CATS DESIGN: A CONTEXT-AWARE TESTING APPROACH

Felyppe Rodrigues da Silva

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

Orientador(es): Guilherme Horta Travassos
Santiago Matalonga Motta

Rio de Janeiro
Março de 2016

CATS DESIGN: A CONTEXT-AWARE TESTING APPROACH

Felyppe Rodrigues da Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) 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.

Examinada por:

_____

Prof. Santiago Matalonga Motta, D.Sc.

_____

Prof. Toacy Cavalcante de Oliveira, D.Sc.

_____

Prof. Lincoln Souza Rocha, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2016

Silva, Felyppe Rodrigues da

CATS DESIGN: A Context-Aware Testing Approach / Felyppe Rodrigues da Silva – Rio de Janeiro: UFRJ/COPPE, 2016.

XII, 117 p.: il.; 29,7 cm.

Orientador(es): Guilherme Horta Travassos,

Santiago Matalonga Motta.

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2016.

Referências Bibliográficas: p. 91-95.

1. Software Engineering. 2. Ubiquitous Systems. 3. Context-Aware Software Systems. 4. Software Testing. 5. *quasi*-Systematic Literature Review. 6. Proof of Concept. I. Travassos, Guilherme Horta *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

"The supreme accomplishment
is to blur the line between
work and play"

(Arnold J. Toynbee)

Agradecimentos

Agradeço primeiramente aos meus familiares por todo apoio, carinho e motivação prestados não somente durante o mestrado, mas toda minha vida acadêmica e profissional. Em especial ao meu pai Antônio José, por acompanhar de perto cada etapa, vibrar com cada pequena conquista e aconselhar a cada obstáculo encontrado. Agradeço a minha namorada Thamires e aos meus amigos pelos momentos de descontração, necessários durante o desenvolvimento de uma pesquisa, e por sempre me aconselharem a ser ousado nos meus passos profissionais e no meio acadêmico.

Agradeço aos amigos do grupo ESE pelas dicas e discussões, sempre extremamente valiosas e que sem dúvida enriqueceram muito o andamento desta pesquisa. Em especial os amigos Fábio e Rebeca por serem mais próximos e poderem também contribuir com apoio motivacional nas horas difíceis. Agradeço ao BTH pela oportunidade de intercambio, essencial no desenvolvimento desta pesquisa, bem como aos professores e pesquisadores do instituto que contribuíram de forma positiva para este trabalho. Em especial, aos amigos Emília e Jefferson que amenizaram as dificuldades encontradas durante o intercâmbio.

Por último, porém não menos importante, agradeço aos meus orientadores Guilherme e Santiago por sua disponibilidade, atenção, ideias, motivação e principalmente paciência ao longo desta pesquisa, agradeço também a UFRJ, a COPPE, ao CNPQ e ao Projeto CAcTUS pela oportunidade e incentivo.

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

CATS DESIGN: UMA ABORDAGEM DE TESTES SENSÍVEL AO CONTEXTO

Felyppe Rodrigues da Silva

Março/2016

Orientador: Guilherme Horta Travassos
Santiago Matalonga Motta
Programa: Engenharia de Sistemas e Computação

Um dos principais aspectos da computação ubíqua é a possibilidade de interação com diversos atores ao mesmo tempo, visando ajudar o usuário a completar suas tarefas de maneira não-intrusiva. Dos fatores que caracterizam um sistema ubíquo, a sensibilidade ao contexto é a habilidade que um sistema pode ter de adquirir informação do contexto no qual ele está imerso, e adaptar seu comportamento de acordo com estes dados. Waze, *smart watches*, casas inteligentes, *Google Now* ou quaisquer sistemas inteligentes que se adaptam com base no perfil do usuário ou das necessidades do ambiente são apenas alguns dos exemplos que caracterizam o conceito de sistemas sensíveis ao contexto.

Entretanto, sendo um novo paradigma de sistemas, traz consigo desafios relativos à qualidade. Uma vez que o contexto no qual o sistema está sendo utilizado pode mudar livremente em tempo de execução, a tarefa de testá-lo torna-se cada vez mais desgastante. Com base nisso, uma revisão sistemática da literatura foi conduzida visando descobrir como este tipo de sistemas tem sido testado.

