 7 shows an example using part of the Restaurant Application scenario from section 3.5.3. The model in Table 7 represents how the context interacts with the system and how the system should behave.

| Context-Variable | Effect |
|---|---|
| GPS Availability | • If it is TRUE, it automatically gets the user's location<br>• If it is FALSE, ask the user's location manually |
| Internet Connection | • If it is TRUE, return to restaurants nearby<br>• If it is FALSE, not defined |

*Table 7. Conceptual Model of the Restaurant Application*

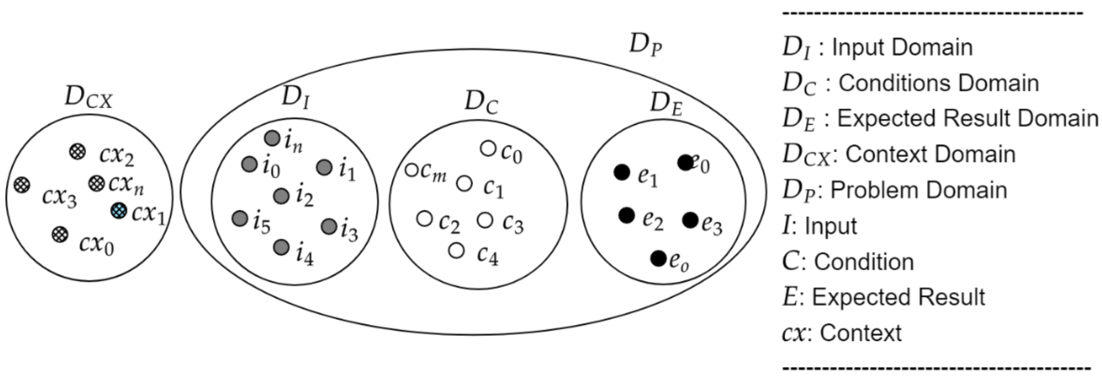
5. **Identify the Thresholds**

This step synthesizes two steps from CATS Design: "Identify the Thresholds in the Conceptual Model" and "Identify the Thresholds in the Analytical Model." The idea is that both models will work together. THRs from one model should be reflected in the other. Consequently, there is no necessity of going through the step twice.

6. **Identify Test Scenarios**

This step is new and aims to use a new Test Case model to represent the context, its interaction with the Test Case, and, consequently, the Test Process.

The model in Figure 9 evolved to the one in Figure 13, using the set theory to represent the problem better. Besides, it evolves the relation among context variables and the other test case variables (Input, Conditions, and Expected Results) while still capable of describing the four scenarios mentioned in section 3.5.3.



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

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

| | |
|---|---|
| 1. No Context<br>$I = \{i_i : i_i \in D_I\}$<br>$C = \{c_i : c_i \in D_C\}$<br>$E = \{e_i : e_i \in D_E\}$ | 2. Context influences the Input<br>$I = \{(i_i,\ cx_i): i_i \in D_I,\ cx_i \in D_{CX}\}$<br>$C = \{(c_i : c_i \in D_C\}$<br>$E = \{(e_i,\ cx_i): e_i \in D_E,\ cx_i \in D_{CX}\}$ |
| 3. Context influences the Conditions<br>$I = \{i_i : i_i \in D_I\}$<br>$C = \{(c_i, cx_i): c_i \in D_C,\ cx_i \in D_{CX}\}$<br>$E = \{(e_i,\ cx_i): e_i \in D_E,\ cx_i \in D_{CX}\}$ | 4. Context influences the Conditions and the Input<br>$I = \{(i_i,\ cx_i): i_i \in D_I,\ cx_i \in D_{CX}\},$<br>$C = \{(c_i, cx_i): c_i \in D_C,\ cx_i \in D_{CX}\},$<br>$E = \{(e_i,\ cx_i): e_i \in D_E,\ cx_i \in D_{CX}\}$ |

*Figure 13. CASS Test Case Model – version 2*

This model is relevant because, as was mentioned before, how the context variation affects the test case will influence the test strategy. Therefore, it is important to guarantee that the test cases and testing execution capture the same variations of context that affect the software system in the production phase when the context can freely vary [9].

The context needs to vary during the testing execution. Furthermore, as far as it could be investigated in this research, these are the strategies that should be followed according to the model from Figure 13:

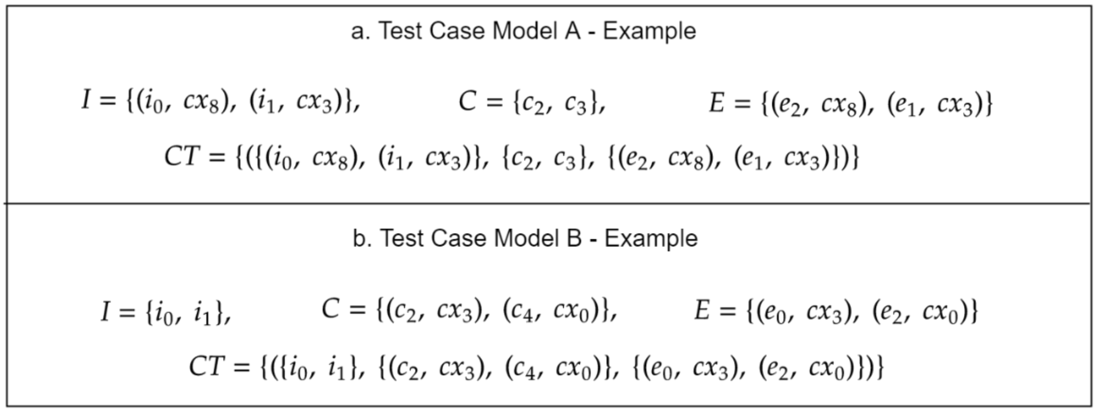
- As mentioned before, when there is no context, there will be no need to capture its variation. Therefore, the model gets back to Conventional Test Case Model, and conventional test strategies should be used.

- **When the context influences just the Test Case Input** and the conditions remain static, the context variations will not affect the testing execution. The only consequence will be increasing the number of test cases to represent the different context variations.
- **When the context influences just the conditions** and the input remains static, it is necessary to use a new test strategy to capture the context variations during the testing execution.
- **When the context influences the input and the conditions simultaneously**, it will be necessary to use a proper test environment capable of varying both Input and Conditions during the testing execution. Unfortunately, there is no such environment available yet [11]. Therefore, it is necessary to create a new environment capable of using computing techniques to simulate these variations. However, this is out of the scope of this work, being a future step.

A summary of these testing strategies is presented in Table 8, which shows a different perspective of the test scenarios presented in Chapter 3. Additionally, Figure 14 shows examples of how the test case would be specified following the CASS Test Case Model A (Figure 14.a) and CASS Test Case Model B (Figure 14.b).

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

*Table 8. Testing Strategies*



a. Test Case Model A - Example

$$I = \{(i_0,\ cx_8),\ (i_1,\ cx_3)\},\qquad C = \{c_2,\ c_3\},\qquad E = \{(e_2,\ cx_8),\ (e_1,\ cx_3)\}$$

$$CT = \{(\{(i_0,\ cx_8),\ (i_1,\ cx_3)\},\ \{c_2,\ c_3\},\ \{(e_2,\ cx_8),\ (e_1,\ cx_3)\})\}$$

b. Test Case Model B - Example

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

$$CT = \{(\{i_0,\ i_1\},\ \{(c_2,\ cx_3),\ (c_4,\ cx_0)\},\ \{(e_0,\ cx_3),\ (e_2,\ cx_0)\})\}$$

*Figure 14. CASS Test Case Model – version 2 - Examples*

7. **Describe the Test Oracle**

The previous step had the goal of helping the software tester to perceive whether the context will affect the Input or the Conditions of the test cases. The software tester should use this information to specify the test cases, creating the test oracle at this step.

In CATS Design, the Test Oracle was composed of three elements: Feature, Context, and Expected Output (Table 9 shows an example).

| Feature | Context | Expected Output |
|---|---|---|
| Search for Restaurants | GPS going unavailable | Ask location manually |
| Search for Restaurants | Loss of Internet Connection | Not specified |

*Table 9. CATS Design Test oracle*

However, in CATS#, the Test Oracle suffered some modifications. The most important one is that the context information will be captured through Inputs and Conditions Variables. Additionally, each test case must have an Identification Number (Id), and the "Expected Output" column was renamed for "Expected Result" to maintain the consistency between this step and the previous one.

Table 10 shows an example of the CATS# Test Oracle. Since the context will directly affect the Input and Conditions variables of a test case, its variation should be captured through them. In the example in Table 10, the context affected the GPS availability, a condition variable. In this way, the context variation is represented at the condition's column with the "GPS Available" variable varying between the value "True" to the value “False.” In addition, the Expected Result column will capture the system’s expected behavior after the context variation. The same applies to the Internet Connection variable.

| Test Case Id | Input | Conditions | Expected Result |
|---|---|---|---|
| TC01 | User’s location | GPS available = True → GPS available = False | Ask location manually |
| TC02 | User’s location | Int. Connection = True → Int Connection = False | Not specified |

*Table 10. CATS# Test Oracle*

### 8. Describe the test case

After creating the test oracle, the test cases must be described so that it will be possible to execute the testing later. It is the most significant step in the CATS# process and the most different from CATS Design.

CATS Design focuses on preparing a test oracle. Then, if any context variation occurs during the test case execution, the oracle would have the behavior expected from the software system in the face of that situation.

In CATS#, the intention is to cause the context variation during the testing execution because it is the only way to verify the system behavior in that situation. Therefore, a specific CV is chosen to vary, while the others must remain fixed during testing. With this goal, the CATS Design template was modified. The new one is presented in Table 11 and has the goal of helping the user to specify which variation must occur and when, exactly, it must occur.

| **Id** | <test case id> |
|---|---|
| **Test Objective** | <Briefly describes the test case goal objective> |
| **Preconditions** | <Describes the preconditions in general> |
| **Fixed Conditions** | <Describes the conditions that must remain static during the test execution> |
| **Input** | <Describes the test case input> |
| **Test Steps** | 1. <Describes the first step that should be followed><br>2. <Describes the second step …><br>3. <Describes the third step …> (c1)<br>4. <Describes the fourth step, after the variation specified in c1> |
| **Varying Conditions** | c1. <Describes the specific conditions that must vary in test step 3> |
| **Expected Result** | <Describes the expected result after the variation specified in c1> |
| **Pos Conditions** | <Describes the pos conditions in general> |

*Table 11. CATS# Test Template*

As shown in Table 11, a new field called “Fixed Conditions” is responsible for specifying which Context Variables, previously identified as test case conditions, must remain static during testing execution. Besides, the “Varying Conditions” field specifies which CV must vary (considering the previously established threshold).

Additionally, the CATS# template gains a structured flow like use cases. The idea is to indicate in which step an exception flow must occur, using tags, such as the "c1" presented in Table 11, to identify when a specific context variation must occur. In this way, if there is a tag 'c1' after step 3, it means the testing execution must be momentaneous paused until the variation specified as 'c1' happens. Thus, the software system must remain paused until the context variation occurs and, once it is over, the test execution can proceed to step 4.

The other presented fields are characteristics of conventional testing templates and are also present in CATS Design. Table 12 shows a comparison between the templates of CATS Design and CATS# v1.

| **CATS Template** | **CATS# v1 Template** |
|---|---|
| Test Case ID | Test Case ID |
| Test Objective | Test Objective |
| Precondition | Precondition |
| <*not available*> | Fixed Conditions |
| Test Input | Test Input |
| Test Steps | Test Steps |
| Relevant Context Variables | Varying Conditions (C) |
| Known Threshold | <*not available*> |
| Expected Result for each Threshold | Expected Result (E) |
| Postconditions | Postconditions |

*Table 12. CATS Test Template x CATS# v1 Test Template*

Step 8 is the last step from the CATS# process. After this, it is expected that the software engineer will be capable of

- Identify how the context will affect the software system and the test cases; and
- Specify the test cases for test scenarios when the context influences the input; and
- Specify the test cases for test scenarios when the context influences the conditions; and
- Use the test template to specify both CASS and conventional test cases whether it is necessary

This process focuses on test case specification. On some occasions, causing the variations will be quite common, such as Disable GPS, Losing Internet Connection, Increasing temperature, among others.

However, there will be occasions that demand a specific environment to simulate the necessary context variations. For these cases, just the CATS# process will not be enough. Developing a context-aware testing environment is out of the scope of this work. However, it is necessary to strengthen the benefits of using CATS# to test context-aware software systems.

After this first version, it was necessary to evaluate it. Therefore, a proof of concept was developed using an example application. The next section will describe the application and present CATS# v1 to specify CASS test cases.

### 4.2.2 Internal evaluation of CATS# v1

At this step, the idea was to use a real-life application from Afanasov, Mottola, and Ghezzi [23] to use CATS# v1 and evaluate the technique. The software system is a Wildlife Tracker to monitor the behavior of wild animals and their encounters.

The solution is a WSN (Wireless Sensor Network). Battery-powered nodes are embedded in collars and attached to animals. In addition, each node is equipped with a GPS sensor, two low-powers short-range radios working as proximity sensors, and solar panels to prolong the node lifetime. A proximity sensor is responsible for detecting the presence of Base-Stations (BS), and the other is responsible for detecting the presence of animals (Figure 15). When an animal is detected, the information about the encounter should be logged in (Figure 16.b). Then, the collected data should be sent to the Base-Station (Figure 16.c). Additionally, the GPS sensor captures the pace of the animal's movement, and it may be disabled if the battery is running low (Figure 16.a). More information about the application can be found in Afanasov, Mottola, and Ghezzi [23].

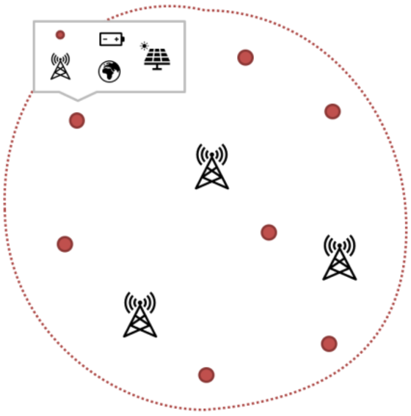


*Figure 15. Wildlife Tracker Application*

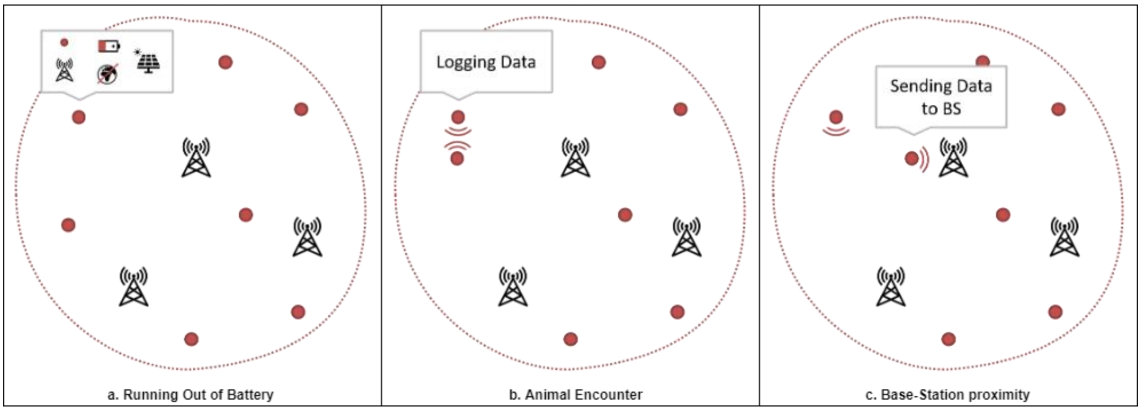


*Figure 16. Wildlife Tracker Scenarios*

The CATS# v1 technique was applied to specify the test cases for the Wildlife Tracker Application according to the steps below.

### 1. Extract Variables

The first step should capture the variables from the Requirement Document (or any document containing the problem specification). In this case, the application description was used. The variables list is presented below in Table 13.

### 2. Identify Context Variables

After listing the variables, the second step highlighted the ones that appear to be Context Variables. In this example, the variables representing the context are **bold** in Table 13.

| **List of Variables** | |
|---|---|
| • GPS Sensor status | • **Base-Station Proximity** |
| • Animal location | • Memory available |
| • Accelerometer | • **Battery level** |
| • **Animal Proximity** | |

*Table 13. Wildlife Tracker - List of variables*

### 3. Generate Conceptual Model

After identifying the CVs, the conceptual model was designed. Figure 17 presents the model. It has four states:

1. **Log-in information from GPS:** when the battery level > THR and there is no proximity with either a BS or another animal. This "state" was considered "normal," the initial one. The other states will have this one as a starting point.
2. **Disable GPS:** occurs when the battery level reaches a pre-defined THR. It was considered that the THR is a value that the user will define. However, its real value does not matter at this point, but the context changes once it is reached, and the system must adapt.
3. **Send data to the BS**: occurs every time an animal gets closer to a BS. Once the proximity is over, the software system must go back to the log-in information "state."
4. **Collect data from the animal's encounter:** every time another animal is near. Like the previous "state," the system must go back to the log-in information state once the proximity is over.

The transitions and thresholds that will "trigger" them are more valuable than the states. They help the software engineer understand the context variation and the consequences (or effects) of these transitions.

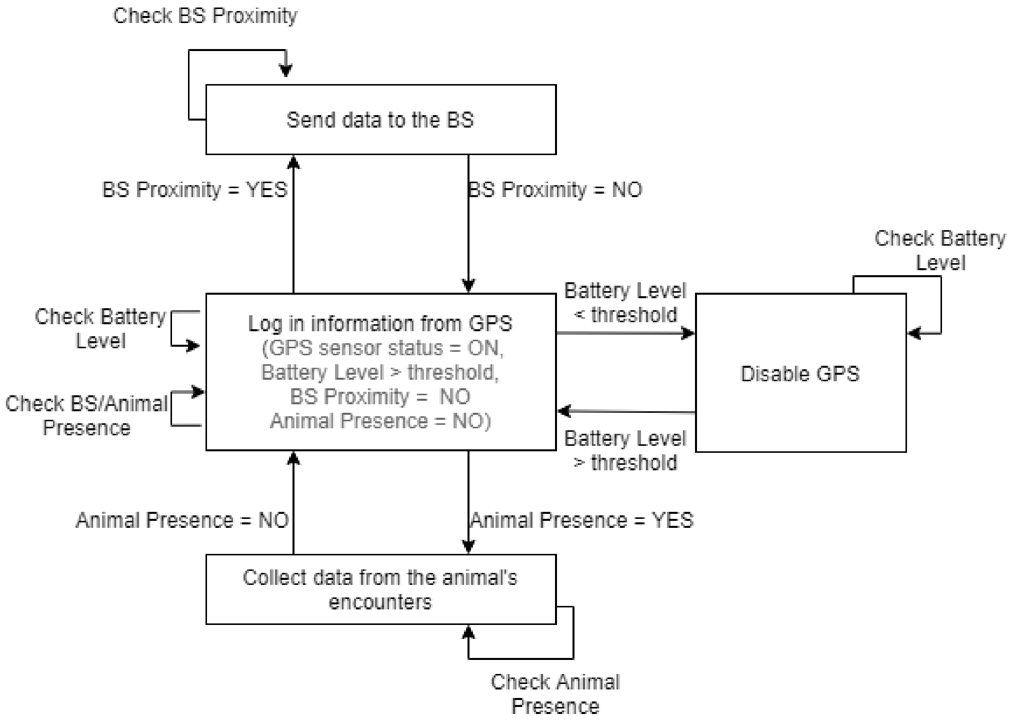

Check BS Proximity
Send data to the BS
BS Proximity = YES
BS Proximity = NO
Check Battery Level
Battery Level < threshold
Check Battery Level
Log in information from GPS (GPS sensor status = ON, Battery Level > threshold, BS Proximity = NO Animal Presence = NO)
Disable GPS
Check BS/Animal Presence
Battery Level > threshold
Animal Presence = NO
Animal Presence = YES
Collect data from the animal's encounters
Check Animal Presence

*Figure 17. WildLife Tracker Conceptual Model*

### 4. Generate Analytical Model

Here, the difference from CATS Design is that the Analytical Model was based on the problem description and the Conceptual Model.

Table 14 presents the model. The first row of the table presents two "states": "Log-in information from GPS" (when the battery level ≥ THR) and "Disable GPS" (when the battery level < THR). The second row is related to the "Collect data from the animal's encounters" state (when animal proximity = YES), and the third row is related to the "Send data to BS" state (when Base-Station Proximity = YES).

| CV | Effect |
|---|---|
| Battery level | • if battery level < THR, GPS status = OFF<br>• if battery level ≥ THR, GPS status = ON |
| Animal Proximity | • if animal proximity = YES, collect data from encounters |
| Base-Station Proximity | • if Base-Station Proximity = YES, send data to the Base-Station |

*Table 14. Wildlife Tracker Analytical Model*

### 5. Identify the Thresholds

After designing both Conceptual and Analytical Models, the next step is the identification of their thresholds. Finally, the identified THRs are listed in Table 15. As expected, the THRs are related to the CVs described in the Analytical Model (Table 14) and the transitions from the Conceptual Model (Figure 17).

| Thresholds |
|---|
| • Battery level going down the threshold<br>• Battery level going up the threshold<br>• Getting closer to a base-station<br>• Getting closer to an animal |

*Table 15. Identified Thresholds*

### 6. Identify the Test Scenarios

After getting all three elements: context variables, thresholds, and effects, it is important to classify each test case according to the CASS Test Scenarios: Conventional, CASS Model A, CASs Model B, CASS Model C. Table 16 presents the classification for this problem.

| Context Variable | Effect | Threshold | Test Scenarios |
|---|---|---|---|
| Battery Level | ≥ THR, GPS status = ON | Battery Level going down the threshold | CASS - Model B |
| | < THR, 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 |

*Table 16. Identification of the test Scenarios*

The first two rows describe the battery level going down and up the threshold. Both cases were classified as CASS Test Case Model B since the "battery level" variable would be a test case condition.

The other two remaining rows of the table describe the scenarios involving Animal or Base Station Proximity. They were classified as CASS Test case Model A since they fit better as a test case input.

7. **Describe the Test Oracle**

At this step, all test scenarios were classified according to the test strategy that should be applied. Four test cases were specified (Table 17) considering Table 16 and the proper testing strategies for CASS – Modal A and B.

- **TC01**: In this case, the CVs were related to the Conditions. Thus, the CASS Test Case Model B was used, and the Battery Level must vary during the test execution.
- **TC02 and TC03**: The CVs were related to the Input in these cases. Thus, the CASS Test Case Model A was used. Since the context variation occurs in the input, two test cases were specified to test this scenario: BS Proximity = NO, and BS Proximity = YES. During all testing execution, the Battery Level must remain bigger than the THR value.

- **TC04** is a conventional scenario with no variation occurring during the test execution. Thus, a conventional test case was specified.

| 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 |
| TC03 | BS Proximity = YES | CX3 | | Send Data to BS |
| TC04 | Animal location | CX0 | Bat. Level ≥ *THR* | Log in GPS information |

*Table 17. WildLife Tracker Test Oracle*

### 8. Describe the Test Cases

After describing the test oracle, the last step was using the CATS# v1 testing template to describe how the test process should occur. Table 18 presents the fulfilled test template from section 4.2.1 for the test case TC01.

The goal of TC01 was to verify the variation of the Battery Level while all other CVs remain constant. Therefore, the specified test steps should be followed to test this functionality. After executing the third test step, the system must be paused, and the variation in c1 must be provoked (manually or automatically). After the variation occurs, the next test step should be executed. The expected behavior, in this case, is the software system disabling the GPS sensor.

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

*Table 18. Wildlife Tracker test template*

### *4.2.3 Discussion*

As was mentioned previously, we used CATS Design as inspiration and evolved its process into CATS# v1. Table 19 shows the difference between the two processes. As it is possible to observe, there were:

1. Removed steps - such as “Analyze the Requirements looking for Context Variables” and “Identify Additional Context Variable”;
2. Maintained steps - such as “Generate Conceptual Model” and “Generate Analytical Model
3. Evolved steps from CATS Design - such as “Extract Variables” and “Identify Context Variables”; and
4. New steps included - such as “Identify the Test Scenarios.”

The main idea was to simplify the process and use our findings from Chapter 3 to evolve the technique.

| Steps | CATS Design | CATS# - First Version |
|---|---|---|
| Analyze the Requirements looking for Context Variables | Included | Removed |
| Identify Additional Context Variable | Included | Removed |
| Extract Variables | It did not exist in this version | An evolution of step 1 from CATS Design |
| Identify Context Variables | It did not exist in this version | An evolution of step 2 from CATS Design |
| Generate Conceptual Model | Included | Maintained |
| Identify the THR in the Conceptual Model | Included | Removed |
| Generate Analytical Model | Included | Maintained |
| Identify the THR in the Analytical Model | Included | Removed |
| Identify the Thresholds | It did not exist in this version | Joining Steps 4 and 6 from CATS Design into one |
| Identify the Test Scenarios | It did not exist in this version | Included |
| Describe the test oracle | Included | Maintained |
| Describe the Test Cases | Included | Maintained with an evolved template |
| Package the Test Suite | Included | Removed |

*Table 19. Wildlife Tracker test template*

After creating CATS# v1, we used the application described in 4.2.2 to evaluate the process itself and the proposed enhancements.

The performed evaluation is limited. Nevertheless, it represents a proof of concept to analyze the proposed process qualitatively and implicitly. Additionally, there were undeniable threats, such as the researcher's natural interest in the outcome and the biases that come with it.

However, the main focus of this first internal evaluation is to observe how much the proposed process would be adequate (or not) to model CASS test cases. Furthermore, a scenario found in the literature was used to analyze the suitability of CATS# version 1 while looking for opportunities for improvements, and we learned from it (as it is the goal of the Learn, Adapt/Build and Measure methodology). Below is what we could conclude after learning with the evaluation:

1. It is possible to specify the analytical model after the two initial steps
2. The analytical model can help the design of the conceptual model
3. The conceptual model can be used to represent the context situations and variations to help the software engineer to understand better the necessary test cases

The entire CATS# v1 process was evolved considering these issues. Its second version is presented in the next section.

## 4.3 The second version of CATS#

### *4.3.1 The CATS# v2 process*

After the first version of CATS# and its internal evaluation, we measured, observed, and learned. As mentioned in the methodology from section 4.1, the next step was adapting. Therefore, we updated the process, and the proposed modifications were included in the second version of CATS#, presented in this section. The major modification to this new version was made on the conceptual model.