Estes resultados indicam que os testadores lidam com sistemas sensíveis ao contexto de forma similar aos sistemas tradicionais no aspecto de testes. Isto leva a uma cobertura de testes menos efetiva, uma vez que o contexto é fixado durante o teste. Assim, esta pesquisa propõe uma abordagem capaz de atender as expectativas de teste para sistemas sensíveis ao contexto, o CATS Design. Observando ideias de outros domínios para problemas similares, um processo foi proposto para apoiar a identificação de casos de teste sensíveis ao contexto e avaliado através de uma prova de conceito.

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

CATS DESIGN: A CONTEXT-AWARE TESTING APPROACH

Felyppe Rodrigues da Silva

March/2016

Advisor: Guilherme Horta Travassos
Santiago Matalonga Motta
Department: Computer Science and Systems Engineering

One of the main features of ubiquitous systems is the possibility of interacting simultaneously with several actors in order to support the user to complete its tasks in a non-intrusive way. From all of the features characterizing a ubiquitous system, context-awareness is the ability of a system to gather information from the context where it is immersed and adapt its behaviors according to this information. Waze, *smart watches*, intelligent houses, *Google Now* or other intelligent systems that adapt their behaviors to the user profile or environmental needs are just some examples characterizing the concept of context-aware software systems.

However, the new paradigm of software systems brings together challenges concerning its quality. Since the context in which the system is being executed can freely change at runtime, the task of testing becomes even more effort consuming. Based on this assumption, a systematic literature review was undertook to observe how this type of system has been tested.

The results indicate that software testers handle context-aware software systems similarly to traditional systems. Therefore, this can lead to a less effective test coverage, since the context is always fixed at testing time. So, this research proposes an approach to meet the testing needs of context-aware software systems, the CATS Design. Taking ideas from other domains that present similar issues, a process was proposed to support the identification of context-aware test cases and evaluated through a proof of concept.

INDEX

INDEX OF FIGURES

INDEX OF TABLES

1 Introduction

In this chapter a brief introduction of the dissertation theme, the problem being addressed, its objective and the background that motivated this research are presented. Besides, the research methodology to be applied is also discussed, followed by the presentation of contributions to the field and the dissertation organization.

1.1 Introduction

One of the main aspects of ubiquitous systems is the possibility of interaction with several actors simultaneously in order to support the user in the completion of its tasks in a non-intrusive way. These actors can be other users, other components or even other systems.

In order to achieve this goal, this type of systems make use of sensors to gather the environment's data that will be used to adapt its behavior. This kind of feature is called context-awareness (Dey, 2001). Being able to interact with several different actors at once makes the quality assurance of such systems more difficult when compared with traditional ones (non-context-aware).

The technical literature presents several studies stating the difficulty to assure the quality of ubiquitous systems and why they should be handled distinctly from other software systems (Ducatel et al. (2003), Malik et al. (2007) and Tang et al. (2011)). These authors argue that ubiquitous systems can lose efficacy and efficiency when treated as traditional software systems. Looking for the necessity of treating this type of software systems in a specific way to assure their quality, the CAcTUS[1] Project was started, aiming the discovery of proper ways to test context-aware systems.

This chapter presents more details about the research being conducted, a brief description about the CAcTUS Project and how this research is connected to it. Also, the problem being addressed in this dissertation, the goal to be achieved, together with the background and motivation for this research are presented. Thereafter the research

[1] Project sponsored by CNPq. More information at http://lens.cos.ufrj.br/cactus/

methodology to be followed, the contribution of this research to the community, as well as the dissertation organization are discussed.

1.2 Problem and Objective

Ubiquitous systems are a specific type of systems, particularly new in the software engineering area, which intends to aid the user on the completion of his/her tasks with minimum interference (adapted from Dey & Abowd (1999) by Spínola (2010) and Mota (2013)). In order to achieve this support for the user, these technologies must become “invisible”, as proposed by Weiser (1991). This means that technologies should be integrated with real objects of day-to-day activities in a way that they become indistinguishable.

To be able to optimize the user experience requiring minimum interference and being invisible, the ubiquitous systems can also be context-aware, i.e. use sensors, or any sort of technology, to capture contextual information. The system then uses the collected information about the user and/or environment (physical or computational) to provide services and relevant information to its actors (adapted from Dey, 2001).

To achieve this goal, distinct types of devices need to interoperate with each other and self-organizational features must exist in order to handle contextual variances. This kind of scenario suggests that traditional software quality assurance technologies might not be enough to handle context-aware systems.

Ducatel et al. (2003) stated that ubiquitous systems might lose efficiency and efficacy whether dealt with traditional software technologies, for instance verification and validation techniques designed for traditional systems might not be so effective when applied to ubiquitous systems. In the technical literature is also possible to find other studies claiming these difficulties (Malik et al. (2007), Spínola et al. (2008) and Tang et al. (2011)). With this perspective in mind, Spínola (2010) and Mota (2013) proposed specialized ways to specify and verify requirements for ubiquitous systems.

Being aware that the handling of requirements in a specific way is not enough to guarantee the quality for this type of software systems, this dissertation claims that tests must be designed and implemented with a view towards handling the context-aware feature. Therefore, this research was conducted with the aim of defining a strategy for designing testing procedures for ubiquitous systems, considering the potential contexts that can exert influence to the use of the system.

To achieve this objective, the following research questions were proposed:

- Is it possible to design a test approach considering context variance?
- If yes, does it improve the test coverage when compared with the traditional testing techniques?

1.3 Background and Motivation

This research can be divided in two phases: Prospection and Development. The first one was conducted using the research environment provided by the CAcTUS Project and was responsible to generate the data to be used in the second phase. In both phases, the problem being addressed is the testing for context-aware software systems and how the test criteria for such systems is handled.

1.3.1 Software Testing and the Testing Criteria Issue

The concerns with the quality of software come prior the existence of high-speed processors and modern programming languages. Leeds and Weinberg (1961) argue that no matter how great the performance of a software is, whether the generated result is not the expected one. Based on this, they introduce the idea of software testing, a practice to verify whether the software is working as it is supposed to.

The activity of testing is expected to reveal failures and, due them, find the defects in the software. The incapacity to reveal failures or not find those defects do not assure that the system is free of them (Dijkstra, 1972), by defect Dijkstra meant any behavior distinct from the expected. Believing that the reason for not assuring the defect-free aspect of a system was the way of testing it, Goodenough and Gerhart (1975, 1977) started a research on software testing by raising the question of “what is a test criterion?”, that is, the criterion identifying what constitutes an adequate test.

For Goodenough and Gerhart, if the test criteria were adequate, it would be possible to state that a system had no defects. They define a test criteria as “what properties of a program must be exercised to constitute a ‘thorough’ test, i.e., one whose successful execution implies no errors in a tested program.”

Although a good test criteria improves the chance of finding defects, the idea of assuring a defect-free system is not that simple. In addition, some researches in the technical literature argue that 50% of the software development effort is spent in the testing activity (Yamada & Osaki (1985), Camuffo et al. (1990) and Mathur (2008)).

Knowing the limitations of time and effort imposed by the testing process, Myers and Sandler (2004) define software testing as the process aiming to certify that a software does what it is supposed to do and do not perform any involuntary execution. According to this definition, the software system might present failures, as long as they do not compromise the system expected behavior.

Beside all the raised points regarding the coverage of software testing and it’s impacts over the total software development process effort, the literature also indicates

that practitioners still execute *ad-hoc* testing, i.e. the executed testing process coverage is not predictable (Glass & Hunt, 2006).

In the attempt to aid practitioners not to trust in the *ad-hoc* testing process, international standards have been established to explain and summarize the existing software testing processes and techniques. For instance, the ISO/IEC/IEEE 29119:2013 (‘Software and systems engineering Software testing Part 1:Concepts and definitions’, 2013), is especially concerned with the testing process, test definitions and techniques.