The conceptual model from CATS Design was inspired in other domains and had the main goal of helping the identification of thresholds. This goal was maintained in CATS# v1. However, this second version includes a conceptual modal to identify thresholds and help the software testers understand the entire software system behavior regarding the context variations.

To do this, we used the concept of context situations presented in Chapter 2. A threshold is still represented and triggers the variation from one situation to another. User events are also represented as triggers. The main idea is that each context variation, represented by an arrow in the model, is a potential test case.

Additionally, we added a matrix to map the software system functionalities across different context situations. Again, the goal is to help the software tester identify what functionality will be affected by the context variation.

Figure 18 shows the second version of CATS#. Again, the modified steps are presented in gray, the new ones are in bold, and the ones that remain the same are in white. More information about the changes made is presented below.

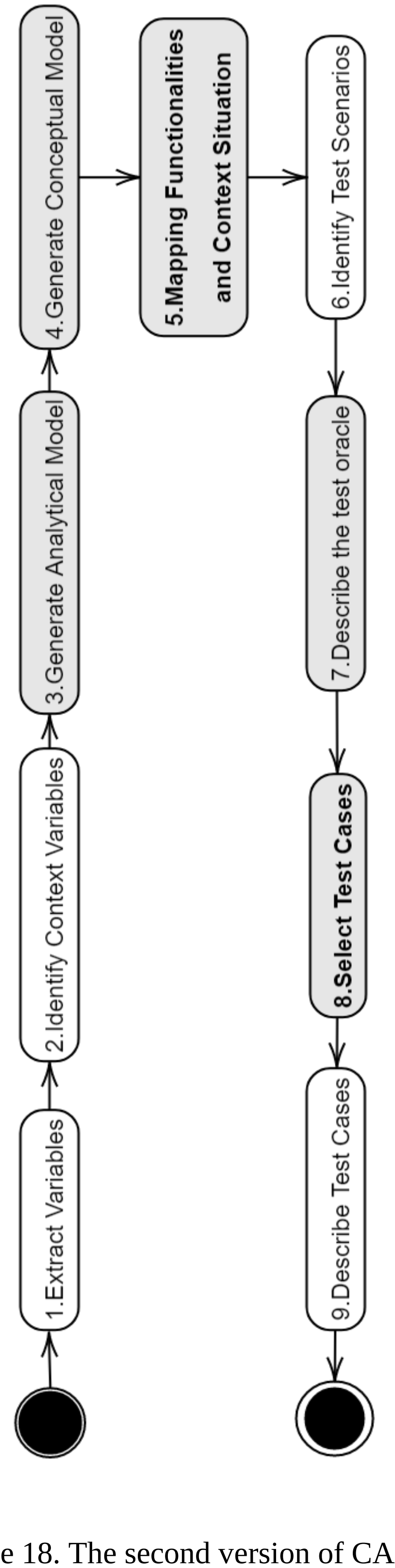

1.Extract Variables
2.Identify Context Variables
3.Generate Analytical Model
4.Generate Conceptual Model
5.Mapping Functionalities and Context Situation
6.Identify Test Scenarios
7.Describe the test oracle
8.Select Test Cases
9.Describe Test Cases

Figure 18. The second version of CATS#

### 1. Extract Variables

This step remains the same as in CATS# v1.

### 2. Identify Context Variables

This step remains the same as in CATS# v1.

### 3. Generate Analytical Model

This step was slightly modified when compared to CATS# v1. The Analytical Model presented in section 4.2.1 has two fields that must be filled out: Context-Variable and Effect. This model was updated to three fields (Context-Variable, Threshold, and Effect) to fill the information.  Although the information captured by the updated model will not change, the idea is to improve the organization and draw the software testers' attention to every piece of information captured in this step. Table 20 shows an example using the Restaurant application from section 3.5.3

| Context-Variable | THR | Effect |
|---|---|---|
| GPS Availability | TRUE | Automatically gets the user's location |
| | FALSE | Ask the user's location manually |
| Internet Connection | TRUE | Returns to Restaurants nearby |
| | FALSE | Not Defined |

*Table 20.CATS# Conceptual Model - version 2 with three different columns to improve organization*

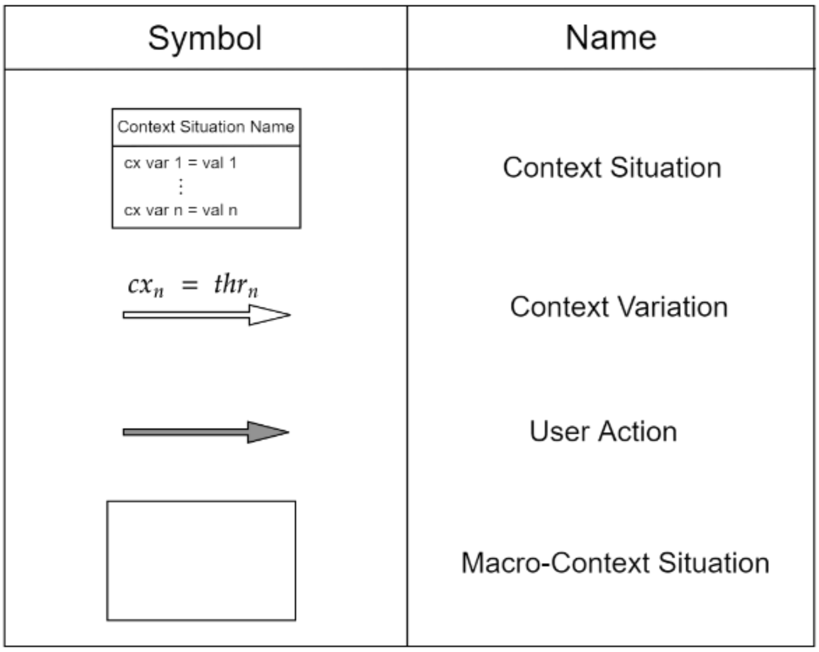
Besides this modification, creating the Analytical Model becomes the third step of the CATS# v2 process, happening before designing the Conceptual Model. Therefore, the choice to change the orders between the designing of the two models was motivated by the fact that creating the Analytical Model is possible before steps one and two of the CATS# v1 process. Additionally, as explained in the next step, it can help the software tester create the conceptual model, which will become more complex.

### 4. Generate Conceptual Model

The conceptual model represents only a small step in the previous processes versions (CATS Design and CATS# v1). However, in this version, it becomes the central part of the CATS# process. Therefore, every step/activity previously executed had the intention of helping to design the conceptual model. Likewise, every step/activity executed later is guided by it.

In CATS# v2, the conceptual model intends to represent the software system behavior using the concepts of context situations and variations. The context situation is characterized by the relevant context variables and their values. A variation represents either a context variable reaching its threshold (white arrow) or a user action (gray arrow). Additionally, each white arrow has a tag describing the threshold value triggering a specific variation. Sometimes, this value includes only one CV, but in some cases, the threshold will be characterized by a combination of CVs indicated by *.AND.* operations.

Usually, *.OR.* operations are unnecessary because, as far as it could be investigated, an *.OR.* operation would represent different cases of variations, which the same arrow will not represent. Figure 19 shows the graphical representation of the CATS# v2 Conceptual Model Elements.



| Symbol | Name |
|---|---|
| Context Situation Name<br>cx var 1 = val 1<br>⋮<br>cx var n = val n | Context Situation |
| $cx_n = thr_n$<br>⇨ | Context Variation |
| ➡ | User Action |
| □ | Macro-Context Situation |

*Figure 19. CATS# v2 - Conceptual Model Elements*

Additional to context situations and variations, the concept of macro-context situations was used as a high-level representation of a specific set of context situations. The situations represented inside a macro-context situation will be affected by a context variation the same way, which allows them to be represented together.

It is possible to represent the entire model through context situations and variations, but the software system's complexity may generate a graphically confused representation. In this case, macro-context situations can help since they represent a high-level abstraction of the software system.

To exemplify the use of the conceptual model elements, Figure 20 shows a small example modeling the Analytical Model previously presented in this section (Table 20). Figure 20 presents three Context-Situations (Get User location Automatically, Ask User's location and Not Defined) as well as a macro-context situation (Return to Restaurants nearby). Both context-situations inside the macro-context will be affected in the same way by the Internet Connection becoming unavailable. Consequently, they can be represented by a Macro-context Situation as well.

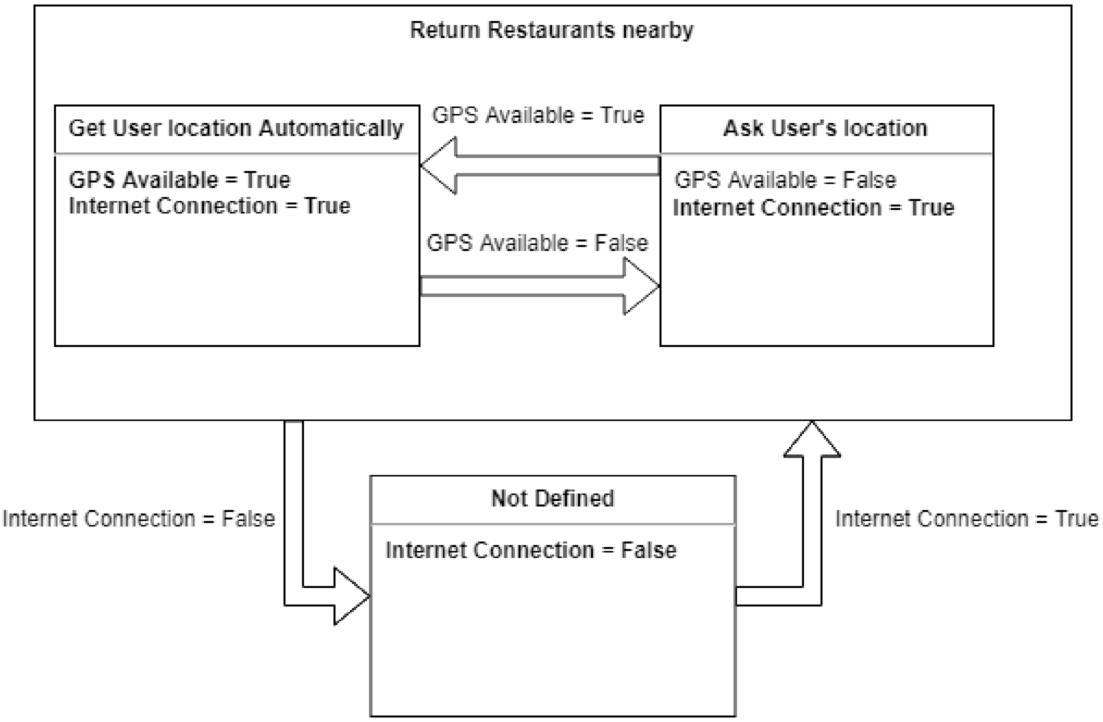

Return Restaurants nearby
Get User location Automatically
GPS Available = True
Internet Connection = True
GPS Available = True
GPS Available = False
Ask User's location
GPS Available = False
Internet Connection = True
Internet Connection = False
Not Defined
Internet Connection = False
Internet Connection = True

*Figure 20. Conceptual Model from the Restaurant Application*

A complete example of how context and macro context situations can be combined will be presented in section 4.3.2, where the internal evaluation of CATS# v2 is presented.

Although the Conceptual Model can be compared with a State Diagram, they represent different levels of abstractions: A State Diagram represents a set of application states [25], and the Conceptual Model will represent a set of context situations and their variations.

Also, the crucial detail about the proposed conceptual model is that each white arrow represents a potential test case that could be explored depending on the testing strategies. Possible testing strategies would be specifying test cases for all the transitions among context and macro-context situations, specifying at least one test case for each context variation, and specifying test cases for the most critical part of the software system.

### 5. Mapping Functionalities and Context Situations

This new step complements the conceptual model to map the system's functionalities with the context situations.

While the Conceptual Model can represent the identified context situations and their variations, it does not show how the context variation will affect the software system's functionalities. Therefore, a situation matrix S has been created to support this representation. Its rows represent the functionalities, and the columns represent the context situations.

$$S_{f,c} = \begin{cases} True & \text{If the context situation } c \text{ enables the functionality} f \\ False & \text{Otherwise} \end{cases}$$

*Equation 1. Situation Matrix function*

Equation 1 shows the function that should be used to fulfill the matrix. For example, if the context situation *c* enables the functionality *f* and the context situation *c'* disables it, the values of each row of the matrix will be:

$$S_{f,c} = True\ and\ S_{f,c'} = False$$

*Equation 2. Example of values for some rows from Situation Matrix*

Therefore, if an event promotes the variation of context situation *c* to context situation *c'*, it will affect functionality *f*. An expected behavior must be specified at the next step because of context variation.

### 6. Identify the Test Scenarios

This step remains the same as in CATS# v1.

### 7. Describe the test oracle

In this version, the main difference from CATS# v1 is the inclusion of the System Feature field in the oracle to describe which functionality should be affected by the specified variation on that test case. All other parts are like CATS# v1. Table 21 shows an example.

Additionally, the test cases from the Conceptual Model (indicated by arrows) should be listed while combined with the Situation Matrix from step 5.

| TC Id | System Feature | Input | Conditions | Expected Result |
|---|---|---|---|---|
| TC01 | Search for Restaurants | User's location | GPS available = True → GPS available = False | Ask location manually |
| TC02 | Search for Restaurants | User's location | Int. Connection = True → Int Connection = False | Not specified |

*Table 21. CATS# v2 Test Oracle*

### 8. Select Test Cases

After listing all possible combinations of test cases and software system features, a set of specific test cases should be selected at this step. The testing strategy used should be considered to select the test cases. In an ideal scenario, all test cases listed in step 7 must be specified and executed. However, in situations with limited resources, a small but meaningful set of Test Cases could be selected from the list and specified.

### 9. Describe the test case

This step remains the same as in CATS# v1.

### *4.3.2 Internal Evaluation*

In section 4.2.2, a small application was used to evaluate CATS# v1. Then, after evolving the process, another internal evaluation was conducted with the same objective. However, while the first one (Wildlife tracker) was a small application found during the Structured Review, the second software system is a real-life application called Parasite Watch.

The Parasite Watch is a software system for supporting the diagnosis of parasitic diseases which captures images of biological samples obtained from patients and uses an image recognition algorithm to make a diagnostic suggestion. As parasitic diseases are a substantial problem in developing countries, some information (as the location of each image) should be collected to help governments with public health politics. Therefore, although it is not purely a safety-critical system, it is a real project, based on real necessities, planned to adapt according to the different usage situations [26][27].

The Parasite Watch must adapt to operate in different environments, such as no network connection or power availability (in this case, a battery and solar panel must be used). There are two modes of operations when considering network availability:

- **Local lab**: there is no network connection available. In this case, the diagnosis must be made locally, and the host must store the images. Some disadvantages of this mode of operation are the restricted space for image storage and outdated image recognition algorithms. The software system automatically transfers the files to solve the limited memory space when a memory drive is detected. If there is no space in the memory left, the system shows a message to the user and pauses its operation until it is solved.
- **Online Lab**: there is a network connection available. In this case, the system must send the captured images to an Online server and receive the diagnosis. The Online server will contain the latest version of the image recognition algorithm and the database used for training the algorithm. When executed at this mode, the system must automatically update the local lab algorithm (if not updated yet) and synchronize the images stored locally with the Online server database.

Additionally, when considering the power availability, there are three modes of operations:

- **Electricity is available**: All functionalities are available as well
- **The software system uses the battery or solar panel**: Some functionalities must be disabled, such as the Software System Update, Synchronization, and Transferring Files to Memory Drive.
- **The software system uses a battery that is running low:** In this case, it should operate at Energy Saving Mode, which means disabling the network connection (whether it is available) and GPS location.

The Parasite Watch Specification Document was used to specify the test cases. However, since it was not written considering the context perspective, some information could not be found. Consequently, it was necessary to act like stakeholders to fill in the blanks. The information missing in the documentation was added later in the Analytical Model and is represented differently, with an underline.

This gap in the original Parasite Watch documentation was not noticed before the application of CATS# v2 and made us realize that CATS# v2 could also work as an inspection technique focusing on the context perspective. Therefore, as a side effect of applying CATS# v2, the quality of the specification document improved.

The CATS# v2 was applied to specify the test cases for this software system according to the steps below.

### 1. Extract Variables

At this first step, the goal is to extract all the relevant variables from the Parasite Watch Specification Document. Table 22 shows the identified variables. Besides, it was possible to identify one missing variable in the documentation, which was included later in Table 22 and is underlined.

### 2. Identify Context Variables

Since the idea of this step is to highlight the CVs, they are represented in bold in Table 22.

| Variables List |
|---|
| • **Internet Available**<br>• Diagnosis<br>• **Power Available**<br>• Updated<br>• **USB Device**<br>• **Battery Level**<br>• Geolocation<br>• Date<br>• Synchronized<br>• Time<br>• Image<br>• Point of Interest<br>• **Memory Available** |

*Table 22. Step 1 and 2 from CATS# v2*

### 3. Generate the Analytical Model

Table 23 shows the Analytical Model describing the Parasite Watch modes of operations using Context Variables and their thresholds.

When the Internet is available, the Parasite Watch will operate in Online Lab mode, enabling the functionality of synchronization and updating. Conversely, when the Internet becomes unavailable, the system should change to Local Lab, interrupting any Online Lab operation.

Considering whether the Power is available, some functionalities should be disabled when executing in battery mode (Power Available = False, Battery Level > 20%), and the software system must function with more restrictions when the system is running out of battery (Energy Saving Mode).

Also, the expected behaviors are briefly described when the memory becomes unavailable and when a memory drive is detected.

It is worth mentioning that some situations will rely on a combination of more than one CV. For instance, when Internet Available is True and Updated is False, the system must update the software if Power Available is True.

| CV | THR | Effect |
|---|---|---|
| Internet Available | True | Use Online Lab to make the diagnosis<br>If Synchronized = False, sync diagnosis<br>If Updated = False, update de Software |
| | False | Use Local Lab to make the diagnosis<br>If Synchronized = False, stop sync<br>If Updated = False, stop updating |
| Power Available | True | Use energy from Power |
| | False | Use energy from Battery or Solar Panel<br>If Synchronized = False, do not sync<br>If Updated = False, do not update<br>If USB Dev. = True, do not transfer files |
| Battery Level | > 20% | If Internet Av. = True: Online Lab Mode<br>If Internet Av. = False: Local Lab Mode |
| | ≤ 20% | Disable GPS<br>Disable Internet |
| Memory. Available | False | Display a message and pause the system |
| Memory Drive | True | Transfer files to Memory Drive |

*Table 23. Analytical Model*

### 4. Generate the Conceptual Model

The Parasite Watch software system model has seven context situations and variations, presented in Figures 21, 22, 23, and 24. In Figure 21, the software system was modeled without macro-context situations, resulting in a confused and incomplete model version. For example, there should be an arrow between the “System Paused” context-situation and “Sending Files Online,” but there is no more space to draw left in Figure 21.

In Figures 22, 23, and 24, some of the context situations were organized in two macro-context situations, representing a high-level view of the software system, and resulting in a more organized model.

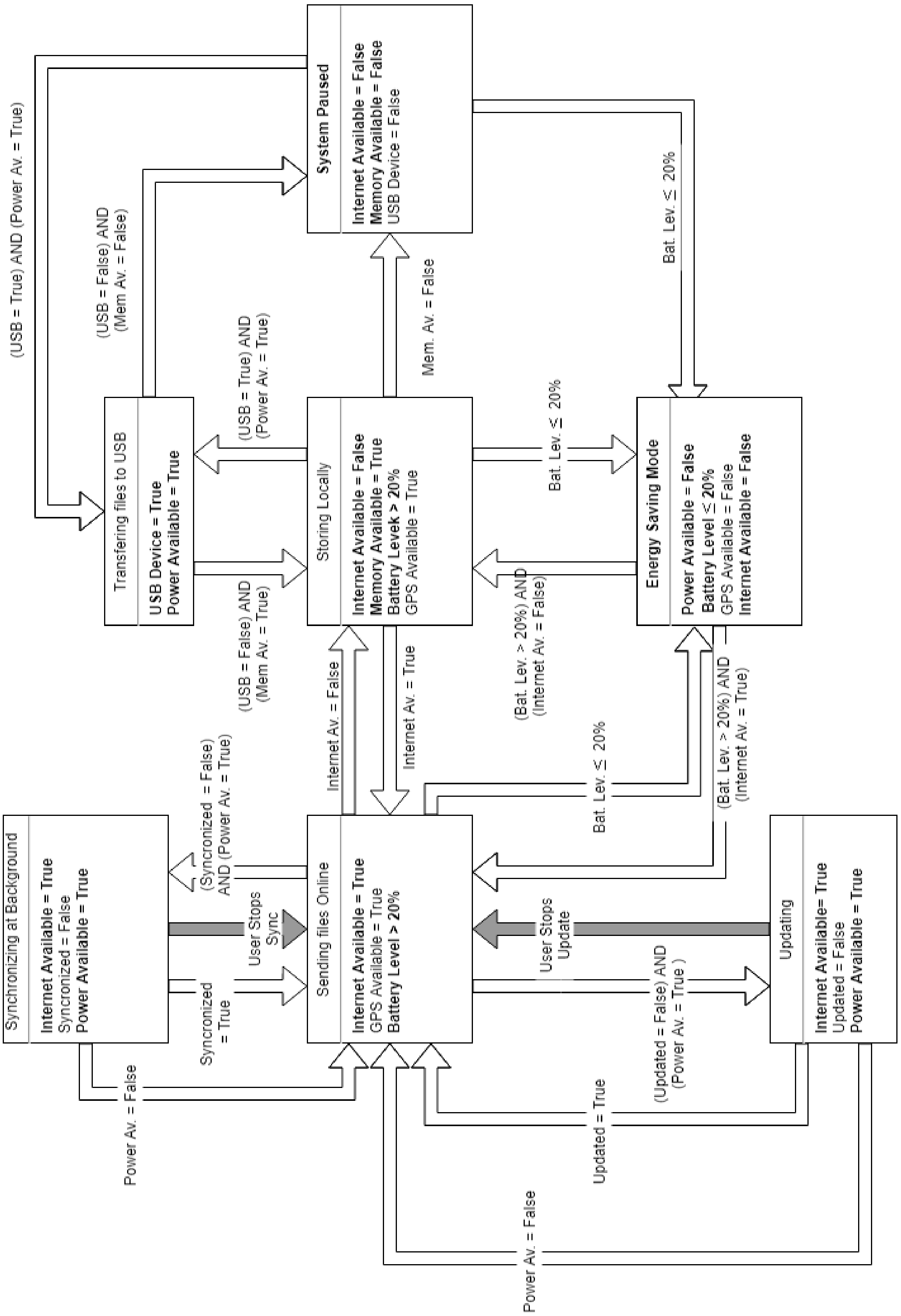


*Figure 21. Conceptual Model without macro-context situations*

As it is possible to see in Figures 21, 22, 23, and 24, each context variation has a tag showing what threshold value causes that modification. Some variations are composed of *.AND.* operations. When a combination of variables varies when the change happens, the other must be set previously at that value. The variations among macro-contexts can also affect the context situations inside them. In this way, if the software system is in the "Sending Files Online" situation and the battery level reaches 20%, it must enter energy-saving mode (Figures 21, 23, and 24), when the Internet Connection and GPS geolocation will be disabled.

Notice that all context situations happening inside a macro-context situation have their characteristics. For example, all context situations inside Online Lab have Internet available. According to this model, the Parasite Watch software system will have at least 17 test cases, considering the variations between context and macro-context situations.

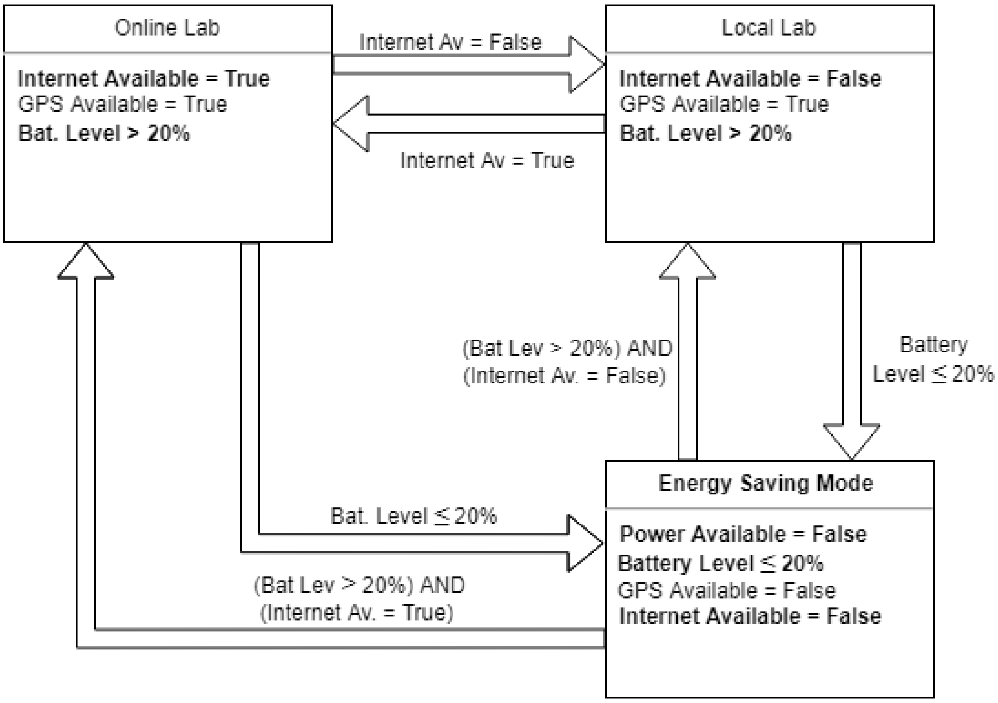


*Figure 22.Conceptual Model using a macro-context situation – part 1*

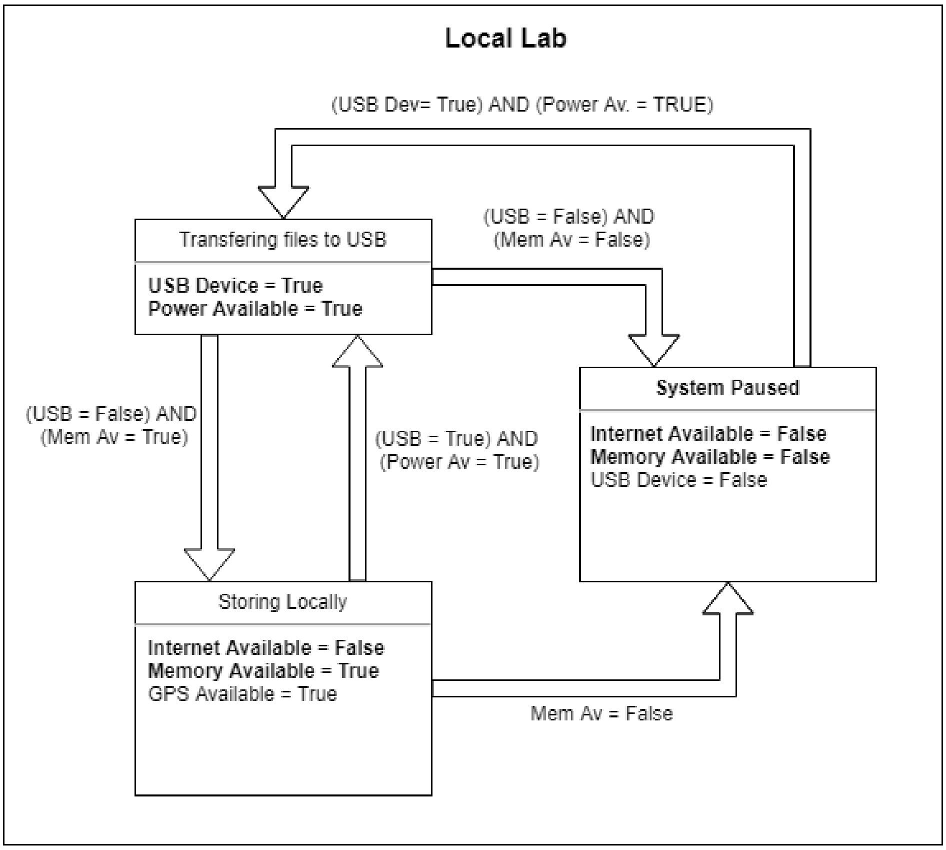


*Figure 23. Conceptual Model using a macro-context situation – part 2*

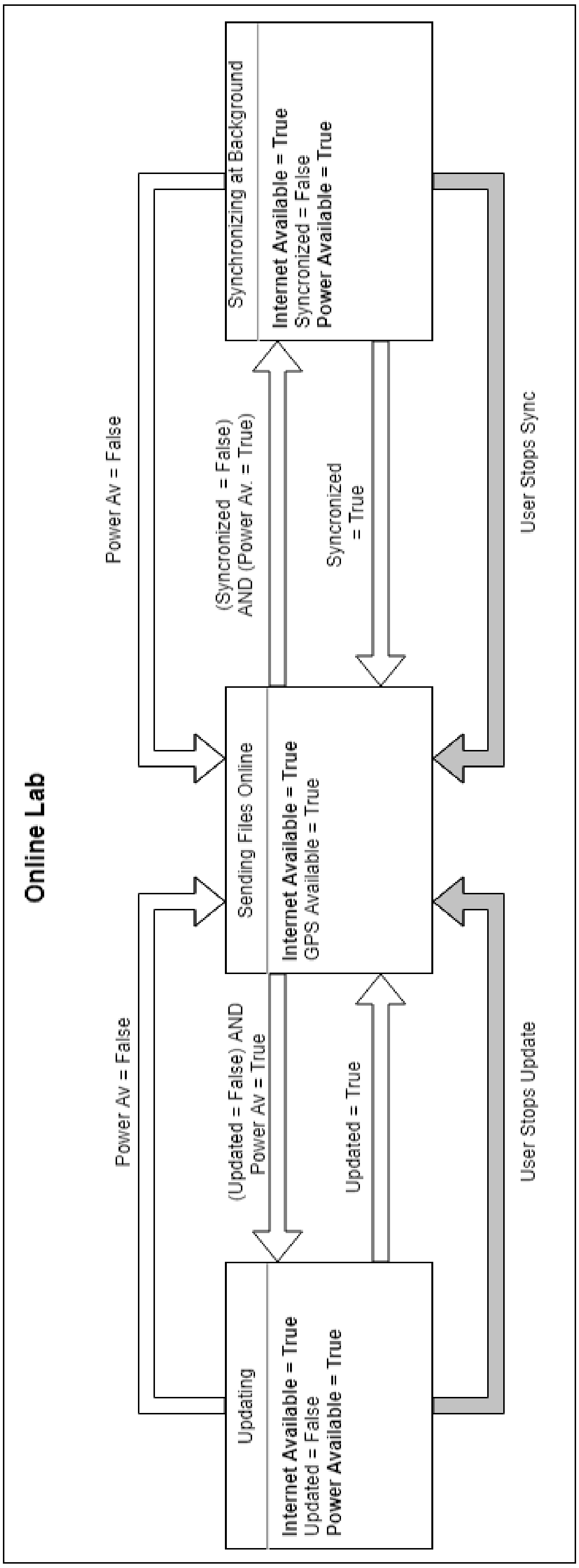


*Figure 24. Conceptual Model using a macro-context situation – part 3*

### 5. Mapping Functionalities and Context Situations

The Situations Matrices for Parasite Watch are presented in Tables 24, 25, and 26. The matrices in Tables 24 and 25 show the functionalities affected by the context variation inside their macro context situations (Local and Online Lab, respectively). Table 26 maps the same functionalities considering the variation among macro-context situations.

When functionality is enabled in a specific context situation and then disabled during a variation to another context situation, the software system must have an expected behavior because it can occur while the functionality is executing.

An example would be when the Internet becomes unavailable during the execution of "Submit to Online Diagnosis." This functionality will be affected (at least interrupted), and how the software system should behave must be specified. In the case of the previous example, it has been determined that the expected behavior would be submitting the diagnosis again to the local lab automatically.

Although the matrix does not specify the expected behavior, this will describe the oracle.

| **System Features** | **Transferring files to USB** | **System Paused** | **Storing Locally** |
|---|---|---|---|
| Capture Image | False | False | True |
| Submit to Local Diagnosis | False | False | True |
| View Diagnosis | False | False | True |
| Transfer to Mem. Drive | True | False | False |

*Table 24. Local lab Situation Matrix for Parasite Watch*

| System Features | Updating | Sending Files Online | Synchronizing at Background |
|---|---|---|---|
| Capture Image | False | True | True |
| Submit to Online Diagnosis | False | True | True |
| View Diagnosis | False | True | True |
| Update Software | True | False | False |
| Sync with the database | False | False | True |

*Table 25. Online lab Situation Matrix Parasite Watch*

| System Features | Local Lab | Online Lab | Energy Saving Mode |
|---|---|---|---|
| Capture Image | True | True | True |
| Submit to Local Diagnosis | True | False | True |
| Submit to Online Diagnosis | False | True | False |
| View Diagnosis | True | True | True |
| Capture Geolocation | True | True | False |
| Enable Local Lab's Features | True | False | False |
| Enable Online Lab's Features | False | True | False |

*Table 26. Situation Matrix Macro-context Situations for Parasite Watch*

### 6. Identifying the Test Situations

The CASS Test Case Model was used to classify the test scenarios at this step. Most of the cases regarding the context variation are of CASS Test Case Model B, and the necessary variation of context in conditions is presented in Table 27.

### 7. Describe the test oracle

The entire list of test cases for the Parasite Watch software system comprises 33 test cases. Table 27 shows a sample of the test case list. The complete specification is available in Appendix B.

As was mentioned before, in this version of CATS#, the list captures the test cases with the variation of context and the functionality in which the test case must be applied.

| Id | System Feature | Input (I) | Conditions (C) | Expected Result (E) |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 01 | Capture Geolocation | User’s location | Bat. Level > 20% → Bat. Level ≤ 20% | Dis. GPS (Energy Saving Mode) |
| 04 | Capture Image | Image | Mem. Av. = True → Mem. Av. = False | System Paused |
| 15 | Submit to Online Diagnosis | Image | Internet Av. = True → Internet Av. False | Submit to Local Lab |
| 29 | Update Software | | Power Av. = True → Power Av. = False; Updated = False | Interrupt Updating |

*Table 27. Parasite Watch Test Oracle*

**8. Select the test case**

Since the purpose of this proof of concept was to evaluate the modifications at the CATS# process, only three test cases, specified as CASS Test Case Model B, were chosen: TC01, TC15, and TC29. They were specified using the CATS# v2 Testing template in Tables 28, 29, and 30.

**9. Describes the test case**

Finally, Tables 28, 29, and 30 present the test cases using the CATS# v2 template. The main goal of the template is to specify which context variables must remain constant and which must vary during the test case execution to verify whether the software system behaves as expected.

| **Id** | TC01 |
|---|---|
| **Test Objective** | To test the "Capturing Geolocation" Functionality from Local Lab |
| **Preconditions** | • GPS Available = True<br>• Battery Level > 20% |
| **Fixed Conditions** | • Internet Available = False<br>• Power Available = False |
| **Input** | User's location |
| **Test Steps** | 1. Register Blade (Id and Date)<br>2. Open Camera<br>3. See Image (c1)<br>4. Capture Image |
| **Varying Conditions** | c1. Bat. Level > 20% → Bat. Level ≤ 20% |
| **Expected Result** | Capture the last location available and disable GPS (Enter in Energy Saving Mode) |
| **Pos Conditions** | • GPS Available = False<br>• Battery Level ≤ 20% |

*Table 28. Test Template for TC01*

| **Id** | TC15 |
|---|---|
| **Test Objective** | Verify the Internet becoming unavailable while executing the functionality "Submit to Online Diagnosis." |
| **Preconditions** | • Internet Available = True |
| **Fixed Conditions** | • Power Available = True<br>• Memory Available = True |
| **Input** | Image |
| **Test Steps** | 1. Register Blade (Id and Date)<br>2. Capture Image<br>3. Submit to Online Diagnosis (c1)<br>4. Open Diagnosis |
| **Varying Conditions** | c1. Internet Available = True → Internet Available = False |
| **Expected Result** | Receive diagnosis from Local Lab |
| **Pos Conditions** | • Internet Available = False |

*Table 29. Test Template for TC15*

| **Id** | TC29 |
|---|---|
| **Test Objective** | Verify the Power becoming unavailable while updating software |
| **Preconditions** | • Updated = False<br>• Power Available = True |
| **Fixed Conditions** | • Internet Available = True |
| **Input** | |
| **Test Steps** | 1. Enter Updating mode<br>2. While Updating (c1) |
| **Varying Conditions** | c1. Power Availability = True → Power Availability = False |
| **Expected Result** | Interrupt Updating |
| **Pos Conditions** | • Internet Available = False |

*Table 30. Test Template for TC29*

### *4.3.3 Discussion*

Applying CATS# v2 in the Parasite Watch software system made it easier to understand the context (which is something abstract, as mentioned earlier) and its variation through context variables and context situations, making it more feasible to plan a testing strategy.

Although it is presented in Chapter 2 as part of our basic definitions, the main idea of using context situations as a set of context variables arises during this second trial of CATS#. Additionally, the Parasite Watch software system confirmed our initial assumption about the need for an enhanced test case model for CASS and the use of macro-context situations to improve organization in case of a more complex model.

Additionally, the Situation matrix comes to map System features and Context-Situations to complement the rationale, so it is possible to observe which functionalities will be affected by the context variation.

Table 31 presents a comparison to understand better the differences between CATS# v1 and v2. It is possible to observe that the core of the process remains the same.

However, some improvements were done such as the evolution of the Conceptual Model and the inclusion of the Situation Matrix to map functionalities and context situations.

| CATS# versions | **First Version** | **Second Version** |
|---|---|---|
| Extract Variables | Included | Included |
| Identify Context Variables | Included | Included |
| Generate Conceptual Model | At Step 3 | An evolved version<br>At Step 4 |
| Generate Analytical Model | At Step 4 | At Step 3 |
| Mapping Functionalities and Context Situations | Absent from this version | Included |
| Identify the Thresholds | Included | Absent from this version |
| Identify the Test Scenarios | Included | Included |
| Describe the test oracle | Included | An evolved version |
| Select Test Cases | Absent from this version | Included |
| Describe the test case | Included | Included |

*Table 31. Comparison between versions of CATS#*

As threats to validity, it is possible to mention the fact that Parasite Watch is not purely a safety-critical and context-aware system, which can limit how much it is possible to generalize the observed outcome. Additionally, this internal evaluation was also performed by the researchers. Therefore, the experience acquired during the research process and the first internal evaluation should also be considered.

As a restriction, we need to mention that the Parasite Watch software system is completely specified but not completely developed. Hence, we could not execute the test case we designed during this trial.

However, since Parasite Watch is a more complex system based on real-world necessities and with some context-aware functionalities, it gave us a shred of initial indication of CATS# v2 feasibility. Furthermore, it allowed us to realize we were almost getting into a stable version. Small modifications and adjusts were done between this version of CATS# and the final one. The final version of CAT# will be presented in the next chapter. Additionally, chapter 6 presents its application in real life, but by people not directly involved in its creation and evolution.

## 4.4 Chapter Considerations

This chapter presented the evolution of CATS Design (described in section 2.5) through the final version of CATS# (which will be described in the next chapter). We adopted a Learn, Adapt/Build, and Measure methodology to evolve the process. First, we learn from CATS Design to create the first version of CATS#. Then, an internal evaluation was conducted using the Wildlife Tracker application from Afanasov, Mottola, and Ghezzi [23].

After the first evaluation results, some improvements were noticed, and the second version of CATS# was proposed and evaluated using the Parasite Watch software system.

Both evaluations performed were proofs of concept performed by the researcher. Thus, although there were undeniable threats in this experimental setting, the goal, which was to observe the CATS# process while looking for opportunities for improvements, was achieved.

After these two evaluation trials to evolve CATS# and based on the results obtained in the second internal evaluation (Parasite Watch software system), it is time to move forward. The final version of CATS#, presented in the next chapter, consolidates the previous versions.

The main difference between CATS# v2 and its final version is that step 5 becomes non-mandatory. This decision was based on the fact that, even if mapping functionalities and context situations are useful in the case of more complex systems, it is unneeded in the case of less complex systems. Therefore, the user can decide whether to use it. Besides this, some adjustments were made in the templates and terminology, as will be possible to observe in the next chapter.

Additionally, Chapter 6 will present the evaluation in a real-life scenario while being applied by other people.

# 5 CATS# - Final version

## 5.1 Introduction

Chapter 2 presented CATS Design, our starting point, and Chapter 4 presented all the evolution between CATS Design until the second version of CATS#. This one will present the final version of CATS#.

Since we realized, with Parasite Watch, that we were getting closer to a stable version, some small modifications were done in CATS# version two to prepare the application to be evaluated by people not involved in the project. Therefore, while the evaluation is described in Chapter 6, this Chapter presents the CATS# process completely, step by step, including the ones maintained from CATS Design or created during the CATS# version 1.

This time is not about evolving the process anymore (as in sections 4.2.1 and 4.3.2.). Instead, it is about describing the final version of CATS#. Hence, the next section will present all the needed information for everyone interested in using it.

## 5.2 CATS# - Final Version

The CATS# final process is presented in Figure 25. It comprises nine steps: one optional (dashed line) and eight mandatories. First, for specifying test cases using CATS#, it is necessary to have a document describing the software system behavior. There is no restriction about the document type (uses cases, scenarios, detailed application description), but it must exist. Once it exists, the software engineer should follow the steps below.

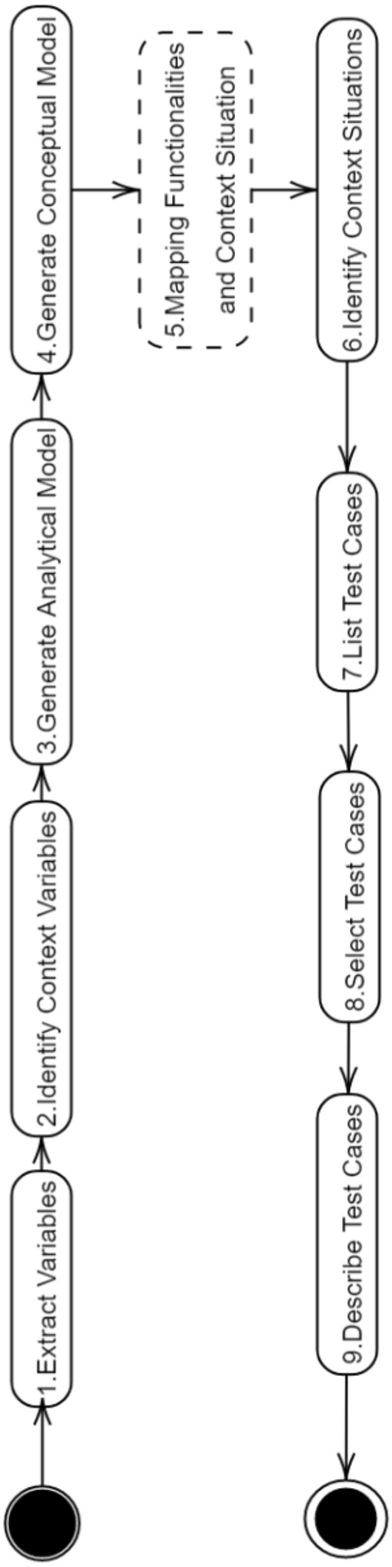

1.Extract Variables
2.Identify Context Variables
3.Generate Analytical Model
4.Generate Conceptual Model
5.Mapping Functionalities and Context Situation
6.Identify Context Situations
7.List Test Cases
8.Select Test Cases
9.Describe Test Cases

*Figure 25. CATS# final version*

### 1. Extract Variables

In this step, it is necessary to go through the documentation to extract every variable representing or influencing the system's behavior. Both context and conventional variables should be listed. The rationale is that the variables that do not represent the context itself could be affected by its variation. Therefore, they can help analyze how the context affects the software system's behavior.

### 2. Identify Context-Variables

After having all variables, it is time to identify the ones representing the context. In this step, it is not necessary to create a new list. Instead, the idea is that using the list from step 1, the user marks the context variables with any graphical element (such as underlining, highlighting, and so on) that differentiates them from the regular ones.

### 3. Generate Analytical Model

An analytical model is a tool used to represent specific aspects of the software system based on observing and analyzing its characteristics and behaviors.  In our case, this analysis will rely on the documentation describing the software system behavior and the context variables identified in the previous step.

CATS# analytical model captures the characteristics and behaviors through three elements: the context variables, the threshold values, and the expected result. Table 32 shows the template. First, it is necessary to fill the template with the Context Variables from step 2. After that, the threshold values must be identified and listed accordingly. The threshold is a value reached by a specific CV that will trigger the variation from one context situation to another. Once this change is triggered, a specific behavior/result is expected.

The model from Table 32 has the goal of helping the user identify these elements and the relationship among them.

| Context Variable | Threshold | Expected result |
|---|---|---|
| $< cv_0 >$ | $> thr_0$ | $< e_1 >$ |
| | $\leq thr_1$ | |
| $< cv_1 >$ | $= thr_2$ | $< e_0 >$ |
| | $= thr_3$ | |

*Table 32. Analytical Model Template*

After describing this model, it will be used to design the Conceptual model in the next step.

### 4. Generate Conceptual Model

Usually, textual descriptions are quite acceptable for conventional software systems considering that all information regarding the testing (input, conditions, results) remains static from the moment an input is submitted to the software system until the moment it deploys the results.

However, when dealing with the context and its variation, textual descriptions can make representing different situations and configurations challenging. Therefore, using a more systematic and precise specification can increase the understanding of the software system execution [25]. Considering this, we used a conceptual model to support the understanding and communication of testing scenarios.

A conceptual model describes some aspects of the software system behavior through concepts and relationships. For example, the CATS# Conceptual Model captures the testing scenarios using two elements: context situations and variations.

Relevant context variables and their values characterize the context situation. A variation represents either a context variable reaching its threshold (white arrow) or a user action (gray arrow). Figure 26 shows their graphical representation.

Each white arrow has a tag describing what threshold value will trigger a specific variation. This variation can depend on just one CV or be characterized by a combination of CVs indicated by an *.AND.* operation. *.OR.* operations will not be necessary since they would represent different cases of variations which, in this case, is represented by different arrows. Every arrow in this model, especially the white ones, will be a potential test case since it characterizes a context variation.

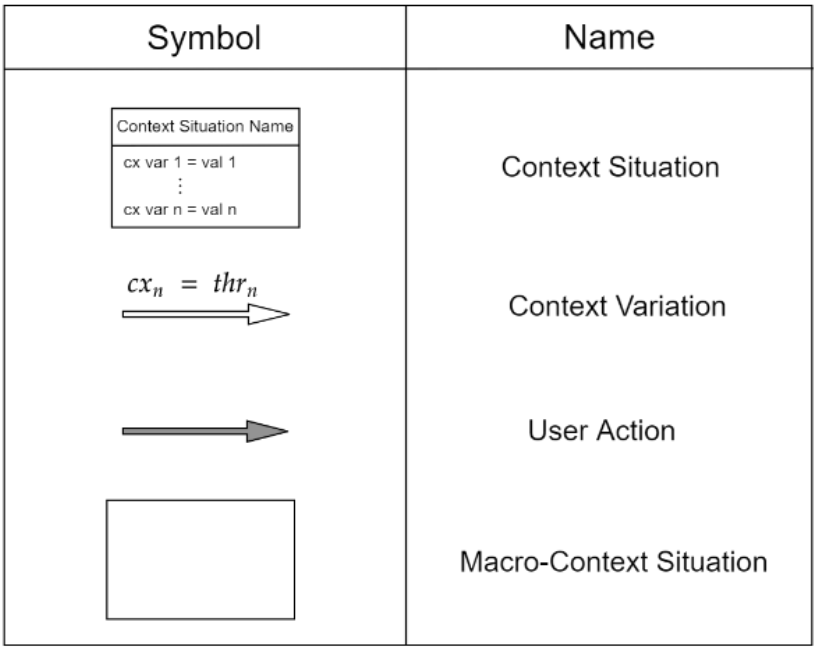


*Figure 26. Conceptual Model elements*

The conceptual model is the core of CATS# since it will guide potential test cases. Therefore, every step previously executed had the intention of helping the conceptual model design. Likewise, every step executed after it follows his guidance. Hence, it is important to have a very organized model. For that reason, a third element was incorporated into the model, the macro-context situations (also presented in Figure 26), a higher-level representation of a set of context situations having the same variations to another context situation (macro or not).

It is possible to design an entire model without using a macro-context situation. However, a very complex software system can result in a messy and incomplete model. For these cases, having a higher-level representation will help.

Figure 27 shows an example/template of a conceptual model designed based on the analytical model template from Table 32. It has three context situations with their respective threshold.

More information about the conceptual model can be found in section 4.3.1.

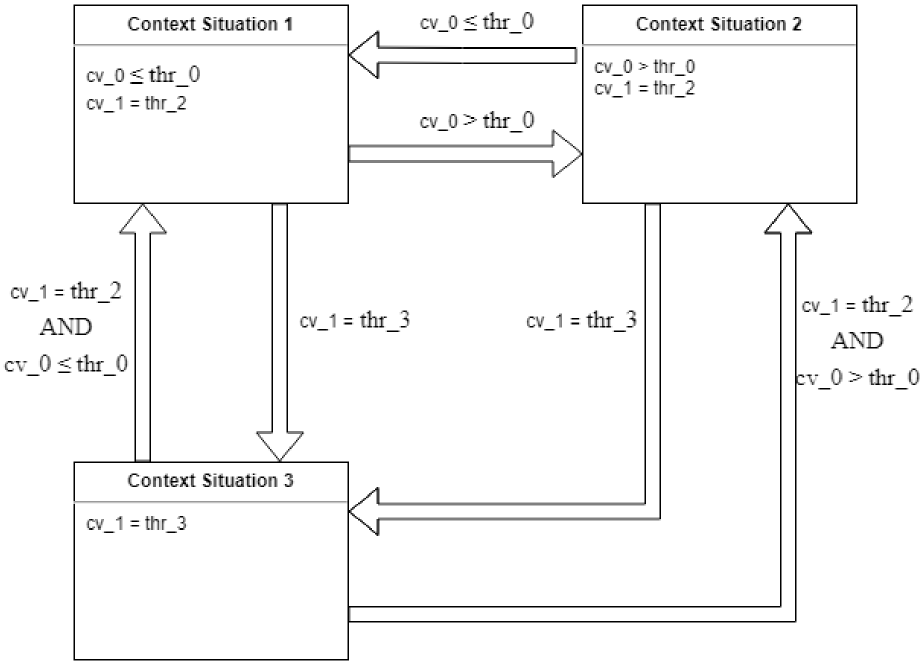


*Figure 27. Conceptual Model example*

### 5. Mapping functionalities and Context Situations

Both analytical and conceptual models from steps 3 and 4 represent context situations and their variation. However, they do not show how the variation will affect the system functionalities. Depending on the functionality being executed, the context variation can affect (or not) the system behavior. Therefore, a situations Matrix S has been created to map this relationship between context-situations and functionalities.

$$S_{f,c} = \begin{cases} True & \text{If the context situation } c \text{ enables the functionality} f \\ False & \text{Otherwise} \end{cases}$$

*Equation 3. Situation Matrix Function*

Equation 3 shows how the matrix must be fulfilled. If functionality $f_0$ is enabled in the context situation $c_0$, the value of the $\mathrm{S}_{\mathrm{f}_0,\mathrm{c}_0} = \mathrm{True}$. If the same functionality is not enabled in a context situation $c_1$, then the value of $S_{f_0,c_1} = False$ . The matrix template (filled accordingly to the previous example) is displayed in Table 33.

|  | Context Situations |  |  |
|---|---|---|---|
| Functionalities | <$c_0$> | <$c_1$> | <$c_2$> |
| <$f_0$> | True | False |  |
| <$f_1$> |  |  |  |
| <$f_2$> |  |  |  |

*Table 33. Situation Matrix example*

This step is the only optional from the CATS#. However, although the software engineer can skip this step for a small system, it is highly recommended to guarantee that all the relevant functionalities will be tested in the case of complex systems.