Understanding the concern with the test aspect, the CAcTUS Project looks for ways to test ubiquitous software systems considering the context-awareness of such systems. More details about the project are presented in the following section.

1.3.2 The CAcTUS Project

The CAcTUS Project – Context-Aware Testing for Ubiquitous Systems – is being conducted among three universities:

- Universidade Federal do Rio de Janeiro,
- Universidade Federal do Ceará,
- University of Valenciennes and Hainaut

The project aims to understand test strategies for the quality assessment of actor-computer interaction in ubiquitous systems. To build a body of knowledge, the researchers are undertaking secondary studies to reveal evidence regarding testing and interoperability in context-aware software systems. In order to achieve this goal, the following research questions were proposed by the project members:

- What tests should be performed to ensure the best actor-computer interaction?
- How to consider the different possible contexts in such tests?
- Are there methods for designing these tests that take the context into consideration?

Based on Biolchini et al. (2005), three systematic literature review protocols were organized in the attempt to answer the proposed research questions. The search structure followed the PICO approach (Pai et al., 2004). The CAcTUS Project plan to characterize the state of the art for context-aware software testing can be observed in Figure 1.1.

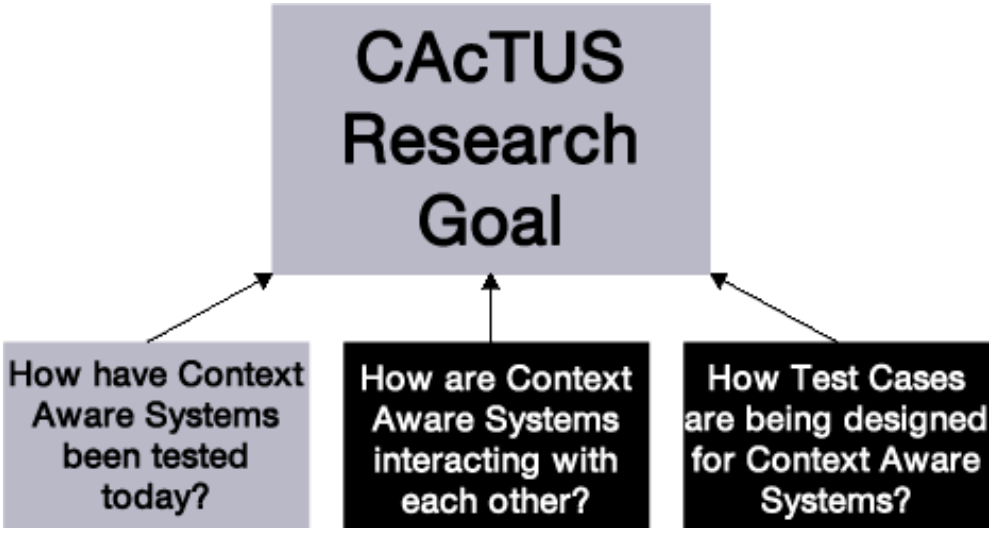

Figure 1.1 - CAcTUS Project Research Strategy

One of the secondary studies part of the CAcTUS project aiming at answering the question of “how have context-aware systems been tested nowadays?” was the trigger for this research. This systematic literature review concerning with test design techniques for context-aware systems found results that motivated a wider research involving other domains, resulting in the proposition of a context-aware testing technique.

1.4 Research Methodology

During the prospection phase, the problem of how to test context-aware systems was identified and more specific research questions were proposed, as presented in Figure 1.1. In order to answer the proposed research questions, a research strategy was raised, planning to conduct a secondary study to establish the state of the art concerning the testing of context-aware systems.

The conducted secondary study was a *quasi*-Systematic Literature Review that aimed at answering the question of “how have context-aware systems been tested nowadays?”. In order to answer this question, a protocol was created and a few trials were executed until the search string was calibrated and the protocol participants’ perspectives were aligned. Therefore, the protocol was executed and the results analyzed.