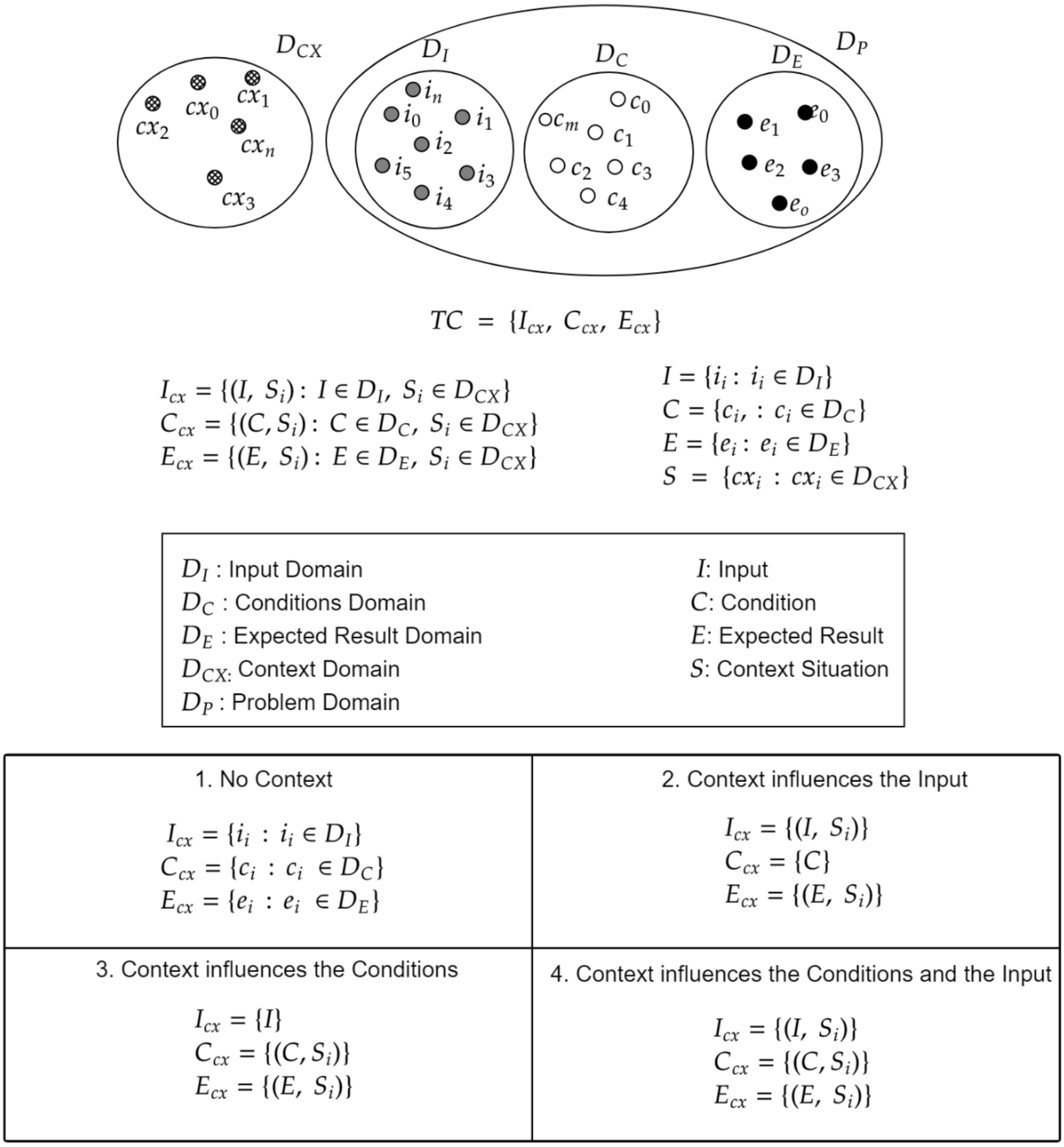
### 6. Identify Context Situations

From a testing perspective, the context variation will either affect the test input or the test conditions, which will demand different test strategies. As far as we know, there will be at least four testing situations when considering CASS testing. They are presented in Table 34 and made us realize the necessity of a new test case model capable of capturing the influence of the context and the different test scenarios.

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

*Table 34. Testing Strategies*

The CATS# test case model considers the context situation as the fourth element of a test case, as Figure 28 displays, to englobe these different scenarios that need to be represented and tested. Relevant variables and their values characterize a context situation. Hence, while specifying Test Case Input, Conditions, and Expected Result, the related context variables must also be considered.



$$TC = \{I_{cx}, C_{cx}, E_{cx}\}$$

$$I_{cx} = \{(I, S_i) : I \in D_I, S_i \in D_{CX}\}$$
$$C_{cx} = \{(C, S_i) : C \in D_C, S_i \in D_{CX}\}$$
$$E_{cx} = \{(E, S_i) : E \in D_E, S_i \in D_{CX}\}$$

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

$D_I$ : 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
$S$: Context Situation

| | |
|---|---|
| 1. No Context<br>$I_{cx} = \{i_i : i_i \in D_I\}$<br>$C_{cx} = \{c_i : c_i \in D_C\}$<br>$E_{cx} = \{e_i : e_i \in D_E\}$ | 2. Context influences the Input<br>$I_{cx} = \{(I, S_i)\}$<br>$C_{cx} = \{C\}$<br>$E_{cx} = \{(E, S_i)\}$ |
| 3. Context influences the Conditions<br>$I_{cx} = \{I\}$<br>$C_{cx} = \{(C, S_i)\}$<br>$E_{cx} = \{(E, S_i)\}$ | 4. Context influences the Conditions and the Input<br>$I_{cx} = \{(I, S_i)\}$<br>$C_{cx} = \{(C, S_i)\}$<br>$E_{cx} = \{(E, S_i)\}$ |

*Figure 28. CATS# Test Case Model*

CATS# Test Case Model can represent the four situations summarized in Table 34:

- **Conventional Test Case Model**: the context does not influence the software system. Consequently, Input, Conditions, and Expected Results do not change (they have static values). In this case, the model falls back to the conventional test case model (Figure 26.1), and conventional test strategies can be used.
- **CASS Test Case Model A**: the context influences the input but not the conditions. Thus, the input will vary, but the Conditions remain static. In this case, the test strategy will be the same as the Conventional Test Case Model. In addition, however, it is necessary to produce more test cases to cover all the context influences combinations of input (Figure 26.2).
- **CASS Test Case Model B**: the context influences the test case conditions but does not influence its input. Thus, the Conditions should accordingly vary while executing the test case. Therefore, a test strategy capturing context variation during the testing execution is necessary (Figure 26.3). This way, it will affect testing execution as much as the software system at runtime.
- **CASS Test Case Model C**: the context simultaneously influences the test input and the test conditions, combining CASS Test Case Models A and B. In this case, a specific testing environment is necessary to control the variations (Figure 26.4) simultaneously. Unfortunately, as far as we know, a test environment with this execution capacity is not available yet [11], and it is an open item for future works.

In this step, the software engineer must use the model from Figure 28 to identify which test scenarios the application has and, consequently, which test strategies should be used. This information should be considered in the next step.

### 7. List Test Cases

The previous steps had the goal of helping identify:

- context variables, thresholds, and expected results (steps 1, 2, and 3);
- the transitions among context situations (step 4);
- how the software system functionalities will be affected by them (step 5); and
- identifying which test strategy should be used for each test case (step 6).

This step had the goal of putting all this information together, creating an oracle, and listing all necessary test cases. The template from Table 35 will help the software engineer with this task since it has all the necessary fields.

| **TC Id** | **System Functionality** | **Input** | **Conditions** | **Expected Result** |
|---|---|---|---|---|
| TC01 | $<f_0>$ | $<I_{cx0}>$ | $<C_{cx0}> \rightarrow <C_{cx1}>$ | $<E_{cx1}>$ |
| TC02 | $<f_0>$ | $<I_{cx1}>$ | $<C_{cx0}>$ | $<E_{cx0}>$ |

*Table 35. CATS# v2 Test Oracle*

### 8. Select Test Cases

After putting all information together in step 7, it is time to move forward and select which test cases will be specified according to the chosen test strategy described in step 9. In an ideal scenario, all test cases listed in step 7 must be specified and executed. However, in situations with limited resources, a small but meaningful set of Test Cases could be selected from the list and specified. Possible test strategies are specifying at least one test case for each variation of context at the conceptual model or specifying the test cases for the most critical part of the software system.

### 9. Describe Test Cases

After creating the test oracle and selecting the test cases, it is time to describe them. CATS# makes available a tailored template to support the description of CASS test cases (see Table 36) to help the software engineer in this task. The intention is to cause a specific context variation during the test execution. In this way, the CATS# test template has a field called "Fixed Conditions" to describe the CVs that must remain fixed during the test execution and a "Varying Conditions" field to specify which CV must vary during the test execution.

However, describing the variation that must occur is not enough. It is also necessary to specify when it should occur. The template gains a structured flow to address this necessity, using tags such as 'ci' to indicate that the variation c1 must occur after each test step. For example, if there is a tag 'c1' after step 2, it means the testing execution must be paused until the variation specified in 'c1' happens. Once it is over, the test execution can continue to the next step.

| Id | <test case id> |
|---|---|
| **Test Objective** | <Briefly describes the test case goal objective> |
| **Preconditions** | <Describes the preconditions in general> |
| **Fixed Conditions** | <Describes the conditions that must remain static during the test execution> |
| **Input** | <Describes the test case input> |
| **Test Steps** | 1. <Describes the first step that should be followed><br>2. <Describes the second step …><br>3. <Describes the third step …> (c1)<br>4. <Describes the fourth step, after the variation specified in c1> |
| **Varying Conditions** | c1. <Describes the specific conditions that must vary in test step 3> |
| **Expected Result** | <Describes the expected result after the variation specified in c1> |
| **Pos Conditions** | <Describes the pos conditions in general> |

*Table 36. CATS# Test Template*

The remaining fields of the template are characteristics of regular test templates for a conventional system such as pre-conditions, test objective, input, expected result (which, in this case, must consider that the variation had occurred), and pos conditions.

It is worth mentioning that the pre and post conditions are not related only to the test case conditions that must vary but also to the test environment conditions, such as having the application installed in an online server.

It is the last step of the CATS# process, and it has, as output, a set of test cases to test context-aware software systems. Conventional Test Cases should also be included in the final set of test cases according to the system characteristics. In this case, the context-aware related fields (fixed and varying conditions) must be ignored, while all the others must be filled up.

## 5.3 Chapter Considerations

This Chapter presented the final version of CATS# with some small evolutions when considering the one presented in section 4.3. The updates done for the final version of CATS# were making step 5 optional, evolving the CATS# test case model, and updating the terms used across all templates to maintain consistency (for example, the "Effect" field from the analytical model became "Expected Result").

For a deeper explanation about each step of the process, Chapter 4 must be consulted. The next chapter will present a study conducted to evaluate the process and compare CATS# final version with the original process, CATS Design.

# 6 Assessment Study

## 6.1 Introduction

In the previous chapters, we described the path from CATS Design until the final version of CATS#. The technique was constructed using the Learn, Adapt/Build and measure Methodology [24]. Thus, we learn first from the literature and CATS Design, then by ourselves, and it was time to move forward and learn from observing other people using CATS#. We executed the assessment study described in this chapter with this goal in mind.

In the final stage of their graduation, the study was conducted by analyzing undergraduate students applying the CATS# technique to specify the test cases for the COVID Safe classrooms application.

This study was conducted during a software engineer course at the Federal University of Rio de Janeiro (UFRJ) during the second semester of 2020, while the whole world, especially Brazil, was suffering from the COVID-19 pandemic. The world's situation, in this case, had consequences in our daily lives. One of them was that UFRJ paralyzed most of the face-to-face activities and established strict biosafety protocols for the ones that still needed to happen. Considering the circumstances, the idea of COVID Safe classrooms arises as an application responsible for monitoring a specific classroom to guarantee the safety of students and professors that need to go to universities.

During the module, the students built a Minimum Viable Product (MVP) of the proposed application, using both requirements document (at the first phase of the project) and User Stories. Then, they created a test suite based on these documents and using CATS#. This chapter describes the details of the assessment study. More information about the COVID Safe classrooms application can be found in the repository[2].

[2] https://git-lab.cos.ufrj.br/ese_tecnodigital/safe/safe-ufrj/-/wikis/home

## 6.2 Study Planning

The main idea of this study was to compare CATS Design and CATS# to observe if there would be any advantages of using CATS#.

To evaluate this, the participants should use the design artifacts created for the COVID Saferoom application to design test cases using the CATS#. On the other hand, the same documentation should be used by us also to create a set of test cases using CATS Design this time.

The participants were undergraduate students from the Object-Oriented Software Development course (DSOO - "Desenvolvimento de Software Orientado a Objeto" in Portuguese), an elective module usually chosen by students at the end of their graduation course. The students enrolled in this module are usually from Engineers courses, especially Computing and Information Engineer, and Electrical and Computing Engineer from UFRJ. Due to COVID-19 restrictions, the module was offered remotely.

The idea was to present the CATS# technique during the class and make all templates from CATS# available for them.

From that point, a test case specification document would be required as one of the project's artifacts. After that, the participants would have the freedom to use just the parts of CATS# they considered useful (or not using CATS# at all). Then, from our side, when their specification document was complete for MVP 1, we would use them to describe test cases using CATS Design.

When both test case specification documents were ready, we would compare them, especially these three factors: Number of Identified CVS, Number of Test Cases in general, Number of CASS Test Cases.

The next section will present the study execution, while section 6.4 will show our results. Since the study was conducted in Rio de Janeiro, Brazil, the original artifacts were designed in Portuguese. Therefore, the examples shown in this Chapter were translated from Portuguese to English by the researchers.

## 6.3 Study Execution

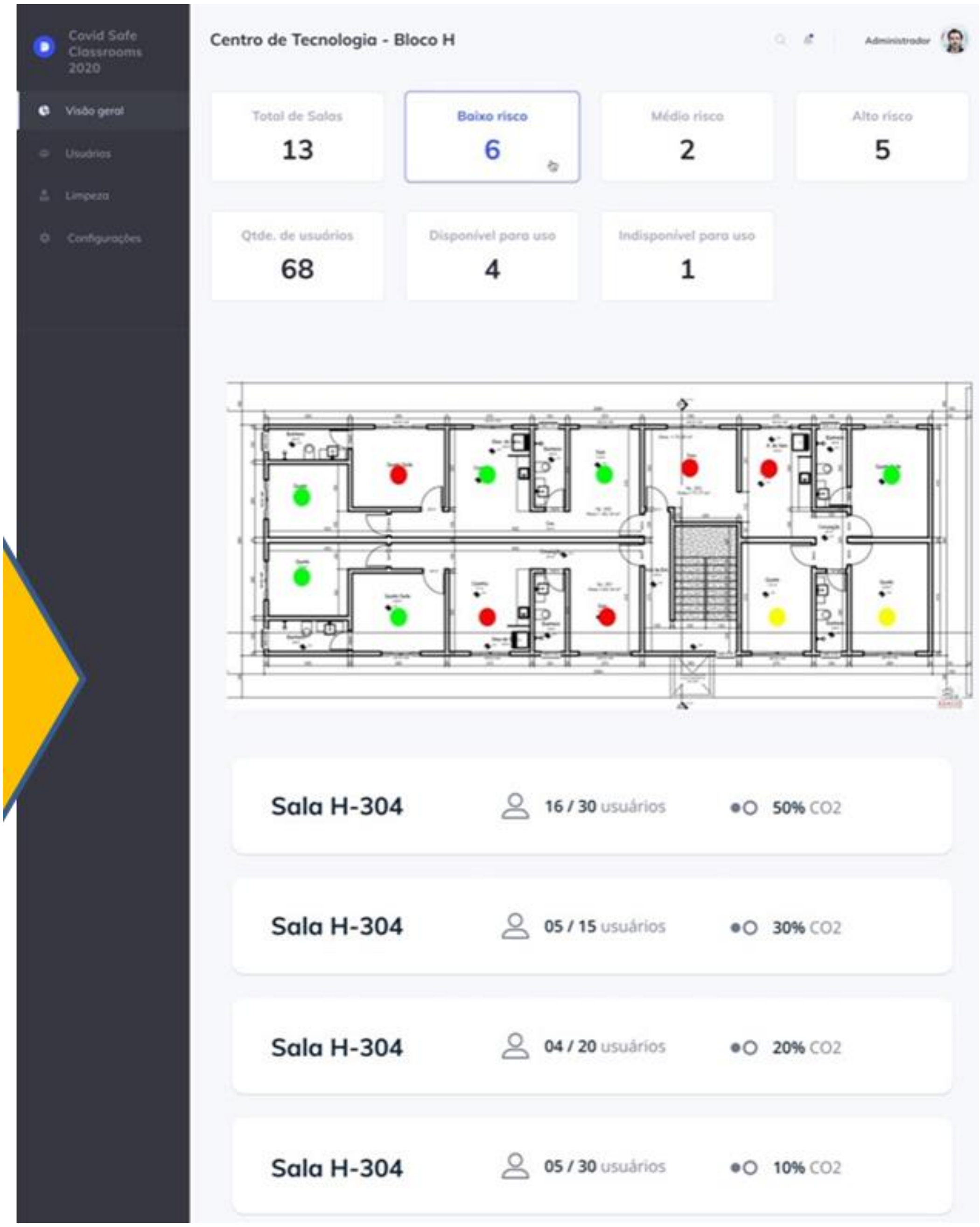
As mentioned in the previous section, the first step in this study execution was presenting CATS# to the participants, students enrolled in the DSOO course from UFRJ. The students must have previously attended the Software Engineering course to participate in this course. Therefore, they were all familiar with the main concepts of software testing for conventional systems (the ones mentioned in section 2.2). However, they needed to learn about Context and Context-Awareness before being introduced to CATS#.

In this way, we did a presentation to review the concepts of the software testing area, expose the concepts of context and context-aware software systems. Then, we introduced them to the CATS# process and artifacts.

Due to COVID-19 restrictions, the study was conducted remotely. The presentation was made during one of the DSOO classes, using a conference room, and the participants had time to explore and review the CATS# templates on their own after the class.

The second step was asking them for a test case specification document as one of the deliverables of the MVP of the COVID-19 Safe room. After that, they were free to use just the parts of CATS# they considered convenient for the application. They were also free to choose other methods and templates to specify the test cases.

As was mentioned in section 6.1, the goal of the COVID-19 Saferoom application was monitoring rooms considering environmental conditions regarding the COVID-19 pandemic. Therefore, the application would monitor CO2, Temperature, and Humidity levels. Additionally, it would also monitor the number of people in the room. Figure 29 shows the dashboard of the built application.



Centro de Tecnologia - Bloco H

| | | | |
|---|---|---|---|
| Total de Salas<br>13 | Baixo risco<br>6 | Médio risco<br>2 | Alto risco<br>5 |
| Qtde. de usuários<br>68 | Disponível para uso<br>4 | Indisponível para uso<br>1 | |

| | | |
|---|---|---|
| Sala H-304 | 16 / 30 usuários | 50% CO2 |
| Sala H-304 | 05 / 15 usuários | 30% CO2 |
| Sala H-304 | 04 / 20 usuários | 20% CO2 |
| Sala H-304 | 05 / 30 usuários | 10% CO2 |

*Figure 29. COVID Safe room dashboard*

The COVID-19 Saferoom MVP was based on a small set of requirements and user stories. Table 37 presents the requirements, and Table 38 presents an example of a User Story. It is worth mentioning that requirements usually define what the system shall do and the system restrictions. At the same time, User Stories are focused on what value a functionality (which can include more than one requirement) can bring to a specific user and the criteria to accept them [28].

| **Requirement ID** | **Description** |
|---|---|
| RF10 | The dashboard must present the risk level of one or more rooms in a floor plan. |
| RF11 | The system should display the detailed risk level of a room in a dashboard. |
| RF12 | The system should send notifications to administrators via the dashboard when a room's risk level changes to Red, following the standards defined in the Biosafety Guide. |
| RF17 | The system should indicate with an icon on the floor plan when a room needs cleaning automatically when the number of people in the installation being used drops to zero |
| RF18 | The system should allow the creation of users with Administrator and Employee profiles. |

*Table 37. Covid Saferoom requirements translated*

| ID | Issue 12 |
|---|---|
| Value Proposition | As an admin, I would like to see a notification with descriptive icons and a written message about the event on the dashboard whenever the risk level of room changes to a critical state to take appropriate action |
| Acceptance Criteria | When there is a problem, a popup message should appear informing the administrator of the problem.<br>Notification must be sent to all administrators to assume the event verification status.<br>The system should inform when a task is already assigned to an administrator<br>The system should not allow the same notification to be assigned to more than one administrator |

*Table 38. Issue 12 translated from Portuguese*

Using both User Stories and Requirements, the participants specified a set of test cases using CATS#. Table 39 shows the relationship between the specified requirements, user stories, and test cases.

| Requirements ID | User Stories | Test Cases ID |
|---|---|---|
| RF10 | Issue 10 | 01,02,03, and 04 |
| RF11 | Issue 11 | 05 |
| RF12 | Issue 12 | 06,07,08, and 09 |
| RF17 | Issue 13 | 10,11,12, and 13 |
| RF18 | Issue 14, Issue 38 | 14,15,16,17, and 18 |

*Table 39. Relation between requirements, user stories, and test cases*

The last step was using the same design documentation to specify the test cases using CATS Design. Until the end of the study execution, the researcher did not have access to the test cases specified by the participants. The participants also did not have access to the test cases specified by the researchers nor the CATS Design technique itself.

After the study execution, both test case specifications were compared. The results of this analysis are presented in the next section.

## 6.4 Study Results

### *6.4.1 Using CATS#*

The participants used the following templates for the test case documentation specification: Analytical Model, Conceptual Model, and Test Template. However, since they wrote User Stories, they broke the general test case specification into small pieces, each of them considering the different US. In this way, instead of having one analytical model and one conceptual model for the entire application, they made one for each US they classified as context-aware.

For example, based on the US described in Table 38, they identified two context variables: CO2 level and Number of People in the room, and designed the analytical model presented in Table 40.

| **Context-Variable** | **Threshold** | **Expected Result** |
|---|---|---|
| CO2 Level | CO2 Level ≥ 1000ppm | A message should appear in a popup informing that the CO2 level is high for a particular room for each admin. |
| Number of people in the room | Number of people in the room > Maximum number of people in the room | For each administrator, a message should appear in a popup informing that, for a given room, the maximum capacity has been exceeded. |

*Table 40. Analytical Model for Issue 12 translated from Portuguese*

Considering the Analytical Model from Table 40, they also designed the Conceptual Model presented in Figure 30, mapping the transitions among fourth context-situation application: No Notification, CO2 Notification, Maximum Number of people in the room notification, and a fourth situation join both CO2 and Maximum Number of people in the room.

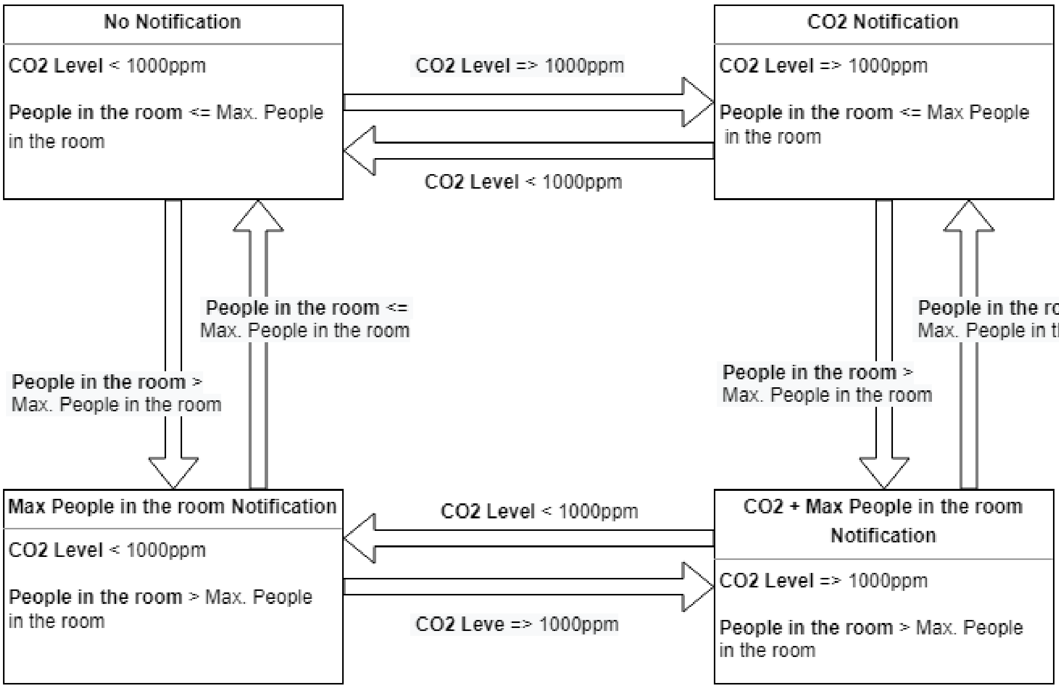

No Notification
CO2 Level < 1000ppm
People in the room <= Max. People in the room
CO2 Level => 1000ppm
CO2 Level < 1000ppm
CO2 Notification
CO2 Level => 1000ppm
People in the room <= Max People in the room
People in the room <= Max. People in the room
People in the room > Max. People in the room
People in the room <= Max. People in the room
People in the room > Max. People in the room
Max People in the room Notification
CO2 Level < 1000ppm
People in the room > Max. People in the room
CO2 Level < 1000ppm
CO2 Leve => 1000ppm
CO2 + Max People in the room Notification
CO2 Level => 1000ppm
People in the room > Max. People in the room

*Figure 30. Conceptual Model for Issue 12*

They specified four test cases for this issue: two conventional and two contexts aware. The two CASS test cases, presented in Tables 41 and 42, were related to the CO2 level going low, reaching 1000 ppm and the number of people in the room reaching their maximum, respectively. Both test cases would trigger system notifications to indicate a critical situation that needs intervention.

| **ID** | 06 |
|---|---|
| **Test Objective** | Testing the functionality of triggering a notification when the CO2 level reaches a critical state |
| **Preconditions** | • CO2 Level < 1000ppm |
| **Fixed Conditions** | Number of people in the room <= Maximum number of people in the room |
| **Input** | Current CO2 Level |
| **Test Steps** | 1. Execute steps 1 to 6 from test case 5<br>2. Wait until the CO2 Level notification arises |
| **Varying Conditions** | CO2 Level < 700ppm → CO2 Level ≥ 1000ppm [1]<br>[1] The variation will occur by using a simulator |
| **Expected Result** | A red notification with the text 'Room XXXX Has a CO2 Level Problem' and a button that says 'Take this case' should appear to all administrators. |
| **Pos Conditions** | CO2 Level ≥ 1000ppm |

*Table 41. Test Case 06 for Issue 12 translated from Portuguese*

| Id | 07 |
|---|---|
| Test Objective | Testing the functionality of triggering a notification when the number of people reaches its maximum<br>the level reaches a critical state |
| Preconditions | - Number of people in the room <= Maximum Number of people in the room |
| Fixed Conditions | CO2 Level < 1000ppm |
| Input | Number of People in the room |
| Test Steps | 1. Execute steps 1 to 6 from test case 5<br>2. Wait until the Maximum people in the room notification arises |
| Varying Conditions | Current Number of People in the room <= Maximum Number of People in the room → Current Number of People in the room > Maximum Number of People in the room[1]<br>[1] The variation will occur by using a simulator |
| Expected Result | A red notification with the text 'Room XXXX Has a Capacity Problem' and a button that says Take this case' should appear to all administrators. |
| Pos Conditions | Current Number of People in the room > Maximum Number of People in the room |

*Table 42. Test Case 07 for Issue 12 translated from Portuguese*

After specifying test cases for the critical situations, they specified two conventional test cases, displayed in Tables 43 and 44.

| Id | 08 |
|---|---|
| Test Objective | Testing the functionality of assigning an employee to a critical situation |
| Preconditions | - A notification should be appearing on the screen with a "take the case" button |
| Test Step | 1. Admin sees the notification<br>2. Admin selects the "take the case" button |
| Expected Result | The notification color should change to orange, and it should inform Admin X is handling the critical situation. |

*Table 43. Test Case 08 for Issue 12 translated from Portuguese*

| **Id** | 09 |
|---|---|
| **Test Objective** | Test the functionality of only one administrator taking on a specific notification. |
| **Preconditions** | • Two users with the admin profile must be logged into the system<br>• A notification should be popping up on the screen with the 'Take the case' button |
| **Test Step** | 1. Admins 1 and 2 see the notification<br>2. Admin1 and 2 click the 'Take the case' button |
| **Expected Result** | Only one Admin was able to take over the notification. Therefore, the notification must change its color to an orange hue and inform the other admin that they are taking action. |

*Table 44. Test Case 09 for Issue 12*

Table 45 summarizes the findings from this part of the experiment. The next section will show the application of CATS Design to specify the test cases.

| COVID-19 Saferoom – CATS# version | |
|---|---|
| CV | 2 |
| Test Cases | 18 |
| CASS Test Cases | 06 |

*Table 45. Summary of test cases*

### 6.4.2 Using CATS Design

The first step was analyzing the requirements while looking for Context Variables. In this step, eight CVs were identified. They are listed below:

- Number of people in the room
- Temperature
- CO2 Level
- Risk Level
- Maintenance Status
- Cleaning Status

- Internet Status

No additional variables were found during the second step, Identify Additional Context Variable.

Since the information from the required documentation was limited, we continued using the User Story. As they were self-contained, a similar approach from section 6.4.1 was used, and the models (both conceptual and analytical) were designed based on each user story. However, while the participants ignored the other User Stories during creating the new models, we tried to aggregate the information found during the specification. One example was the Conceptual Model for Issue 12 (in section 6.4.1), Figure 31. Two CVs were considered to design the model: Number of people in the room and CO2 Level.

| ID | Issue 10 |
|---|---|
| Value Proposition | As a general user, I would like to have access to a floor plan view with the risk level of every room so I can assess the risk I will be exposed to |
| Acceptance Criteria | • The risk level should be represented by a circle with the color assigned to its static classification.<br>• When the number of people in a room is greater than or equal to the room limit, an icon indicating that the room is full should be shown.<br>• When the CO2 level exceeds 700ppm, a yellow icon should be displayed on the floor plan indicating a problem with CO2, and when this level exceeds 1000ppm, the same red icon should be displayed. |

*Table 46. Issue 10*

However, we knew from Issue 10 (Table 46) that if the CO2 level reaches 700 ppm, the system should change the color of the room in the dashboard for yellow, and we knew from Issue 12 that if the CO2 Level reaches 1000 ppm, a notification must appear to the admins. Hence, both issues were considered while designing the Conceptual Model from Figure 31.

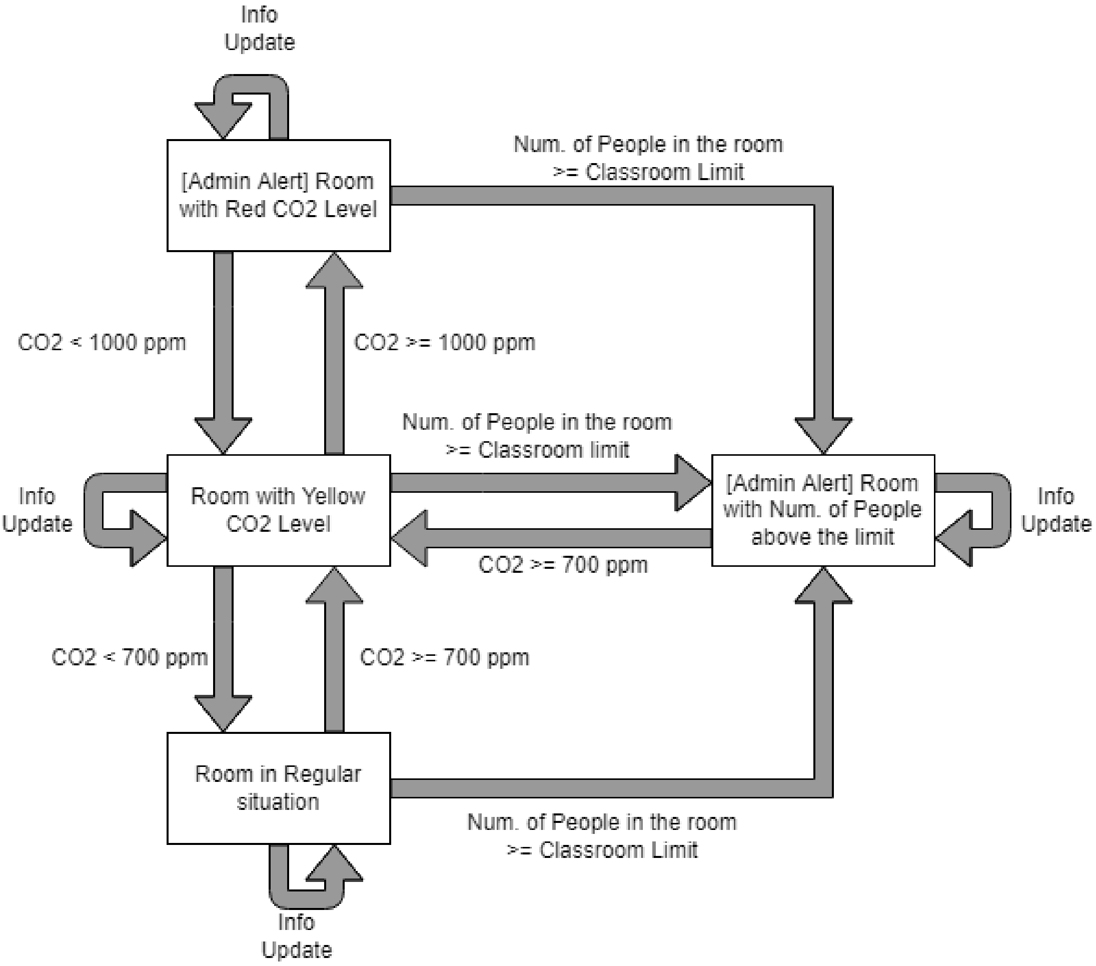

Info
Update
[Admin Alert] Room with Red CO2 Level
Num. of People in the room >= Classroom Limit
CO2 < 1000 ppm
CO2 >= 1000 ppm
Num. of People in the room >= Classroom limit
Info
Update
Room with Yellow CO2 Level
[Admin Alert] Room with Num. of People above the limit
Info
Update
CO2 >= 700 ppm
CO2 < 700 ppm
CO2 >= 700 ppm
Room in Regular situation
Num. of People in the room >= Classroom Limit
Info
Update

*Figure 31. Issue 12 Conceptual Model - Cats Design*

With this model, two thresholds were identified: The number of people reaching the maximum number and CO2 reaching 1000 ppm.

The next step was creating the analytical model from Table 47. With this model, a new threshold was found, Number of people in a room = 0. Although this case's effect was not described in Issue 12, we found the information in Issue 13 and added it to the model.

| Context Variables | Effect |
|---|---|
| Number of People | = 0: Send a notification about the need for cleaning<br>≥ room's limit: Display an icon indicating that the Number of People is above the limit |
| CO2 | ≥ 1000 ppm: Triggers Red Alert/Notification |

*Table 47. Analytical Model from CATS Design translated from Portuguese*

After identifying the threshold, the test oracle was described, as shown in Table 48.

| Feature | Context | Expected Output |
|---|---|---|
| Inform the need for cleaning | Number of People = 0 (empty room) | Send a notification about the need for cleaning |
| Notify that the number of people in a room is above the limit (show icon) | Number of People reaches the room's limit | Send a notification about the number of people in the room above the limit |
| Inform about the CO2 level | CO2 level reaches 700 ppm | Send a notification about the CO2 level |
| Notify about the CO2 level | CO2 level reaches 1000 ppm | Send a notification about the CO2 level |

*Table 48. CATS Design Test Oracle translated from Portuguese*

The last step, in this case, is to describe the Test template, as shown in Table 49.

| | |
|---|---|
| Test Case ID | CATS02 |
| Test Objective | Verify the functionality of informing the risk level |
| Precondition: | The user is authenticated as an admin |
| Test Input: | |
| Test Steps: | 1. An authenticated user accesses the application mains page |
| Relevant Context Variables: | 1. Number of People<br>2. CO2 Level |
| Known Thresholds: | a. Number of People = 0 (empty room)<br>b. Number of People >= room's limit (crowded room)<br>c. CO2 level reaches 1000 ppm |
| Test Expected Outputs for each Threshold | a. Send a notification about the need for cleaning<br>b. Send a notification about the number of people in the room above the limit<br>c. Send a notification about the CO2 level |

*Table 49. CATS Design test template*

Table 50 summarizes the findings from this part of the experiment. The next section will compare both parts of this study, CATS# (section 6.4.1) and CATS Design, and discuss the general findings.

| **COVID-19 Saferoom – CATS Design version** | |
|---|---|
| CV | 2 |
| Test Cases | 03 |
| CASS Test Cases | 03 |

*Table 50. Summary of test cases*

### 6.4.3 Discussion

Table 51 shows a comparison between CATS Design and CATS#. As it is possible to observe, the number of Identified CVs to this first MVP was the same in both techniques. However, the number of test cases increased from three in CATS Design to 18 in CATS#. Regarding this, we could identify two reasons.

| | CATS Design | CATS# |
|---|---|---|
| **Identified CVs** | 02 | 02 |
| **Number of Test Cases** | 03 | 18 |
| **Number of CASS Test Cases** | 03 | 06 |

*Table 51. Comparison between CATS Design and CATS#*

The first one is that the CATS# process includes four test cases model, including one for situations where context will not interfere, and the conventional test case should be used. Therefore, conventional test cases will also be part of one specification conducted by the CATS# process. The participants identified and specified 12 conventional test cases for the COVID-19 Saferoom application. On the other hand, CATS Design was completely focused on specifying just CASS test cases, and no conventional test case was identified in this case.

Nevertheless, the number of CASS test cases also increased from three in CATS Design to six in CATS#, and the reason was the difference between the test template used by each technique.

The goal of the CATS Design test template is to create an oracle and map all the possible variations that could occur during the test execution. It does not mean they will occur, but the expected behavior is mapped if they do. Table 52 shows the same test template from section 6.4.2. The goal of the test is to "Verify the functionality of informing the risk level," and if one of the changes described in the Known Threshold fields occurs, the "Test Expected Outputs for each threshold" can tell us what to expect. However, these context variations might not happen at all.

| Test Case ID | CATS02 |
|---|---|
| Test Objective | Verify the functionality of informing the risk level |
| Precondition: | Usuário Autenticado como Administrador<br>User is authenticated as an admin |
| Test Input: | |
| Test Steps: | 2. An authenticated user accesses the application mains page |
| Relevant Context Variables: | 3. Number of People<br>4. CO2 Level |
| Known Thresholds: | d. Number of People = 0 (empty room)<br>e. Number of People >= room's limit (crowded room)<br>f. CO2 level reaches 1000 ppm |
| Test Expected Outputs for each Threshold | d. Send a notification about the need for cleaning<br>e. Send a notification about the number of people in the room above the limit<br>f. Send a notification about the CO2 level |

*Table 52. CATS Design test template example*

In CATS#, the idea is to specify which variations must occur during the test execution and cause them to validate the software system behavior. In this way, for the CATS Design test case from Table 52, three different CATS# test cases should be specified: one for the number of people reaching zero, another for the number of people reaching the maximum allowed, and the last one for the CO2 level reaching 1000 ppm. It exactly was what the participants did, as shown in Tables 53, 54, and 55, respectively.

| **ID** | 10 |
|---|---|
| **Objetivo do Teste** | Test the functionality of triggering a cleaning notification when the number of people in the room is zero. |
| **Pré-condições** | Current Location > 0 |
| **Condições Fixas** | Not Applicable |
| **Entrada** | Number of People in the room |
| **Passos do Teste** | 1. Access the application's home page<br>2. Click the login button located at the top right of the screen<br>3. Put the user with the profile Employee testefuncionario@teste.com<br>4. Enter the password testefuncionariossafeurfj<br>5. Click on the login button<br>6. Wait for the login processing to finish<br>7. Wait until the number of people in the room is zero. |
| **Condições Variáveis** | Number of People in the room > 0 → Number of People in the room = 0 [1]<br>[1] The variation will occur by using a simulator |
| **Resultado Esperado** | • A red notification with the text 'Room XXXX Needs Cleaning' and a button that says "Take the case" should appear to all employees<br>• A cleaning icon should appear on the floor plan |
| **Pós-condições** | Number of people in the room = 0 |

*Table 53. Test Case 10 for Issue 12*

| Id | 07 |
|---|---|
| Test Objective | Testing the functionality of triggering a notification when the number of people reaches its maximum<br>the level reaches a critical state |
| Preconditions | - Number of people in the room <= Maximum Number of people in the room |
| Fixed Conditions | CO2 Level < 1000ppm |
| Input | Number of People in the room |
| Test Steps | 1. Execute steps 1 to 6 from test case 5<br>2. Wait until the Maximum people in the room notification arises |
| Varying Conditions | Current Number of People in the room <= Maximum Number of People in the room → Current Number of People in the room > Maximum Number of People in the room[1]<br>[1] The variation will occur by using a simulator |
| Expected Result | A red notification with the text 'Room XXXX Has a Capacity Problem' and a button that says Take this case' should appear to all administrators. |
| Pos Conditions | Current Number of People in the room > Maximum Number of People in the room |

*Table 54. Test Case 07 for Issue 12*

| | |
|---|---|
| **ID** | 06 |
| **Test Objective** | Testing the functionality of triggering a notification when the CO2 level reaches a critical state |
| **Preconditions** | • CO2 Level < 1000ppm |
| **Fixed Conditions** | Number of people in the room <= Maximum number of people in the room |
| **Input** | Current CO2 Level |
| **Test Steps** | 1. Execute steps 1 to 6 from test case 5<br>2. Wait until the CO2 Level notification arises |
| **Varying Conditions** | CO2 Level < 700ppm → CO2 Level ≥ 1000ppm [1]<br>[1] The variation will occur by using a simulator |
| **Expected Result** | A red notification with the text 'Room XXXX Has a CO2 Level Problem' and a button that says 'Take this case' should appear to all administrators. |
| **Pos Conditions** | CO2 Level ≥ 1000ppm |

*Table 55. Test Case 06 for Issue 12*

Besides the test template, another difference between CATS Design and CATS# is that the last classify the CVs in two different types regarding the test case: the ones that impact the test case input and the ones that impact the test case conditions. As it was possible to observe by the specified test cases, the participants classified both variables, the number of people in the room and CO2 level, as attached to the test cases conditions. It is a discrepancy between what we thought initially and what they did. CATS Design did not require context variable classification, but as going through the process, we noticed, based on our experience, they would impact mostly the test case input (CASS Model A).

Two hypotheses can justify this discrepancy. First (and most probable), the participants' lack of experience in Context-Aware applications might cause some confusion between the CASS Model A and B, and they end up classifying erroneously. The second one is that, during practice, there will be no real differences between CASS Model A and CASS Model B regarding the validation. Therefore, varying the CVs during the test execution would be welcomed for both models. These hypotheses need further investigation and are open items to be addressed in future works.

Another unexpected point for using both CATS Design and CATS# is that they were both built considering as an initial input the complete requirements specification. However, they did well with an incremental agile development approach. In addition, we did not notice any cons in having more than one analytical and conceptual model since the test cases were specified the same way they would be using a Waterfall methodology.

Finally, two more considerations must be made. The first is that the participants classified the CATS# process as easy to use. However, the test case classification discrepancy showed they misunderstood some of the CATS# concepts. Additionally, since the MVP1 was constructed remotely and could not access a classroom with sensors and tags, they simulated the data by code. It is interesting because, as was mentioned in Chapter 4, there will be cases that cause the variation can be complicated. For them, using simulated data as input sent to the software system could be a nice thing to have.

### *6.4.4 Threats to validity and limitations*

The main limitation of this study was that it was conducted during the COVID-19 pandemic when, due to safety protocols, the participants could not access the University. Therefore, it was necessary to execute the study remotely without having control of the participants' environment and behaviors.

Another limitation was the application itself. When we planned to use the COVID-19 Saferoom application, we considered the general scope, with context-aware test situations from CASS Models A and B. However, since just the first MVP was ready during the study execution, we had limited test situations classified as conventional and CASS Modal A. Therefore, CASS Model B, the most interesting scenario for us, could not be truly observed during this study. This issue also impacted the generalization of results since we did not have a safety-critical and complete context-aware application at the end.

Additionally, it was not possible to execute the specified test cases since the application went offline right after the end of the course.

Finally, it is also necessary to mention the researcher's experience when compared with the participants. Although undergraduate students from software engineer areas can be compared with practitioners at the beginning of their careers [29], the researcher had more experience regarding the context. Therefore, some particularities would be more easily perceived independently of the process that guided the test case specification.

The researcher's natural interest in the outcome was also a threat to validity. We partially handled this by not accessing the test cases designed by the participants before having our test case specification. However, the previous knowledge acquired during the creation of CATS# can influence even if we use the CATS Design technique.

## 6.5 Chapter Considerations

This chapter presented the assessment study to compare CATS# and CATS Design. To do this comparison, we used the COVID-19 Safe classrooms application, responsible for monitoring rooms to guarantee the safety of students and professors that needed to go to universities during the COVID-19 pandemic. The first MVP was used to create the set of test cases that would be compared. The participants, students from computing engineer courses from UFRJ enrolled in the Object-Oriented Software Development module, specified the test cases using CATS# while the researcher used CATS Design.

As a result of this study, we noticed that both techniques identified the same number of Context Variables. However, the specification using CATS Design returned three test cases (all of them context-aware), while the one done using CATS# returned 18 test cases (six of them context-aware). Two reasons explain this difference. First, CATS# also include the test specification of conventional test cases scenario. Second, while CATS Design intended to map any possible context variation that could (but might not) happen during the test execution, CATS# wanted to specify the exact variation that must occur to validate the system.

We also noticed that both techniques behaved well in an incremental agile development approach, even if they were not constructed to this methodology. Additionally, the participants reported the technique was easy to use, but the produced models indicate they misunderstood some of the CATS# concepts.

Finally, the study was executed during the COVID-19 pandemic when, due to safety protocols, the participants could not access the University. Therefore, it was necessary to execute the study remotely without having control of the participants' environment and behaviors. In addition, the MVP used to design the test cases was also not completely context-aware, which limited our results. However, our general findings show that CATS# works, even if it needs validation. During the study execution, some other questions also arose, such as how difficult it could be to differentiate between CASS models A and B. In practice, these differences might not be considered since both scenarios require accurate testing. Nevertheless, these questions will be considered open items to be addressed in future work.

# 7 Conclusion

## 7.1 Introduction

This research had four phases: acquire initial knowledge on the problem, complement by searching the literature, construct a solution and evaluate the solution.

The first one presented the particularities of testing Context-Aware Software System and why conventional testing strategies are not enough for testing them. It also presented how the meaningful parts of the context, which are infinite and abstract, can be captured through context variables and how they can affect software systems.

The second phase was about conducting a Structure review to observe how the context and its variation usually affected context-aware application in the production phase. After analyzing the results found in the literature, a new test case model for CASS was proposed and evolved throughout this work.

In the third phase, we evolved the CATS Design testing technique into a new one, called CATS#. CATS# was built using the Learn, Adapt/Build, and Measure methodology, and it had the goal of englobe the knowledge acquired during the second phase. In addition, it evolved the CASS test case model mentioned previously and proposed a new testing template to englobe the context variation during the test execution.

The last phase was conducting an assessment study to observe people not involved in this research using CATS#.

This chapter will present the main contributions of this research. It also presents the identified limitations and how this work can evolve in future work.

## 7.2 Contributions

The main contribution of this research is the CAST# technique. An evolution of the CATS Design technique, grounded in evidence to help software engineers during the specification of test cases for context-aware software systems from different domains (such as IoT, Smart Buildings, WSN, among others).

Although CATS# englobes both process and templates meant to work together, they can also be used separately. Therefore, the analytical and conceptual models, the Situation Matrix, and the test template (which help the user cause the context variation during the test execution) can also be considered contributions of this work.

Additionally, as a side effect of applying the CATS# technique, gaps in the specification document were found, which means the initial steps of CATS# can also be used as an inspection technique to improve requirements considering the context perspective.

This work also contributes to the organization of the context-awareness conceptual background, transforming them into mathematical models that allow a more precise representation of elements such as context, context variables, context situations. Going one step ahead, we also created an enhanced test case model, which includes the context as a fourth element to demonstrate how it will affect the test of CASS. We also propose different strategies that should be adopted in each case and can be used as a guide to software engineers during the testing of CASS.

As a contribution of the master's in general, it is possible to cite the following material:

- The Towards Supporting the Specification of Context-Aware Software System Test Cases article [15] and presentation[3] into the XXIII Ibero-American Conference on Software Engineering (CIbSE) where we presented the first version of CATS# technique
- An article with the title "Digitalizando o Microscópio Óptico: a solução Parasite Watch" presented in the "Simpósio Brasileiro de Computação Aplicada à Saude", a conference focused on the computing application aplied in the health area. In this case, the article was about the construction of the Parasite Watch Solution

[3] https://www.youtube.com/watch?v=qKilvz8FvTE&ab_channel=CIbSE2020

- An article with the name "Utilizando o Framework MDA para Avaliar a Estética de um Jogo: Um Estudo Preliminar sobre a Percepção de Estudantes de Graduação" published in the SBGames, a Brazilian conference focused in games.
- A technical report named “A Rapid Review on Testing of Context-Aware Contemporary Software Systems " searched strategies for testing CASS in the literature. The protocol from this work was reused and adapted in Chapter 3

It is also worth mentioning the participation in some academic projects such as:

- A Research Internship at the Rochester Institute of Technology, conducting a structured review about the architectural antipatterns
- The participation in different projects from the Experimental Software Engineer lab such as Parasite Watch, Camarão IoT (an application to monitor the creation of freshwater shrimp; OximetroIoT (a project to create a low-cost oximeter);
- The COPPE/UFRJ Women's support group helped with technical activities and participated in a speech about gender parity challenges in the post-pandemic context.
- The participation in a project to teach kids (especially girls) how to code using Python
- The co orientation of an undergrad student in his final work where he created a chatbot to attend and help the students from the Computing and Information Engineer course from UFRJ

## 7.3 Limitations

The main limitation of this work is the lack of evaluation of CATS# with a real Context-Aware Software System which englobes the four test scenarios mentioned before. The assessment study conducted and presented in Chapter 6 was limited and presented many threats to validity, such as:

- The participants were not practitioners but software engineers students
- The study was conducted remotely without the possibility of controlling the participants' environment or fully observing their behavior during the execution.
- The application was limited when considering the context-aware testing scenarios available, and the most significant scenario (CASS Test Case Model B) could not be evaluated.
- It was not possible to execute the specified test cases in any of the performed experiments since they were not available

Due to issues related to research time and environment (such as the COVID-19 crisis), it was not possible to execute more experimental studies to assess the validity of the proposed technique. Therefore, as will be mentioned in the next section, more robust experiments are considered future work.

## 7.4 Future Work

It is possible to mention the necessity of more robust experiments using CATS# to specify the test cases for complex and fully context-aware applications in future work and execute the specified test cases. With these evaluations, it would be possible to use the Learn, Adapt/Build, and Measure methodology and continue evolving CATS#.

Additionally, some questions arose after the study execution, such as how difficult it could be to differentiate between CASS models A and B and whether these differences will be significant in practice as they are in theory. These questions need further investigation and should be addressed in the future.

Finally, as was mentioned before in this research, forcing the context to vary as specified is not always an easy task, especially in cases where both input and conditions must vary during the test execution. Therefore, to completely support CASS testing, it is necessary to build a tool to help software engineers control the Test Environment.

As far as we are aware of it, to ensure the quality of CASS, the context must vary as much as it would in real life. CATS Design was the first step towards this goal, and

CATS# was another one. Building this new test environment is the next. After putting all these pieces together, it will be, finally, possible to unchain the context and set it free. [9]

# REFERENCES

[1] AL-FUQAHA, A. et al., “Internet of things: A survey on enabling technologies, protocols, and applications.” *IEEE communications surveys & tutorials*, v. 17, n. 4, p. 2347-2376, 2015.

[2] MARTINI, B. G. et al., “IndoorPlant: A Model for Intelligent Services in Indoor Agriculture Based on Context Histories.” *Sensors*, v. 21, n. 5, p. 1631, 2021.

[3] ANDRADE, R. M. C. et al. “Multifaceted infrastructure for self-adaptive IoT systems.” *Information and Software Technology*, v. 132, p. 106505, 2021.

[4] PRIYADARSHINI, I. et al., “A new enhanced cyber security framework for medical cyber-physical systems.” **SICS Software-Intensive Cyber-Physical Systems**, p. 1-25, 2021.