Thereafter, started the development phase of this research, using the research opportunities that were identified in the protocol results. One of these discoveries suggests the lack of existing approaches for context-aware software testing. Based on this, a research was conducted to find if any other domains present similar problems.

Adapting identified concepts and solutions from other domains (Cybernetics and Organizational Resilience), a test design process focused on the context-aware feature

has been proposed. A few trials using non-real examples were used to evolve and adapt the solutions found into other domains for the context-aware software systems reality. Finally, a proof of concept was conducted to evaluate the proposed technique. The research methodology described above can be observed in Figure 1.2. The green steps were part of the Prospection Phase and the blue one was the Development Phase.

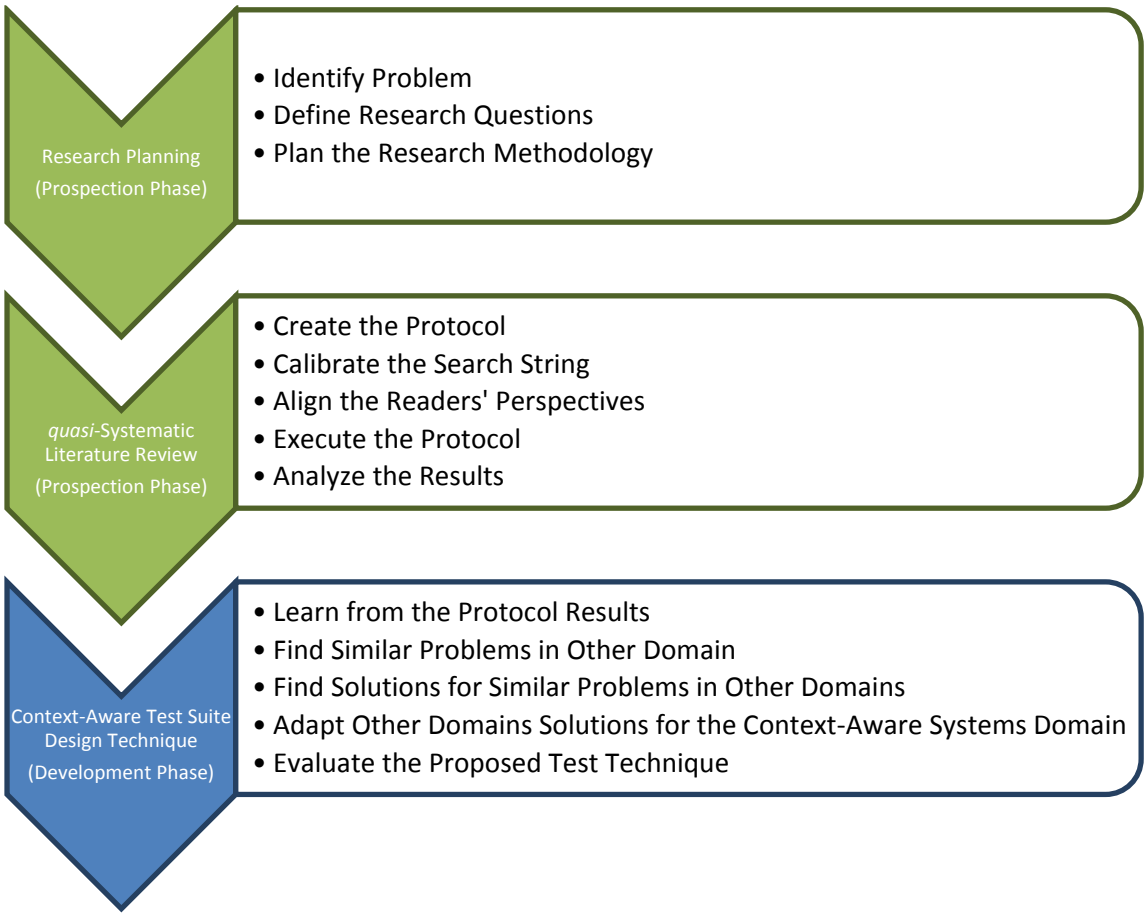

Figure 1.2 - Research Methodology

The environment for the secondary study execution was provided by the CAcTUS project. Table 1.1 shows the research schedule in details. The yellow marks with the number “1” represent the CAcTUS Project, the green ones with the number “2” indicate the Prospection Phase and the blue ones with the number “3” the Development Phase.