[5] DE SOUZA, B. P., MOTTA, R. C., TRAVASSOS, G. H., “The first version of SCENARIotCHECK: A Checklist for IoT based Scenarios.” *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. 2019. p. 219-223.

[6] MOTTA, R. C., DE OLIVEIRA, K. M., TRAVASSOS, G. H., “Rethinking interoperability in contemporary software systems.” *2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS)*. IEEE, 2017. p. 9-15.

[7] DEY, A. K., ABOWD, G. D., 1999, *Towards a better understanding of context and context-awareness*. GVU Technical Report GIT-GVU-99-22. Georgia Institute of Technology. Atlanta, GA. Retrieved from https://smartech.gatech.edu/handle/1853/3389

[8] SILVA, F. R., 2016, *Cats Design: A Context-Aware Testing Approach*. Master’s dissertation, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil.

[9] MATALONGA, S., TRAVASSOS, G. H., “Testing context-aware software systems: Unchain the context, set it free!”. *In Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES'17)*. ACM, New York, NY, USA, 250-254. DOI: https://doi.org/10.1145/3131151.3131190

[10] QUADRI, S. M. K., FAROOQ, S. U., 2010, “Sofware Testing – Goals, Principles, and Limitations,” *International Journal of Computer Applications*, v. 6, n. 9 (Sep), pp. 1.

[11] AMALFITANO, D. et al., 2019, *A Rapid Review on Testing of Context-Aware Contemporary Software System*. https://www.cos.ufrj.br/uploadfile/publicacao/2910.pdf.

[12] MOTTA, R. C., “Towards a Strategy for Supporting the Engineering of Contemporary Software Systems." **arXiv preprint arXiv:1904.11741**, 2019.

[13] International Organization of Standardization. (2013). *Software and systems engineering – Software testing –* (ISO/IEC/IEEE 29119-1)

[14] DELAMARO, M. E., MALDONADO, J. C., JINO, M. “Conceitos Básicos”. *Introdução ao Teste de Software,* 4th ed, chapter 1, Rio de Janeiro, Brazil, Elsevier Editora Ltda, 2007

[15] DORESTE, A. C. S., TRAVASSOS, G. H., “Towards Supporting the Specification of Context-Aware Software System Test Cases*,” XXIII Ibero-American Conference on Software Engineering (CIbSE),* Curitiba, Brazil, 2020

[16] MATALONGA, S., RODRIGUES, F., TRAVASSOS, G. H., 2017, "Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review." *Journal of Systems and Software*, v. 131 (2017), pp. 1-21.

[17] LUO, C. et al., 2020, "A survey of context simulation for testing mobile context-aware applications." *ACM Computing Surveys (CSUR),* v.53, n.1 (2020), pp. 1-39

[18] ALMEIDA, D. R., MACHADO, P. D. L., ANDRADE, W. L., 2019, "Testing tools for Android context-aware applications: a systematic mapping." *Journal of the Brazilian Computer Society* v. 25, n.1 (2019), pp. 1-22.

[19]MATALONGA, S. et al., 2021, "Alternatives for Testing of Context-Aware Contemporary Software Systems in industrial settings: Results from a Rapid review." arXiv preprint arXiv:2104.01343, Retrieved from https://arxiv.org/abs/2104.01343

[20]SIQUEIRA, B. R., et al., 2021, "Testing of adaptive and context-aware systems: approaches and challenges." *Software Testing, Verification, and Reliability*, v. 31, n. 7, e1772.

[21] BOOTH, A., SUTTON, A., PAPAIOANNOU, D., 2016, “Systematic approaches to a successful literature review.”, 2nd ed, SAGE Publications Ltd

[22] MIRZA, A. M., KHAN, M. N. A., “An Automated Functional Testing Framework for Context-aware Applications.” *IEEE Access*, 6, 46568-46583, 2018.

[23] AFANASOV, M. et al., “Software Adaptation in Wireless Sensor Networks.” ACM Trans. Auton. Adapt. Syst. 12, 4, Article 18, 29 pages, Jan. 2018. https://doi.org/10.1145/3145453

[24] CAROLI, P. 2017, *To the Point – A Recipe for Creating Lean Products*, Leanpub

[25] DELAMARO, M. E., MALDONADO, J. C., JINO, M. “Conceitos Básicos”. *Introdução ao Teste de Software,* 4th ed, chapter 3, Rio de Janeiro, Brazil, Elsevier Editora Ltda, 2007

[26] DORESTE, A. C. S., et al., “Digitalizando o Microscópio Óptico: a solução do Parasite Watch”. Simpósio Brasileiro De Computação Aplicada À Saúde (SBCAS), 19. , 2019, Niterói. Rio de Janeiro, Brazil, 2019

[27-] DORESTE, A. C. S., *Pipeline de Implantação Contínua no Contexto de Internet das Coisas para Raspberry Pi*, Trabalho de Conclusão de Curso, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. Available at: http://www.repositorio.poli.ufrj.br/monografias/monopoli10024252.pdf, Last Accessed on November 22nd, 2021

[28-] VALENTE, M. T., “Engenharia de Software Moderna,” Available at: https://engsoftmoderna.info/, Last Accessed on November 22nd, 2021

[29-] CARVER, L. et al., “Issues in using students in empirical studies in software engineering education,” in Proceedings - I*nternational Software Metrics Symposium*, vol. 2003-Janua, pp. 239– 249. 2003

[30] DIAS-NETO, A. C., MATALONGA, S., SOLARI, M., ROBIOLO, G., TRAVASSOS, G. H., 2017, "Toward the characterization of software testing practices in South America: looking at Brazil and Uruguay," *Software Quality Journal*, v. 24, n. 4 (2017), pp. 1145-1183

[31] SANTOS, I. S., ANDRADE, R. M. C., ROCHA, L. S., MATALONGA, S., OLIVEIRA, K. M., TRAVASSOS, G. H., 2017, "Test case design for context-aware applications: Are we there yet?", *Information and Software Technology*, v. 88 (2017), pp. 1-16

[32] MATALONGA, S., RODRIGUES, F., TRAVASSOS, G. H. (2015). "Matching context-aware software testing design techniques to ISO/IEC/IEEE 29119", *International Conference on Software Process Improvement and Capability Determination*, Springer, Cham, 2015. p. 33-44.

# Appendix A – Structure Review Extractions

## A.1 An Automated Functional Testing Framework for Context-aware Applications

| Paper ID | 12 |
|---|---|
| Bibliography: | Mirza, A. M., & Khan, M. N. A. (2018). An Automated Functional Testing Framework for Context-aware Applications. *IEEE Access*, *6*, 46568-46583. |
| Abstract: | “In the modern era of mobile computing, context-aware computing is an emerging paradigm due to its wide spread applications. Context-aware applications are gaining increasing popularity in our daily lives since these applications can determine and react according to the situational context and help users to enhance usability experience. However, testing these applications is not straightforward since it poses several challenges, such as generating test data, designing context-coupled test cases, and so on. However, the testing process can be automated to a greater extent by employing model-based testing technique for context-aware applications. To achieve this goal, it is necessary to automate model transformation, test data generation, and test case execution processes. In this paper, we propose an approach for behavior modeling of context-aware application by extending the UML activity diagram. We also propose an automated model transformation approach to transform the development model, i.e., extended UML activity diagram into the testing model in the form of function nets. The objective of this paper is to automate the context-coupled test case generation and execution. We propose a functional testing framework for automated execution of keyword-based test cases. Our functional testing framework can reduce the testing time and cost, thus enabling the test engineers to execute more testing cycles to attain a higher degree of test coverage.” |
| General Information: | • “ However, for both of the testing types, test case generation and maintenance are expensive and difficult tasks. Using model based testing techniques are generally considered as a solution to this problem. Model based testing is a test automation technique which is regarded as a process to automate test designing to generate test cases from system under test (SUT) model [2].”<br>• “Our proposed framework would automate testing process of context-aware applications which includes generation and |

| | |
|---|---|
| | execution of context-coupled test cases to evaluate accuracy of context recognition and adaptation."<br>- "we present our test automation framework named as ContextDrive. Our proposed model consists of six phases. First two phases deal with behavior modeling and model transformation. Our previous study [13] mainly focuses at model transformation and explains step by step the entire model transformation process followed by the initial results obtained by implementing first two phases. In this paper, we have adopted the same transformation process followed by describing an automated functional testing procedure to generate and execute context-coupled test cases. This study employs the same mapping as discussed in our previous study. Model annotation, abstract test cases generation, executable test case generation and automated test case execution phases are discussed in detail in this study. We have conducted two case studies and have validated the results with selected contemporary studies. Context Drive is illustrated in Figure 1." |

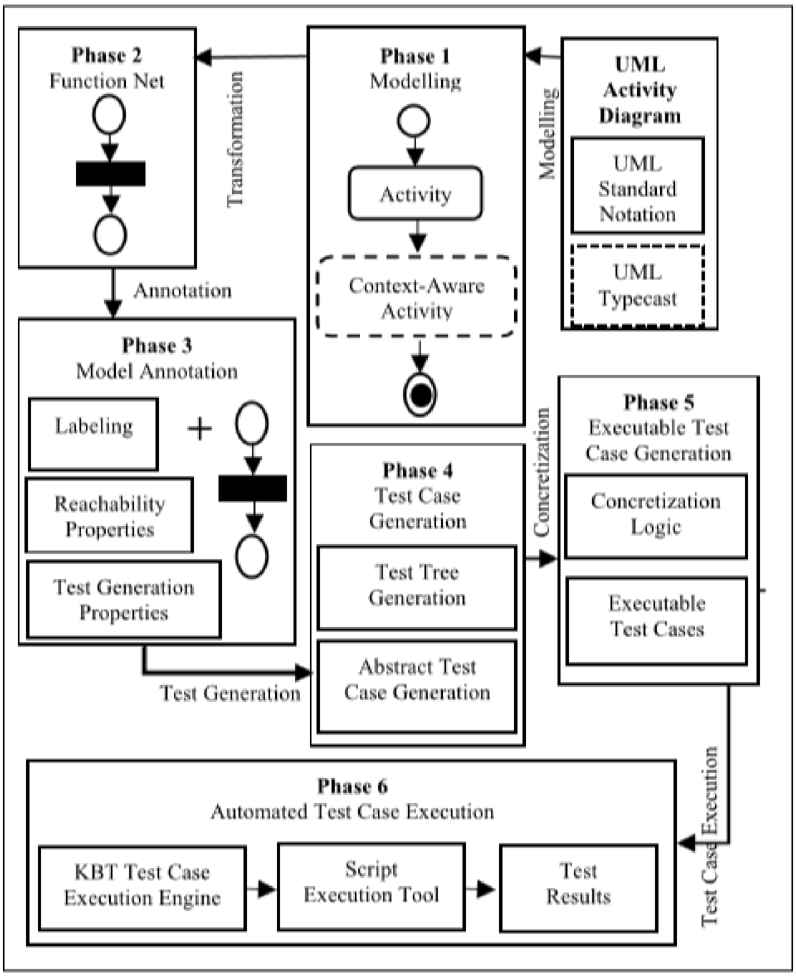


**FIGURE 1. ContextDrive - a test automation framework.**

- “Test cases generated from test model are in the form of abstract test cases, so they are platform and tool independent. Abstract test cases are human readable and can be executed manually. To execute generated test cases automatically, abstract test cases need to be converted according to tool specific test scripts referred as concrete test scripts [44].”
- “Test automation tools are categorized into four main categories: Record and Playback, Functional Decomposition, Data Driven and Keyword-driven tools. All the approaches except keyword-driven testing suffer from the issues of maintainability. Keeping in view the merits and demerits of automated testing approach employed by test automation tools, our framework supports two test script execution methods: functional decomposed test scripts and keyword-driven test scripts. We use Appium for test script execution which is an open source tool for testing mobile applications. Appium supports Windows, iOS and Android platforms which implies that the developed test scripts are reusable and

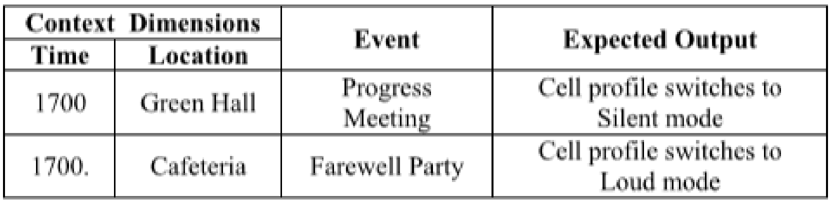
|  |  |
|---|---|
|  | can be executed across all supported platform with no or minimal changes [45]." |
| Context-Awareness Information | • "In this modern age, small and powerful smart devices are commonly being used to communicate with each other and perform complex computational tasks in concurrent fashion. The technological advancements have led to the development of new type of applications known as context-aware applications or self-adaptive applications. Context-aware applications can infer and react to their environment and adapt to situational context instantly in order to provide a better user experience. Some examples of context-aware applications include WALKPATH [3] and City Guide [4]. Context-aware applications are used in many walks of life such as healthcare, entertainment etc. Context-aware applications have several distinctive features which make these applications different from the conventional or non-context-aware applications. The foremost element that makes a context-aware application distinctive from the conventional application is the context itself. Context is a form of information detachable from an activity/action and it defines characteristics of the environment where that activity/action has taken place [5]. The context could have several dimensions and there are a number of models to identify the context dimensions. One such model is the pentagonal model proposed in [6]. This model identifies five context dimensions of an entity namely individuality, time, activity, location and relationships."<br>• "To test context-aware applications, it is important to understand these features and plan test strategy accordingly. Few important features of context-aware applications are context, quality of context, sources of context, context interpretation and reasoning."<br>• "Context information is retrieved from different sources which can be grouped into two broad categories, physical sensors and data sensors. Examples of physical sensors areGPS,heatandproximitysensorswhichareusedtoobtain location and temperature of the device as well as proximity to other neighboring devices respectively. Similarly, examples of data sensors include preferred usage profiles, social networking profiles, calendar and task list of a smartphone. However, context information retrieved from both types of sensors can introduce imperfection e.g., ambiguity, imprecision, errors/omissions about the sensed context due to many reasons such as noise or failure of sensors [7]. These imperfections in the context information may cause context-aware application to behave erroneously." |

| | |
|---|---|
| | • "Context-aware applications apply analytical and reasoning techniques for interpretation of context to identify user needs and adapt to change in the user's context accordingly. Context reasoning is the process of analyzing context data to comprehend new knowledge from raw data obtained from sensors [7]. Testing the context-aware applications is a difficult task due to many challenges such as developing test adequacy and coverage criteria, context adaptation, context data generation [8], designing context-aware test cases, developing test oracle and devising new testing techniques to test context aware applications [9], [10]. Designing, maintaining and executing context-aware test cases is a hard and time-consuming task due to high volatility of the context. In view of this, there is a need to develop an automated testing framework for context-aware applications to make the testing process efficient and effective. Addressing this research gap, hence serves as a motivation to undertake this study."<br>• "A brief account of the contributions of our study is as follows. Context adaptation cannot be modelled using standard notation of UML activity diagram; thus, we have extended UML activity diagram for behavior modeling of context-aware applications by adding a context-aware activity node. MBT facilitates automation of testing process and we have utilized MBT to generate context-coupled test cases, are search challenge which was not addressed earlier."<br>• "In this regard, Satoh [37] were constricted to evaluate their proposed approach for just one context dimension which was simply the location context. Most of the proposed techniques aim at solving a very specific problem [38] e.g., test oracle development or test case generation etc. Similarly, some testing approaches developed for context-aware applications are software platform depended which are prone to become outdated with technological advancement. For instance, MobileTest [39] was developed for Symbian platform which has now become obsolete. Yet another issue is that mutation testing techniques have been used for generating context-aware test cases despite the fact that mutation testing does not ensure correctness of the functional requirements [9]."<br>• "Several factors such as information heterogeneity and user's mobility etc. influence context information and can cause change in context at any point in time. For example, user's context could have multiple dimensions such as time and location. User's mobility perpetually affects these two context dimensions which results in changes in the surrounding objects such as people and places. Because of these changes, the context-aware application needs to adapt with the changing situations to reflect user's current context status. |

Since, context information is retrieved from many sources, therefore, it is quite possible that these sources provide the same information in different formats with varying degree of context quality [40]. In view of this, standard UML Activity diagram is too general to model context-aware applications [41] since it cannot adequately model all the adaptation factors and aspects of the user's context [42]. To overcome this shortcoming, it is recommended to use extended UML notations [40]."

- "Our target is to generate context-coupled test cases from the test model. The test cases designed for conventional software have static output. This is not true for context-aware applications where context-coupled test cases are designed to test context-adaptation. In context-coupled test cases the expected output of the test case is dynamic and changes according to the current context, even during the execution of test cases. Thus, identifying context dependent functionalities in development model is imperative to generate context-coupled test cases. Since context adaption cannot be modelled using standard notation of activity diagram; therefore, we propose to extend UML activity diagram using the stereotype mechanism of UML notation. Thus, in this study, we have used a typecast of Activity Node named context aware activity node for behavior modeling of context-aware functionalities."
- "There are several advantages of using context-aware activity node e.g., context reconfiguration points can be easily identified. Context reconfiguration points refer to those events that alter values of the context parameters. Using context reconfiguration points, different context dimensions and context activities such as jogging, driving etc. can be easily identified. After identifying context reconfiguration points, main and alternative flows of context depended functionalities are identified. Moreover, we can also identify those parameters where context change can occur so that expected output for each functional flow could be determined. Some of context factors identified for a context reconfiguration point ''meeting'' are listed in Table 3."

**TABLE 3.** **Context parameters identified for meeting.**



| Context Dimensions | | Event | Expected Output |
|---|---|---|---|
| Time | Location | | |
| 1700 | Green Hall | Progress Meeting | Cell profile switches to Silent mode |
| 1700. | Cafeteria | Farewell Party | Cell profile switches to Loud mode |

| | |
|---|---|
| Study Type | • “To validate our framework, we conducted two case studies and results of these case studies are compared with the results of selected contemporary studies.” |

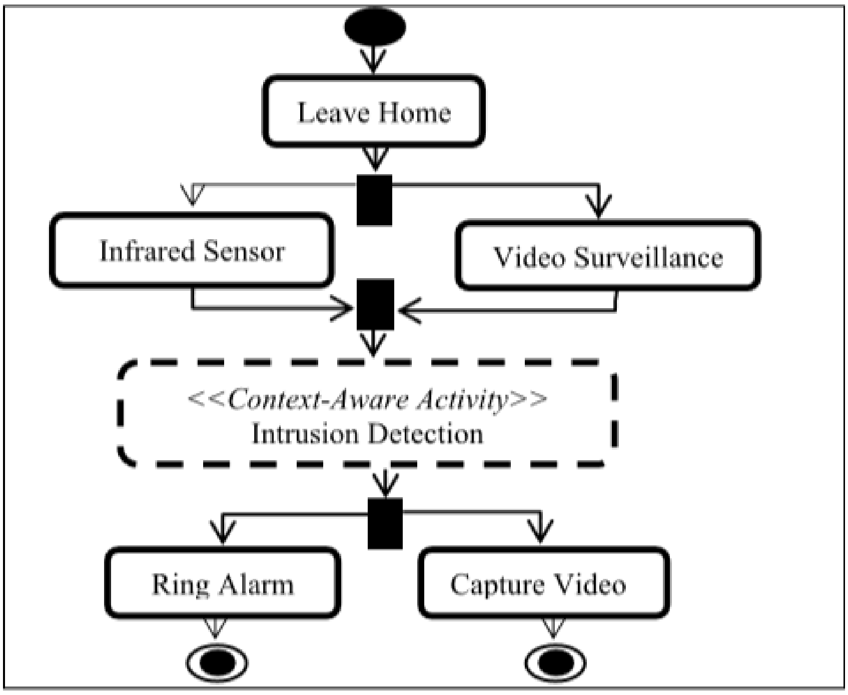
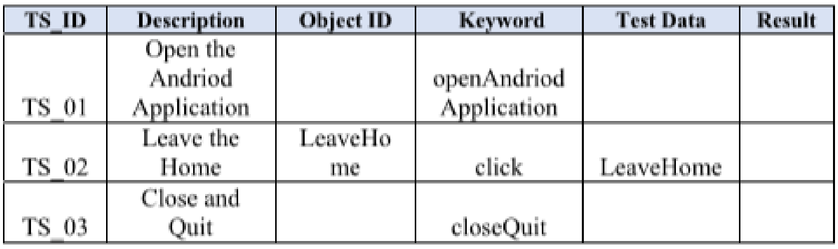
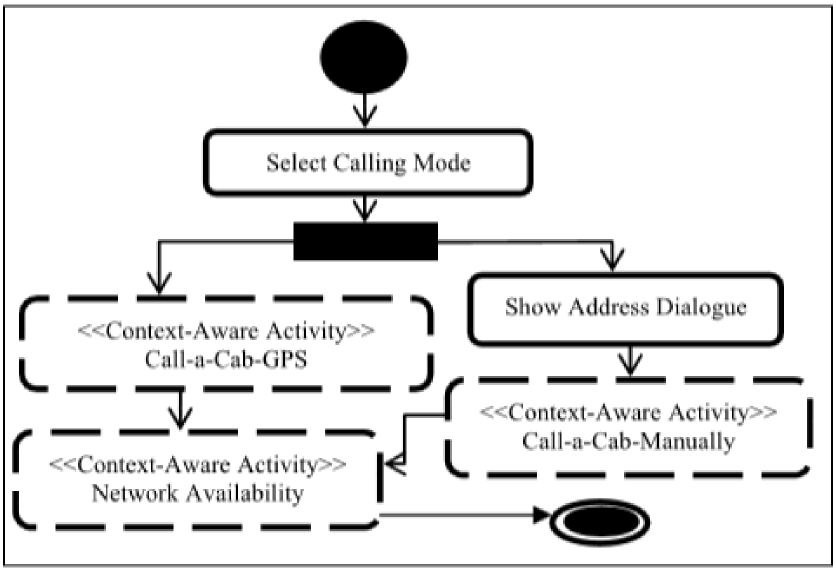
| | |
|---|---|
| Application’s Name | Smart Home Application |
| Application Description | • “Using our proposed approach, we have developed UML activity diagram for a smart home application as depicted in Figure 9.<br><br> Leave Home<br>Infrared Sensor<br>Video Surveillance<br><<Context-Aware Activity>><br>Intrusion Detection<br>Ring Alarm<br>Capture Video<br>**FIGURE 9. Context-aware activity diagram for smart home app.**<br>• “The intrusion detection system is enabled when user selects ‘‘Leave Home’’ option. The intrusion detection system is composed of video surveillance and infrared sensor. On detecting motion using infrared sensor, intrusion detection system instantiate video surveillance system and record intruder’s video. In Figure 9, Intrusion Detection is a context aware activity node.”<br>• “Keyword-driven test case is depicted in Table 8. TS_ID filed represent test step ID which is used for logging purposes. Outcome of each step (Pass/Fail) is record in the ‘‘Result’’ column. If a test step fails, test case will be marked failed.” |

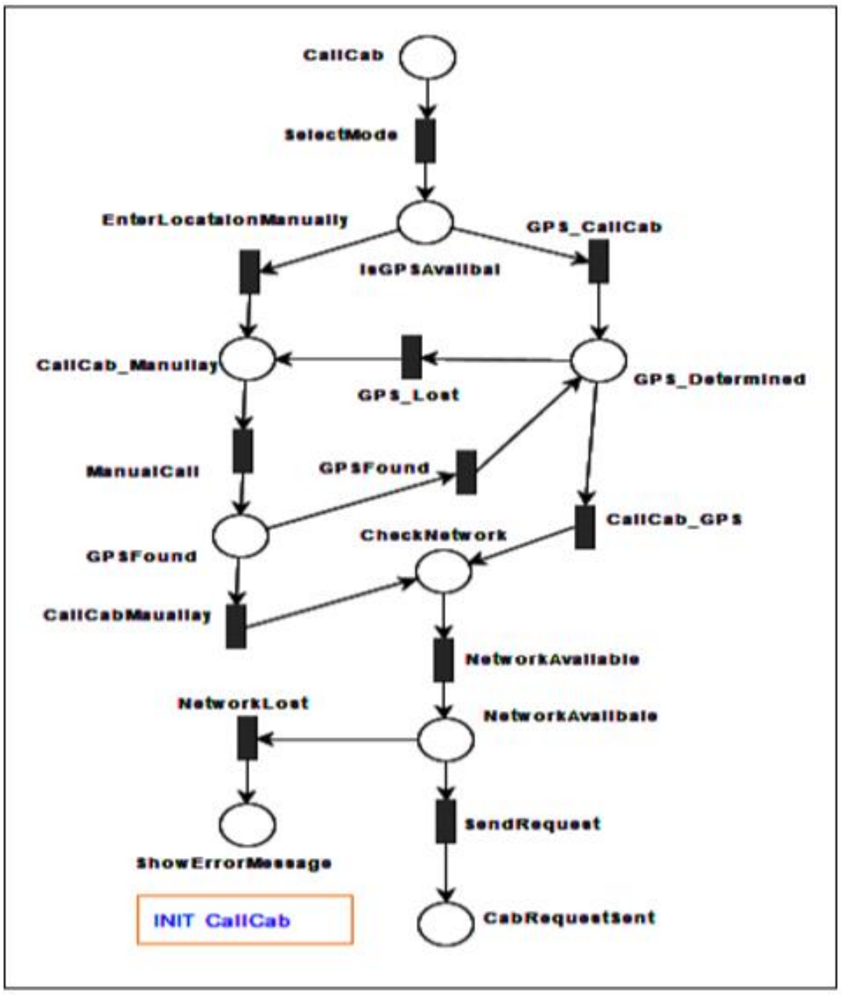
**TABLE 8. Keyword-driven test case.**



| TS_ID | Description | Object ID | Keyword | Test Data | Result |
|---|---|---|---|---|---|
| TS_01 | Open the Andriod Application | | openAndriod Application | | |
| TS_02 | Leave the Home | LeaveHome | click | LeaveHome | |
| TS_03 | Close and Quit | | closeQuit | | |

| Application's Name | Call-a-Cab App |
|---|---|
| Application Description | - "Our first case study is based on call-a-cab context-aware application [47]. This application allows users to call a cab to their current location. User location can be obtained using GPS sensor or can be fed manually. If application fails to automatically obtain GPS location, then it reverts to manual mode requiring the user to feed the location. Testing this application requires test cases to include location determination modes, setting valid and invalid location and manipulation the network connection to simulate unexpected service loss."<br>- "Firstly, we develop UML activity diagram for Call-A-Cab app using our proposed modeling notations as depicted in Figure 12."<br><br> Select Calling Mode<br>Show Address Dialogue<br><<Context-Aware Activity>><br>Call-a-Cab-GPS<br><<Context-Aware Activity>><br>Call-a-Cab-Manually<br><<Context-Aware Activity>><br>Network Availability<br>**FIGURE 12. Activity diagram for Call-A-Cab.** |

| | |
|---|---|
| | • “As evident from Figure 12, we identify three context reconfiguration points (Call-a-Cab-GPS, Call-a-Cab-Manually and Network Available) where application needs to collect current context to carry out further functionality. While calling a cab using GPS, if GPS connection fails then application will fall back to manual mode. Similarly, while calling cab manually, if GPS location is found, application will fall back to automatic mode. After obtaining user location,application needs to send cab request using cellular network. If cellular network is lost, then an error dialog will be displayed otherwise request will be sent.”<br>• “In the second phase, we transform UML activity diagram into function net and import it in MISTA. We annotated our model in the next phase and added initial marking along with labeling function net elements as depicted in Figure 13.”<br>• “In the fourth phase, we generated test suite consisting of 91 test cases. Out of these 91 test cases, we had 5 test cases with valid path (positive test cases) and 88 test cases with invalid path (negative test cases). Depth of the deepest test was 6.” |

| | |
|---|---|
| | <br> CallCab<br>SelectMode<br>EnterLocatalonManually<br>GPS_CallCab<br>IsGPSAvalibal<br>CallCab_Manullay<br>GPS_Lost<br>GPS_Determined<br>ManualCall<br>GPSFound<br>CallCab_GPS<br>GPSFound<br>CheckNetwork<br>CallCabMaually<br>NetworkAvailable<br>NetworkLost<br>NetworkAvalibale<br>SendRequest<br>ShowErrorMessage<br>INIT CallCab<br>CabRequestSent<br>**FIGURE 13. Function net to call a Cab.** |

| Application's Name | Smart Home App |
|---|---|
| Application Description | • "Our proposed solution can also be used with complex models. In the second case study, we selected a Smart Home app which is a quite complex app. We developed UML activity diagram of smart home application shown in Figure 14. This UML activity diagram is adapted from [48] for experimentation in our study. When a user selects ''Leave Home'' mode, Fire Protection and Security Systems are enabled. Fire Protection system is composed of three sensors, SMOG sensor, Gas sensor and Temperature sensor. These three sensors provide continuous input to context activity node named ''FireDetection''. If fire is detected,then an alert SMS is sent on the predefined cell number and fire extinguisher is |

also instigated. Security system of smart home app has already been elaborated in section 4.1”

- “Smart home app senses environment and drives context as ‘‘Normal’’, ‘‘Intrusion’’ or ‘‘Fire’’. Context can change suddenly, and an adaptation can occur at any time from ‘‘Normal’’ context to ‘‘Fire’’ or ‘‘Intrusion’’ context and vice versa. It is also possible that due to erroneous input from the sensors, adaptation of ‘‘Intrusion’’ or ‘‘Fire’’ occur and alarm is triggered and abruptly ‘‘Normal’’ context can occur when sensor does not detect any further intrusion or fire.”
- “In the second phase, we transformed development model into testing model using our own algorithm. We have imported this function net in our test modeling tool. Imported test model is depicted in Figure 15. It can be observed that elements of generated test model need to be labeled to enhance readability. Thus, in phase 3, we perform model annotation.”

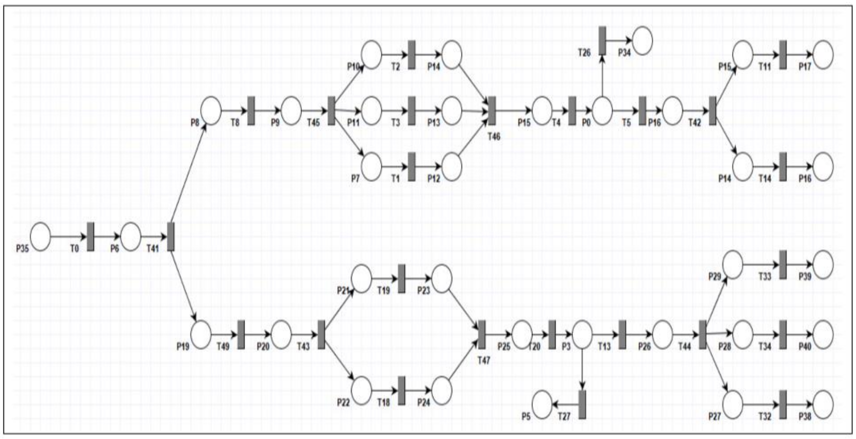


FIGURE 15. A complex testing model of smart home app.

- “In this phase, we labeled places and transition as well as add Initial Marking and Goal States in our model. It can be noticed that generated test model is complex and difficult to understand and simulate. Therefore, we need to decompose this large model into smaller modules. These small modules are called subnets in function nets.”
- “MISTA supports decomposing large and complex function nets into hierarchal function nets comprising one main function and many sub-function nets. Using this functionality of MISTA, we decomposed our complex model into hierarchal function net composed of one main function net and two sub-function nets. Main function net is depicted in Figure 16. It can be observed that IntrusionSensors (shown in circle) and FireSensor transitions (shown in rectangle) indicating that these transitions are entry point of subnets. Initial marking defines starting point of test model for simulation and test tree generation. Test case is represented as a sequence of nodes from root node to leaf nodes of the test tree.”

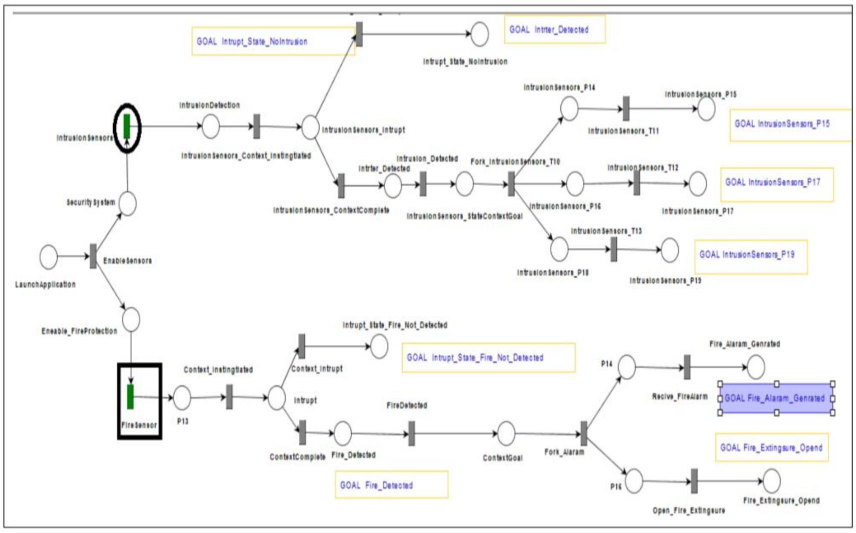


**FIGURE 16.** Main function net.

- “FireSensor transition (shown in rectangle in Figure 16) leads to subnet labeled Fire Protection System which is depicted in Figure 17.”

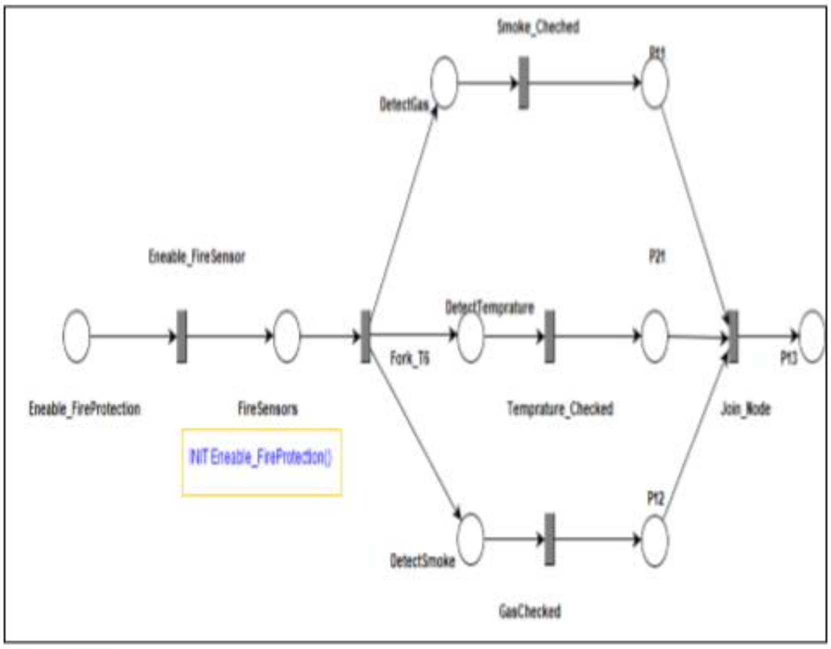


**FIGURE 17.** Fire protection system.

- “IntrusionSensors transition (encircled in Figure 16) leads to subnet labeled as Intrusion Detection System and is depicted in Figure 18.”
-

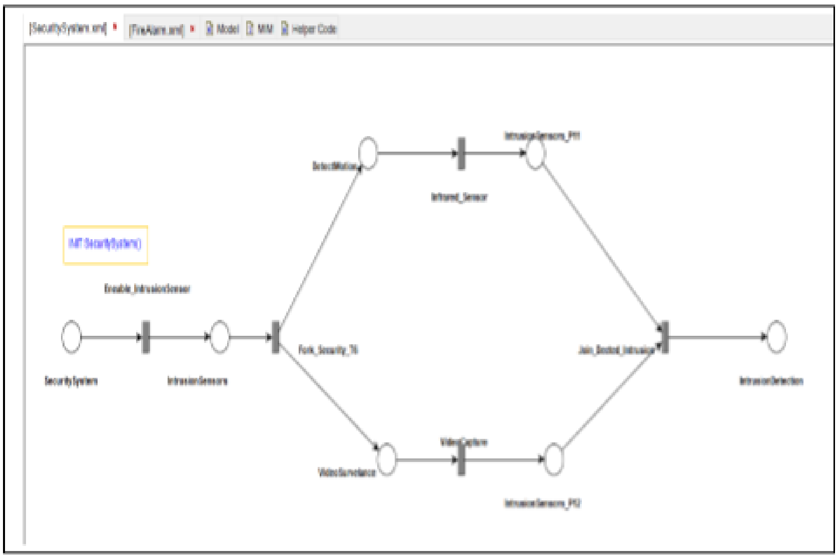
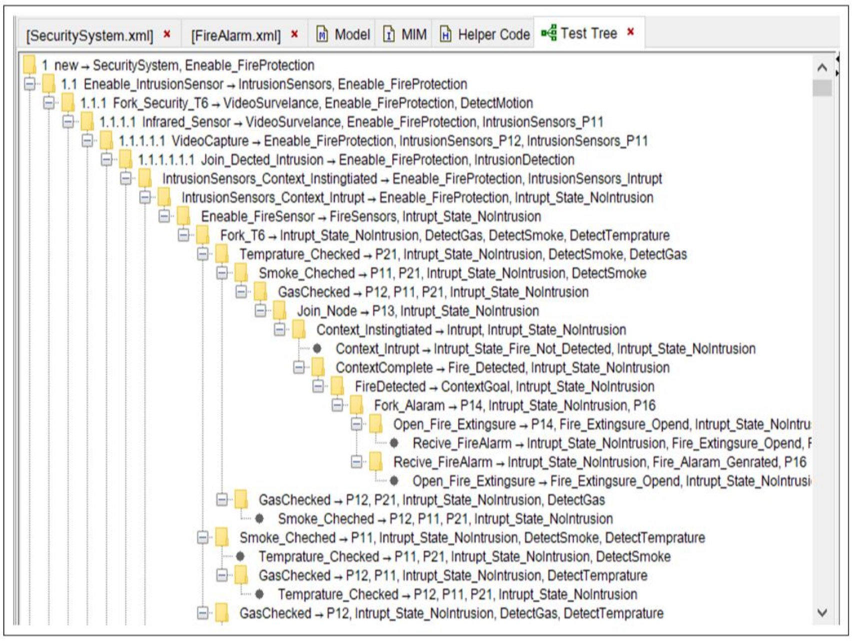


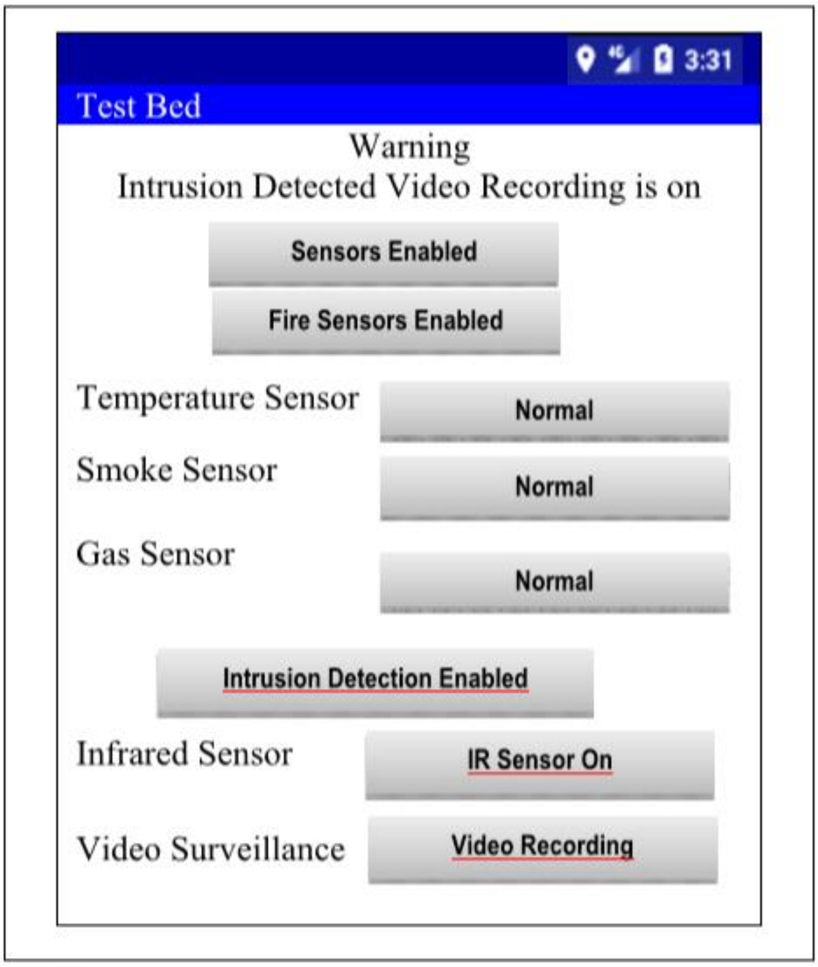
**FIGURE 18. Securitys system.**

- "In phase 4.4, we generated abstract test cases from our test model. For test case generation, we have used breadth first search algorithm and set maximum depth of test tree parameter to 100. Based on these parameters, test tree comprising 538 test cases spanning on 361 states was generated as depicted in Figure 19. The depth of the deepest test was 24. In phase 5, we transformed abstract test cases into concrete test scripts. In phase 6, we executed concrete test case on our SUT. We generated an intrusion situation for SUT. Our smart home application detected intrusion and generated a warning. Output of our SUT for intrusion detection is shown in Figure 20."



```
1  new → SecuritySystem, Eneable_FireProtection
  1.1  Eneable_IntrusionSensor → IntrusionSensors, Eneable_FireProtection
    1.1.1  Fork_Security_T6 → VideoSurvelance, Eneable_FireProtection, DetectMotion
      1.1.1.1  Infrared_Sensor → VideoSurvelance, Eneable_FireProtection, IntrusionSensors_P11
        1.1.1.1.1  VideoCapture → Eneable_FireProtection, IntrusionSensors_P12, IntrusionSensors_P11
          1.1.1.1.1.1  Join_Dected_Intrusion → Eneable_FireProtection, IntrusionDetection
            IntrusionSensors_Context_Instingtiated → Eneable_FireProtection, IntrusionSensors_Intrupt
              IntrusionSensors_Context_Intrupt → Eneable_FireProtection, Intrupt_State_NoIntrusion
                Eneable_FireSensor → FireSensors, Intrupt_State_NoIntrusion
                  Fork_T6 → Intrupt_State_NoIntrusion, DetectGas, DetectSmoke, DetectTemprature
                    Temprature_Checked → P21, Intrupt_State_NoIntrusion, DetectSmoke, DetectGas
                      Smoke_Cheched → P11, P21, Intrupt_State_NoIntrusion, DetectSmoke
                        GasChecked → P12, P11, P21, Intrupt_State_NoIntrusion
                          Join_Node → P13, Intrupt_State_NoIntrusion
                            Context_Instingtiated → Intrupt, Intrupt_State_NoIntrusion
                              ● Context_Intrupt → Intrupt_State_Fire_Not_Detected, Intrupt_State_NoIntrusion
                              ContextComplete → Fire_Detected, Intrupt_State_NoIntrusion
                                FireDetected → ContextGoal, Intrupt_State_NoIntrusion
                                  Fork_Alaram → P14, Intrupt_State_NoIntrusion, P16
                                    Open_Fire_Extingsure → P14, Fire_Extingsure_Opend, Intrupt_State_NoIntru
                                      ● Recive_FireAlarm → Intrupt_State_NoIntrusion, Fire_Extingsure_Opend, F
                                    Recive_FireAlarm → Intrupt_State_NoIntrusion, Fire_Alaram_Genrated, P16
                                      ● Open_Fire_Extingsure → Fire_Extingsure_Opend, Intrupt_State_NoIntrusi
                      GasChecked → P12, P21, Intrupt_State_NoIntrusion, DetectGas
                        ● Smoke_Cheched → P12, P11, P21, Intrupt_State_NoIntrusion
                    Smoke_Cheched → P11, Intrupt_State_NoIntrusion, DetectSmoke, DetectTemprature
                      ● Temprature_Checked → P11, P21, Intrupt_State_NoIntrusion, DetectSmoke
                      GasChecked → P12, P11, Intrupt_State_NoIntrusion, DetectTemprature
                        ● Temprature_Checked → P12, P11, P21, Intrupt_State_NoIntrusion
                    GasChecked → P12, Intrupt_State_NoIntrusion, DetectGas, DetectTemprature
```

**FIGURE 19. Test tree generation.**



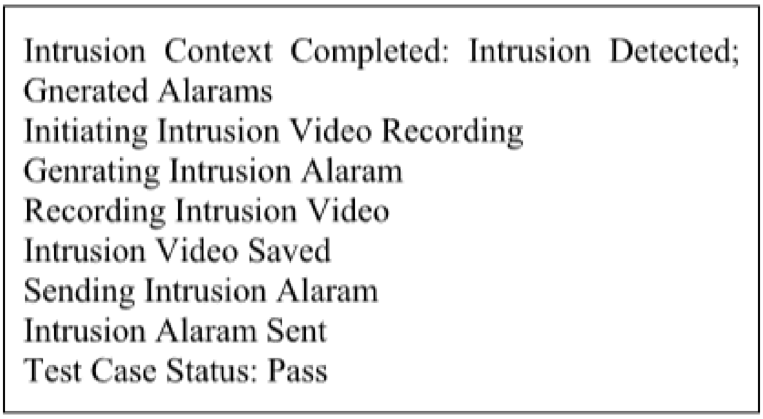
Warning
Intrusion Detected Video Recording is on

Temperature Sensor Normal
Smoke Sensor Normal
Gas Sensor Normal

Infrared Sensor IR Sensor On
Video Surveillance Video Recording

**FIGURE 20. Intrusion detected.**

- “When context-aware smart home application detects an intrusion at home, it records video and sends alarm a san SMS on predefined cell phone. Output of the executed test case is depicted in Figure 21.”

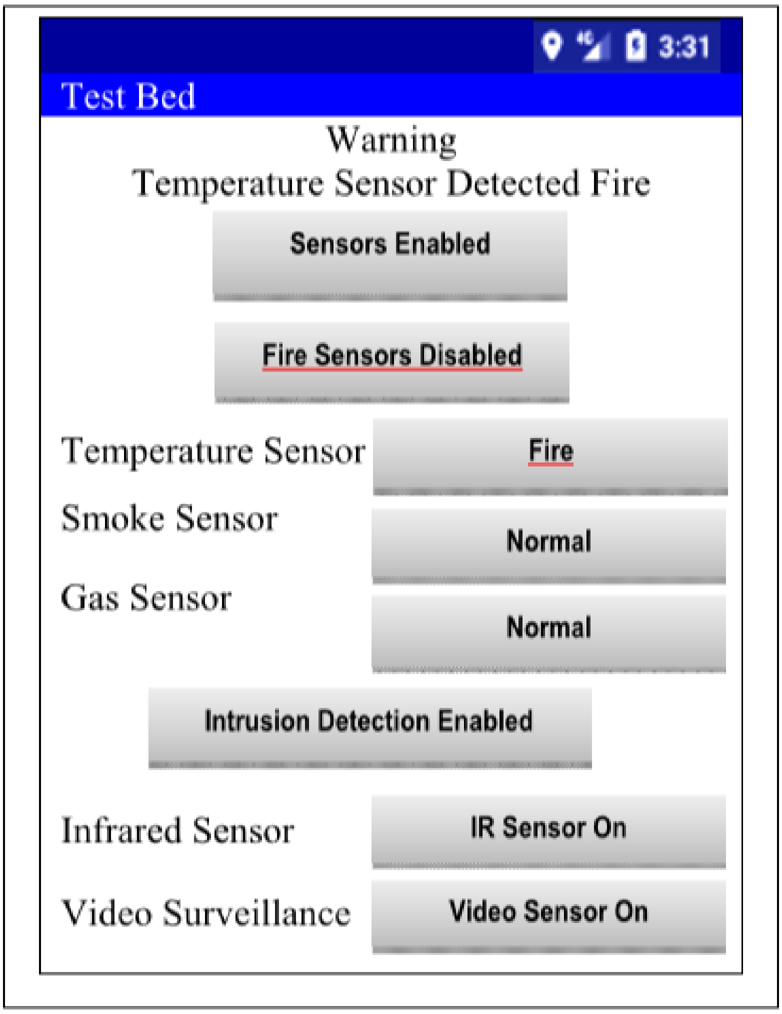


```
Intrusion Context Completed: Intrusion Detected;
Gnerated Alarams
Initiating Intrusion Video Recording
Genrating Intrusion Alaram
Recording Intrusion Video
Intrusion Video Saved
Sending Intrusion Alaram
Intrusion Alaram Sent
Test Case Status: Pass
```

**FIGURE 21. Output of test case on intrusion detection.**

- “Context can change suddenly, and context change during execution of test case can cause a test case to fail due to mismatch of expected and actual output. Therefore, a context coupled test case should cater for context change during execution of test case."

- "Test cases generated through our proposed framework are context-coupled and can cater for context changes during execution of test case. To demonstrate this capability, we provide an example of false fire alarm."
- "Suppose temperature sensor detected high temperature due to some glitch thus a warning message is displayed at user interface of mobile application and label of ''Temperature'' sensor changed to ''Fire'' (Figure 22)."



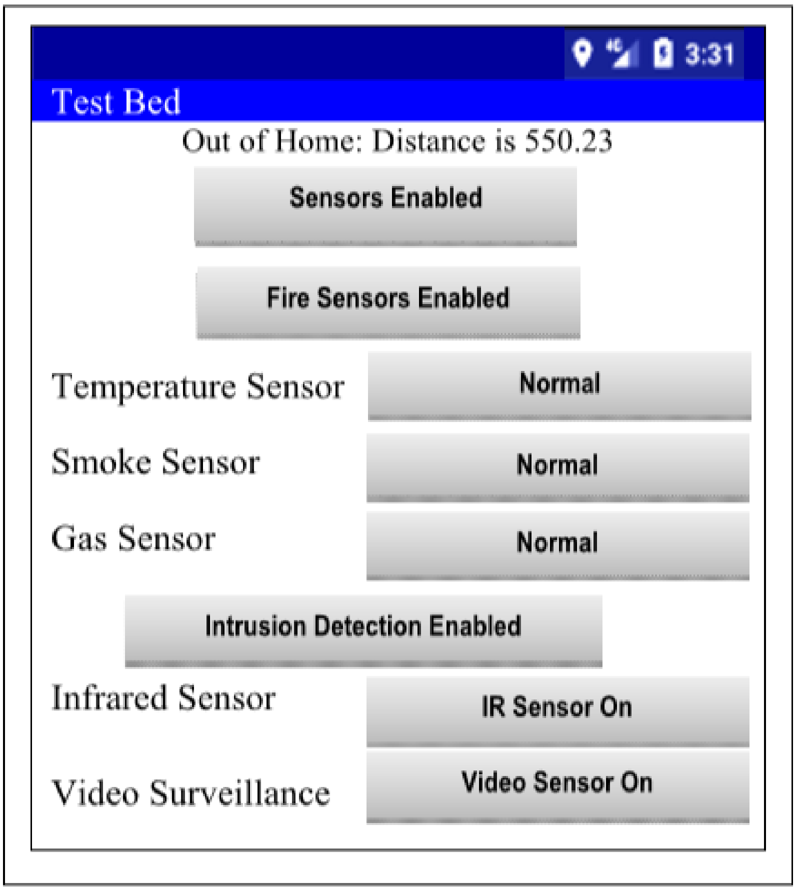
Test Bed

Warning
Temperature Sensor Detected Fire

Temperature Sensor: Fire
Smoke Sensor: Normal
Gas Sensor: Normal
Infrared Sensor: IR Sensor On
Video Surveillance: Video Sensor On

**FIGURE 22. Temperature sensor detected fire.**

- "Meanwhile temperature sensor detected normal temperature and reverted to ''Normal'' status as depicted in Figure 23."



3:31

Test Bed

Out of Home: Distance is 550.23

Sensors Enabled

Fire Sensors Enabled

Temperature Sensor: Normal

Smoke Sensor: Normal

Gas Sensor: Normal

Intrusion Detection Enabled

Infrared Sensor: IR Sensor On

Video Surveillance: Video Sensor On

**FIGURE 23. Temperature sensor revert to normal situation.**

- "This caused sudden context change and to adapt to current context, smart home application should abort alarm generation functionality. We have executed our test cases to test this functionality. Figure 24 shows output of test case."
- "From Figure 24, it can be observed that execution logo four test case indicates that context was changed during execution of test case. Since our test cases are context-coupled thus expected output of test cases also changed accordingly with respect to the context. Therefore, status of test case was set to ''Pass'' as the context change was handled properly.Whereas, with context-decoupled test cases, status of test case would have been marked fail. As a result, test execution log would incorrectly indicate a defect in the application functionality."