Months / Research Steps	feb/14	mar/14	apr/14	may/14	jun/14	jul/14	aug/14	sep/14	oct/14	nov/14	dec/14	jan/15	feb/15	mar/15	apr/15	may/15	jun/15	...
CAcTUS Project	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
qSLR Protocol		2	2	2	2	2	2	2	2	2	2	2						
qSLR Trial 1			2	2														
qSLR Trial 2				2														
qSLR Trial 3				2	2	2	2	2	2									
Readers Perspectives Alignment		2	2	2	2	2	2	2										
qSLR Trial 4									2	2	2	2						
Search for Similar Problem in other Domains												3	3	3				
Search for Solutions from other Domains												3	3	3				
Process Adaptation from other Domains														3	3	3		
Adapted Process Evaluation														3	3	3	3	

Table 1.1 - Dissertation Schedule

1.5 Contributions

Firstly, in the Prospection Phase:

- A qSLR protocol of test design techniques for context-aware software systems.
- Recommendations of practices to be used by practitioners while the proposed process technology is validated.
- A discussion about the applicability of the ISO/IEC/IEEE 29119:2013 test design techniques to context-aware software systems.
- A discussion of the available test design techniques found in the qSLR.

Secondly, in the Development Phase:

- A process for designing context-aware test cases.
- An initial validation of the context-aware test process.

All this contributions are presented and detailed during the dissertation.

1.6 Dissertation Organization

The presented work is organized in six chapters as follows:

- Chapter 1 – Title: Introduction
  This first chapter explains the problem being addressed, its importance and the steps followed in order to solve it.
- Chapter 2 – Title: Concepts and Definitions
  The second chapter presents the definitions to be used along the dissertation, including ubiquitous computing, context-aware systems and software testing.
- Chapter 3 – Title: Testing Context-Aware Software Systems: A quasi-Systematic Literature Review
  The third chapter explains the steps of the *quasi*-systematic literature review conducted to find how context-aware systems are being tested and the lessons learned from this qSLR.
- Chapter 4 – Title: Towards a Context-Aware Test Process
  The fourth chapter presents the adaptation made from other domains until the test process was finally complete.
- Chapter 5 – Title: Context-Aware Test Process Evaluation
  The fifth chapter presents the test process evaluation using proof of concept.

- Chapter 6 – Title: Conclusions and Future Work
  The last chapter summarizes the conclusions, results and contributions of this research.

In addition, this study presents three appendices as follows:

- Appendix A – Title: Smart Camera Requirements
- Appendix B – Title: Smart Camera Non-Context-Aware Test Suite
- Appendix C – Title: CAUS Context-Aware Test Suite

1.7 Chapter Conclusion

This first chapter introduced the context in which this work is presented. An initial perspective of ubiquitous systems and the context-aware aspect were presented together with the difficulties to assure quality for such systems, exemplified by the test criteria issue.

In addition, the main objectives and the problem being addressed were also shown, together with the research methodology to be followed, the main contributions of this research to the community and how the dissertation is organized.

The next chapter presents the definitions used as a basis for this entire research together with the actual state of the art regarding context-aware systems and software testing in this area.

2 Concepts and Definitions

The state of the art and the basic concepts used during the development of this research are described in this chapter. Details about software testing, test coverage criteria, ubiquitous computing and context-aware features are presented here as well. In addition, the importance of these concepts and how they are connected with each other is also discussed.

2.1 Introduction

Software testing is a dynamic technique for the verification and validation of software, which consists of executing the software under planned conditions with the objective of revealing defects. Once the failures are observed, it is possible to look for the faults (defects) and then correct them to improve software quality and reliability (Rocha et al., 2001).