## A.2 Software adaptation in wireless sensor networks.

| Paper ID | #24 |
|---|---|
| Bibliography: | Afanasov, M., Mottola, L., & Ghezzi, C. (2018). Software adaptation in wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *12*(4), . |
| Abstract: | "We present design concepts, programming constructs, and automatic verification techniques to support the development of adaptive Wireless Sensor Network (WSN) software. WSNs operate at the interface between the physical world and the computing machine and are hence exposed to unpredictable environment dynamics. WSN software must adapt to these dynamics to maintain dependable and efficient operation. However, developers are left without proper support to develop adaptive functionality in WSN software. Our work fills this gap with three key contributions: (i) design concepts help developers organize the necessary adaptive functionality and understand their relations, (ii) dedicated programming constructs simplify the implementations, (iii) custom verification techniques allow developers to check the correctness of their design before deployment. We implement dedicated tool support to tie the three contributions, facilitating their practical application. Our evaluation considers representative WSN applications to analyze code metrics, synthetic simulations, and cycle-accurate emulation of popular WSN platforms. The results indicate that our work is effective in simplifying the development of adaptive WSN software; for example, implementations are provably easier to test and to maintain, the run-time overhead of our dedicated programming constructs is negligible, and our verification techniques return results in a matter of seconds." |
| General Information: | ● "Wireless Sensor Networks (WSNs) bridge the gap between the physical world and the computing machine [30] by seamlessly gathering data from the environment through sensors, and by taking actions on it through actuators. Because of their intimate interactions with the physical world, WSNs are exposed to multiple and unpredictable environment dynamics that affect their operation."<br>● "Multiple environmental dimensions evolve concurrently and independently, such as location and battery levels. WSN software needs to adapt to such dynamics to maintain efficient performance. For example, in wildlife tracking, the inability to adapt to different situations may result in |

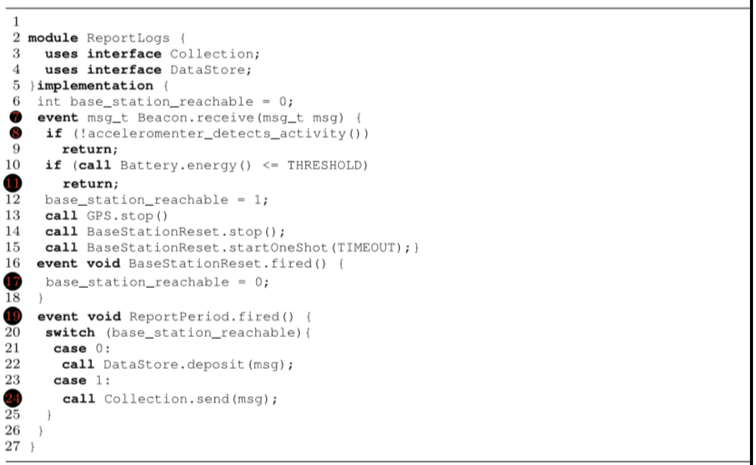
| | |
|---|---|
| | earlier battery depletion, preventing WSN nodes to eventually upload sensor data to the base-stations and thus hampering the analysis."<br>● " We extend nesC with notions of Context-oriented Programming (COP) [28]. Section 4 describes the resulting language, called ConesC, which ameliorates the coupling between functionality, rendering implementations easier to understand and to maintain. The design concepts of Section 3 map to the programming constructs we introduce, easing the transition from design to implementation"<br>● "COP is a programming paradigm often employed to implement adaptive software. Central to COP is the notion of layered function, that is, a function whose behavior changes depending on the current situation and transparently to the caller. COP already proved effective in creating adaptive software in mainstream applications, such as user interfaces [34] and text editors [32]. In these settings, programmers rely on COP extensions of popular high-level languages, such as Java [58]."<br>● |
| Context-Awareness Information: | ● "Our work is centered on a notion of context. Such a notion is vastly employed in various areas of computing, including proximate selection, contextual reconfiguration, contextual information, and context-triggered actions [1, 14, 56], yet not in WSN software. **We specifically consider a context to be a specific situation, including both environmental and system features,that WSN software might find itself in. This is similar to the notion of "situation" employed in context aware computing [13]."**<br>● "We introduce two key concepts: (i) individual contexts, and (ii) context groups. A context represents an individual situation the software running on a given WSN device may encounter [13]. Whenever that situation occurs, the software changes its functioning accordingly, implementing an appropriate adaptation decision. For example, in the wildlife-tracking application described in the Introduction, the reachability of the base-station based on the physical location of a device represents an individual context coupled to a corresponding functionality. This is different to the context and functionality representing the situation where the base-station is unreachable. A context group is a collection of contexts sharing common characteristics, for example, being determined by the same environment dimension. We may group together the two contexts representing the (un)reachability of the base-station, as both depend on a device's physical location." |

| | |
|---|---|
| | ● |
| Study Type: | |

| | |
|---|---|
| Application's Name | |
| Application Description | ● "Consider the use of WSNs to track wildlife [49]. Battery-powered WSN nodes are embedded in collars attached to animals, such as zebras or badgers. The devices are equipped with sensors to track the animals' movement, for example, based on GPS and accelerometer readings, and to detect their health conditions, for example, based on body temperature. Low-power short-range radios are used as proximity sensors by allowing nodes to discover each other whenever they are within communication range, using a form of periodic radio beaconing. A node logs the radio contacts to track an animal's encounters with other animals, enabling the study of their social interactions. The radio is also used to off-load the contact traces when in reach of a fixed base-station. Small solar panels harvest energy to prolong the node lifetime [5]."<br>● "Using battery-powered WSN devices makes energy a precious resource that developers need to trade against the system functionality, depending on the situation. For example, GPS sampling consumes non-negligible energy. The difference between consecutive GPS readings may be taken as an indication of the pace of movement, and used to tune the GPS sampling frequency and granularity. The contact traces can be sent directly to the base-station whenever the latter is within radio range, but they need to be stored locally otherwise. When the battery is running low, developers may disable GPS sampling to make sure the node survives until the next encounter with a base-station, not to lose the collected contact traces."<br>● "To give a concrete feeling of the issues at stake, Figure 1 shows a simplified implementation of adaptive functionality using nesC [24], a dialect of C commonly used for WSN development. NesC function calls are asynchronous; results are returned using a notion of event |

that essentially operates as a callback. The code implements only one aspect of the adaptation needed in wildlife tracking: to send readings to the base-station whenever reachable, or to store them locally otherwise.”

- “In Figure 1, multiple orthogonal concerns are intertwined and functionality are tightly coupled. For example, the decision on what operating mode to employ, that is, whether to consider the base-station as reachable, is implemented from line 19 to 24 . This lies in the the same module as the adaptive processing itself from line 7 to 17 . Both functionality depend on the same global variable base_station_reachable, whose management is entirely on the programmer’s shoulders.”



```
1
2 module ReportLogs {
3   uses interface Collection;
4   uses interface DataStore;
5 }implementation {
6  int base_station_reachable = 0;
7  event msg_t Beacon.receive(msg_t msg) {
8   if (!accelerom enter_detects_activity())
9     return;
10  if (call Battery.energy() <= THRESHOLD)
11    return;
12  base_station_reachable = 1;
13  call GPS.stop()
14  call BaseStationReset.stop();
15  call BaseStationReset.startOneShot(TIMEOUT);}
16 event void BaseStationReset.fired() {
17  base_station_reachable = 0;
18 }
19 event void ReportPeriod.fired() {
20  switch (base_station_reachable){
21   case 0:
22    call DataStore.deposit(msg);
23   case 1:
24    call Collection.send(msg);
25   }
26  }
27 }
```

Fig. 1. Example nesC implementation of adaptive functionality. *Several orthogonal functionality be tangled and need to share global data.*

- “Moreover, the checks to perform before changing operating mode, such as those in lines 8 and 11, are mixed with the functionality that changes the mode itself.”
- “Figure 2 represents the complete design of the wildlife tracking application based on contexts and context groups. The four context groups, shown as the outer boxes, represent collections of individual contexts depending on battery level, base-station reachability, as well as an animal’s health conditions and activity levels. The individual contexts, shown as the inner boxes in every group, are described by a name and by actions taken when entering or leaving a context, and by processing executing as long as the context is active, that is, the context corresponds to the current situation. Context and context groups provide structure and help factor out the adaptation necessary to deal with independent environment dimensions.”
- “At most one context is active in each context group at any point in time. However, multiple contexts belonging to

different groups may be active at the same time. Contexts within the same group are tied with transitions that express the conditions triggering a change of the current context. In Figure 2, for example, a change in the battery voltage below a threshold triggers a change from the Normal to the Low context in the Battery group. The evolution of active contexts in different groups thus mimics the semantics of parallel state machines, but for the following features:

  - Context transitions may contain dependencies. For example, if a body sensor reads an abnormal temperature, it might indicate that the animal is Diseased, and require a transition to the corresponding context. In this situation, however, an animal is most probably moving slightly or not at all; therefore, the active context in the Activity group should not be Running.
  - Context activation may also trigger a transition in a different context group, as is the case in the Reachable context of Figure 2. Because the base-station is deployed at a known location, its reachability indicates the device is nearby. Therefore, we trigger a transition to the NotMoving context in the Activity group to disable GPS tracking and assume the base-station location as the one of the device.”

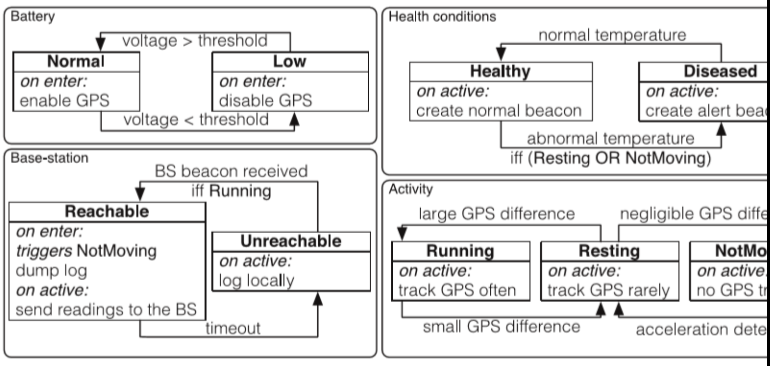


Fig. 2. Context-oriented model for a wildlife tracking application.

- “Based on experience, we observe distinct patterns emerging that provide structured ways to address specific types of adaptive functionality. These patterns, discussed next, allow developers to express complex functionality with only a handful of concepts.”
- “Behavior control. Different behaviors of the same high-level functionality are often represented in a single context group. Figure 2 shows one such example in the Base-station group, which includes two different behaviors for

| | |
|---|---|
| | the same high-level functionality of processing the collected logs. The same pattern is found also in other applications. For example, an adaptive protocol stack [22, 25] uses different protocols for the same underlying physical layer depending on node's mobility. The high-level packet relay functionality is expressed with a similar design, as we show in Section 7."<br>● "Figure 3 shows an abstract view of the behavior control pattern and its characterizing elements. Developers define a single context group to export a functionality whose behavior depends on the active context. An external context "controller" drives the transitions between the contexts in the group. In the wildlife tracking application, for example, the context controller checks if beacons are received indicating a nearby base-station, and accordingly activate a specific context in the Base-station group."<br>● "Content provider. We also observe cases where context-dependent data is offered to other functionality with little to no processing involved, differently from the behavior control pattern that provides non-trivial context-dependent processing. An example is in the Health conditions group of Figure 2. Depending on the active context, the periodic beacon is generated differently. The actual processing that involves the beacon happens elsewhere in the system; in this case, throughout the network stack responsible for transmitting the beacon over the air. We notice this pattern in other applications as well. For example, the smart-home application we describe in Section 7 employs the same pattern to manage user preferences depending on time of the day.<br>● The characterizing elements, abstractly shown in Figure 4, differ from those of behavior control. The "controller" component is often fairly trivial. For example, the "controller" in the smart-home application of Section 7 simply checks the time of the day. Differently, the component consuming the context-dependent data plays a key role. While functionality structured as behavior control can be considered stand-alone, the content provider needs to be tailored to the data consumer." |

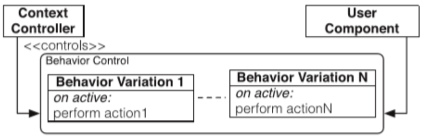


Fig. 3. Behavior control pattern.

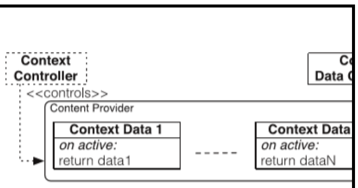


Fig. 4. Content provider pattern.

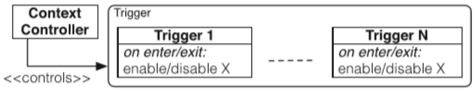


Fig. 5. Trigger pattern.

- “Trigger. We also recognize designs where contexts are used only to trigger specific operations, especially on hardware components, without any significant context-dependent processing or data offered. An example is the Battery group in Figure 2. The contexts in the group are used to enable or disable the GPS sensor depending on the battery level. In the smart-home application of Section 7, we notice a similar pattern when tuning lights in a room. Depending on the amount of natural light, different contexts are activated that tune the artificial lighting accordingly. As shown in Figure 5, the “controller” drives context transitions similar to behavior control. However, unlike the other patterns, there is no external components that either uses context dependent functionality or consumes context-dependent data.”

# Appendix B - Parasite Watch - CATS# v2

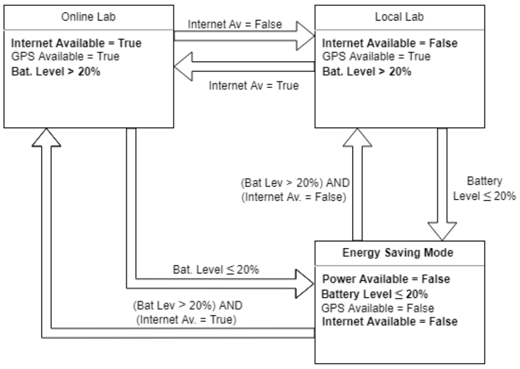
Steps 1 and 2 – Extract and Identify Context Variables:

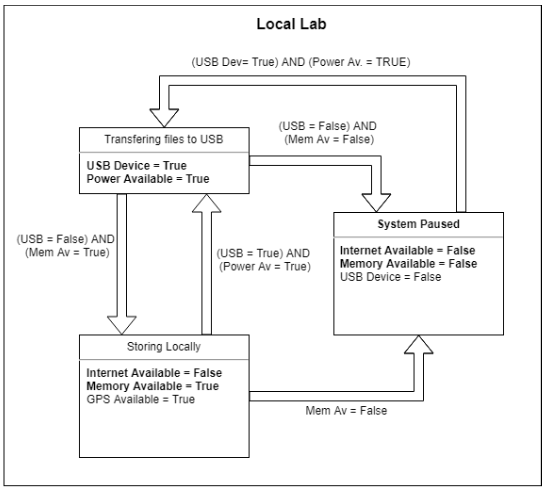
| Variables List |
|---|
| • **Internet Available**<br>• Diagnosis<br>• **Power Available**<br>• Updated<br>• **USB Device**<br>• **Battery Level**<br>• Geolocation<br>• Date<br>• Synchronized<br>• Time<br>• Image<br>• Point of Interest<br>• **Memory Available** |
| **Bold: Context Variables**<br>**Bold and underlined: Later Included** |

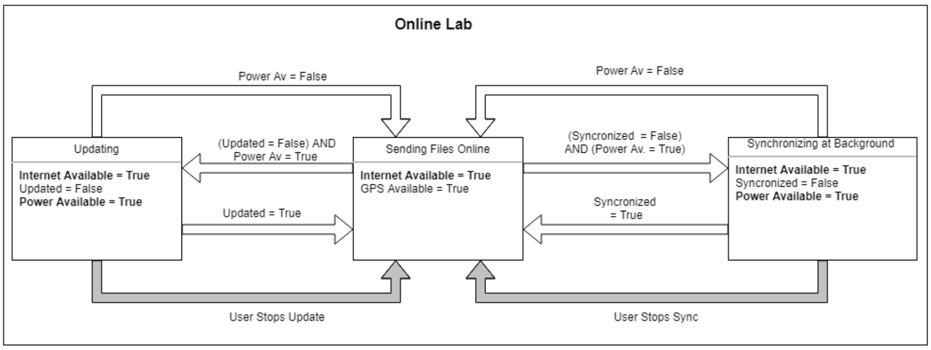
Step 3 – Generate Analytical Model

| **CV** | **THR** | **Effect** |
|---|---|---|
| Internet Available | True | Use Online Lab to make the diagnosis<br>If Synchronized = False, sync diagnosis<br>If Updated = False, update de Software |
| | False | Use Local Lab to make the diagnosis<br>If Synchronized = False, stop sync<br>If Updated = False, stop updating |
| Power Available | True | Use energy from Power |
| | False | Use energy from Battery or Solar Panel<br>If Synchronized = False, do not sync<br>If Updated = False, do not update<br>If USB Dev. = True, do not transfer files |
| Battery Level | > 20% | If Internet Av. = True: Online Lab Mode<br>If Internet Av. = False: Local Lab Mode |
| | ≤ 20% | Disable GPS<br>Disable Internet |
| Memory. Available | False | Display a message and pause the system |
| Memory Drive | True | Transfer files to Memory Drive |

Step 4 – Generate Conceptual Model







Step 5: Mapping Functionalities and Context Situations

| **System Features** | **Transferring files to USB** | **System Paused** | **Storing Locally** |
|---|---|---|---|
| Capture Image | False | False | True |
| Submit to Local Diagnosis | False | False | True |
| View Diagnosis | False | False | True |
| Transfer to Mem. Drive | True | False | False |

| **System Features** | **Updating** | **Sending Files Online** | **Synchronizing at Background** |
|---|---|---|---|
| Capture Image | False | True | True |
| Submit to Online Diagnosis | False | True | True |
| View Diagnosis | False | True | True |
| Update Software | True | False | False |
| Sync with the database | False | False | True |

| **System Features** | **Local Lab** | **Online Lab** | **Energy Saving Mode** |
|---|---|---|---|
| Capture Image | True | True | True |
| Submit to Local Diagnosis | True | False | True |
| Submit to Online Diagnosis | False | True | False |
| View Diagnosis | True | True | True |
| Capture Geolocation | True | True | False |
| Enable Local Lab's Features | True | False | False |
| Enable Online Lab's Features | False | True | False |

Step 6: Describe the Test Oracles

**Local Lab**

| **Id** | **System Feature** | **Input (I)** | **Conditions (C)** | **Expected Result (E)** |
|---|---|---|---|---|
| 01 | Capture Geolocation | User’s Location | Battery Level = 21% -> 19% | Disable GPS (Energy Saving Mode) |
| 02 | Transfer to Memory Drive | | USB = True;<br>Battery Level = 21% -> 19% | Interrupt transferring the files (Energy Saving Mode) |

- **Storing Locally**

| **Id** | **System Feature** | **Input (I)** | **Conditions (C)** | **Expected Result (E)** |
|---|---|---|---|---|
| 03 | Register Blade | Blade Id, Date | Mem Av -> Mem Not Av | System Paused |
| 04 | Capture Image | Image | Mem Av -> Mem Not Av | System Paused |
| 05 | Submit to Local Diagnosis | Image | Mem Av -> Mem Not Av | System Paused |
| 06 | View Diagnosis | Diagnosis Id | Mem Av -> Mem Not Av | System Paused |
| 07 | Register Blade | Blade Id, Date | Power Av = True;<br>USB False -> True | Transfer to Memory Drive |
| 08 | Capture Image | Image | Power Av = True;<br>USB False -> True | Transfer to Memory Drive |
| 09 | Submit to Local Diagnosis | Image | Power Av = True;<br>USB False -> True | Submit to Local Diagnosis Start Transfer to Memory Drive |
| 10 | View Diagnosis | Diagnosis Id | Power Av = True;<br>USB False -> True | Transfer to Memory Drive |

- **System Paused**

| **Id** | **System Feature** | **Input (I)** | **Conditions (C)** | **Expected** |
|---|---|---|---|---|

|  |  |  |  | Result (E) |
|---|---|---|---|---|
| 11 | Transfer Files to Pendrive |  | Power Av. = True,<br>USB presence = False -> True | Transferring Files to USB |
| 12 | Transfer Files to Pendrive |  | Power Av. =False,<br>USB presence = False -> True | Do not transfer files to USB |

- **Transferring Files to USB**

| Id | System Feature | Input (I) | Conditions (C) | Expected Result (E) |
|---|---|---|---|---|
| 13 | Transfer Files to Pendrive |  | Mem Av. = False<br>USB presence = True -> False | System Paused |
| 14 | Transfer Files to Pendrive |  | USB presence = False;<br>Mem Not Av -> Av | Finish transferring the files and going to Storing Locally Stage |

## Online Lab

| Id | System Feature | Input (I) | Conditions (C) | Expected Result (E) |
|---|---|---|---|---|
| 15 | Submit to Online Diagnosis | Image | Internet Available = True -> False | Submit to Local Lab |
| 16 | Update Software |  | Bat. Level = 21% -> 19% | Interrupt Updating and Disable Internet |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  |  | (Energy Saving Mode) |
| 17 | Sync with the database |  | Bat. Level = 21% -> 19% | Interrupt Synchronization and Disable Internet (Energy Saving Mode) |
| 18 | Capture Geolocation | User’s Location | Battery Level = 21% -> 19% | Disable GPS and Internet (Energy Saving Mode) |
| 19 | Submit to Online Diagnosis |  | Bat. Level = 21% -> 19% | Submit to Local Diagnosti and Disable Internet (Energy Saving Mode) |

- **Sending Files Online**

| **Id** | **System Feature** | **Input (I)** | **Conditions (C)** | **Expected Result (E)** |
|---|---|---|---|---|
| 19 | Register Blade | Blade Id, Date | Power Av. = True;<br>Synchronized = True -> False | Synchronizing at background |
| 20 | Capture Image | Image | Power Av. = True;<br>Synchronized = True -> False | Synchronizing at background |
| 21 | Submit to Online Diagnosis | Image | Power Av. = True; | Synchronizing at background |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  | Synchronized = True -> False |  |
| 22 | View Diagnosis | Diagnosis Id | Power Av. = True;<br>Synchronized = True -> False | Synchronizing at background |
| 23 | Register Blade | Blade Id, Date | Power Av. = True;<br>Updated = True -> False | Synchronizing at background |
| 34 | Capture Image | Image | Power Av. = True;<br>Updated = True -> False | Synchronizing at background |
| 24 | Submit to Online Diagnosis | Image | Power Av. = True;<br>Updated = True -> False | Synchronizing at background |
| 25 | View Diagnosis | Diagnosis Id | Power Av. = True;<br>Updated = True -> False | Synchronizing at background |

- **Synchronizing at background**

| **Id** | **System Feature** | **Input (I)** | **Conditions (C)** | **Expected Result (E)** |
|---|---|---|---|---|
| 26 | Sync with the database |  | Power Av. True | Interrupt the |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  | -> False<br>Synchronized = False | synchronization |
| 27 | Sync with the database |  | Power Av. = True;<br>Synchronized = False -> True | Interrupt the synchronization |
| 28 | Sync with the database | User Cancels Synchronization | Synchronized = False<br>Power Av. = True | Interrupt the synchronization |

- **Updating**

| **Id** | **System Feature** | **Input (I)** | **Conditions (C)** | **Expected Result (E)** |
|---|---|---|---|---|
| 29 | Update Software |  | Power Av. True -> False;<br>Updated = False | Interrupt Updating |
| 30 | Update Software |  | Power Av. =true;<br>Updated = False -> True | Come back to Sending Files Online Mode |
| 31 | Update Software | User cancels updating | Updated = False<br>Power Av. = True | Interrupt the software updating |

- **Energy Saving Mode**

| Id | System Feature | Input (I) | Conditions (C) | Expected Result (E) |
|---|---|---|---|---|
| 32 | Enable Local Lab's Features | | Battery Level = 19% -> 20%;<br>Internet Av. = False | Go to Local Lab |
| 33 | Enable Online Lab's Features | | Battery Level = 19% -> 20%;<br>Internet Av. = True | Go to Online Lab |

Step 7: Describe the test case

| | |
|---|---|
| **TC Id** | 01 |
| **Test Objective** | To test the "Capturing Geolocation" Functionality from Local Lab |
| **Precondition** | GPS Available = True<br>Battery Level > 20% |
| **Fixed Conditions** | Internet Available = False,<br>Power Available = False |
| **Input (I)** | User's Location |
| **Test Steps** | 1. Register Blade (Id e Date)<br>2. Open Camera<br>3. See image (c1)<br>4. Capture Image |
| **Varying Conditions (C)** | C1. Bat. Level > 20% → Bat. Level ≤ 20% |
| **Expected Result (E)** | Capture the last location available and disable GPS (Enter in |

| | |
|---|---|
| | Energy Saving Mode) |
| **Post Condition** | GPS Available = False<br>Battery Level ≤ 20% |

| **TC Id** | 15 |
|---|---|
| **Test Objective** | Verify the Internet becoming unavailable while executing the functionality “Submit to Online Diagnosis.” |
| **Precondition** | Internet Available = True |
| **Fixed Conditions** | Power Available = True<br>Memory Available = True |
| **Input (I)** | Image |
| **Test Steps** | 1. Register Blade (Id e Date)<br>2. Capture Image<br>3. Submit to Online Diagnosis (c1)<br>4. Open Diagnosis |
| **Varying Conditions (C)** | c1. Internet Available = True → Internet Available = False |
| **Expected Result (E)** | Receive diagnosis from the local lab |
| **Post Condition** | Internet Available = False |

| | |
|---|---|
| **TC Id** | 29 |
| **Test Objective** | Verify the Power becoming unavailable while updating software |
| **Precondition** | Updated = False<br>Power Available = True |
| **Fixed Conditions** | Internet Available = True |
| **Input (I)** | |
| **Test Steps** | 1. Enter Updating mode<br>2. While Updating (c1) |
| **Varying Conditions (C)** | c1. Power Availability = True -> Power Availability = False |
| **Expected Result (E)** | Interrupt Updating |
| **Post Condition** | Internet Available = False |