Although this sort of practice can support the software quality, it is not feasible to test all possible usage situations of the system (Delamaro et al., 2007). Based on this, it is important to wisely choose the test coverage criteria to be adopted during the software testing process, i.e. the aspects that will be covered during testing. The more features a system present, the wider are the possibilities regarding the test coverage criteria to be selected.

On the other hand, Weiser (1991) proposed the concept of ubiquitous systems, which are software systems that are immerse into the environment and intend to aid the user on the achievement of its tasks. A special case of ubiquitous systems are those that present properties of context-awareness. Such software systems capture context information (user, place, environment, object or else) through sensors or logs in order to improve the user experience with the system. GPS, mobile applications, smart televisions, self-regulating air-conditioners and others are just some examples of this type of systems.

Since context-aware software systems need to deal with the variation of context in real time, one more feature need to be considered when selecting the test coverage criteria during the test plan. Therefore, this chapter intends to provide a deeper discussion of software testing (considering the international standard ISO/IEC/IEEE 29119:2013), ubiquitous software systems, context-aware software systems, the relations among these topics and the challenges that rise with these relations.

2.2 Software Testing and the ISO/IEC/IEEE 29119:2013

In chapter 1, the basis for software testing was presented together with the issue of achieving a proper testing criteria. In order to support this goal, the international standard ISO/IEC/IEEE 29119:2013 is presented in this section and used as a former test process during this dissertation.

2.2.1 Glossary of Terms

This section presents a summary of terms that will be used during this dissertation regarding software testing. The definitions for **test artefacts** were recovered from the ISO/IEC/IEEE 29119:2013 ('Software and systems engineering Software testing Part 2:Test processes', 2013) and the definitions for **fault**, **failure**, **error** and **defect** were adapted from the ISO/IEC 24765:2009 ('Systems and Software Engineering Vocabulary', 2009).

- **Test Plan:** Detailed description of test objectives to be achieved and the means and schedule for achieving them, organized to coordinate testing activities for some test item or set of test items.
- **Test Coverage Criteria:** Conditions under which the testing activities are considered complete.
- **Test Design:** Test process for deriving and specifying test cases and test script/test suite.
- **Test Case:** Set of test case preconditions, inputs (including actions, where applicable) and expected results.
- **Test Input:** Input to which the Test Item will be stimulated.
- **Test Item/Test Unit:** Product/functionality being tested.
- **Test Output:** Response returned by the Test Item after receiving an input.
- **Test Oracle/Expected Result:** Observable predicted behavior under conditions based on its specification or another source.
- **Test Result:** The comparison between the Test Output and the Test Oracle.
- **Test Environment:** Refer to facilities, hardware, software, firmware, procedures, documentation intended for or used to perform the testing of software.
- **Test Script:** Test procedure specification for manual or automated testing.
- **Test Suite:** Group of Test Cases for a particular Test Criteria.
- **Test Incident:** It is when the test output and the test oracle do not match.

- **Error:** A human action that produces an incorrect result.
- **Fault:** A manifestation of an error in software.
- **Failure:** An event in which a system or system component does not perform a required function within specified limits, it is consequence of a fault.
- **Defect:** An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.

2.2.2 ISO/IEC/IEEE 29119:2013 Test Process

The ISO/IEC/IEEE 29119 series of standards was designed to define an internationally agreed set of standards regarding software testing that can be used by any organization when performing any form of software testing (‘Software and systems engineering Software testing Part 1:Concepts and definitions’, 2013). The ISO/IEC/IEEE 29119 also reinforces the purpose of aiding organizations on their software testing processes, independent of the organizational context, domain, scale or software development lifecycle adopted.

The types of tests presented by the standard are not limited to dynamic testing execution for projects or products, but encompasses all testing lifecycle, from organizational planning to execution. Therefore it can be classified into three layers: Organizational (concerned with test policy to be followed by the entire organization), Management (concerned with the planning of tests for products) and Dynamic (concerned with the execution of the planned test on the product). These layers can be observed in Figure 2.1 retrieved from the ISO/IEC/IEEE 29119 part 2 (‘Software and systems engineering Software testing Part 2:Test processes’, 2013).

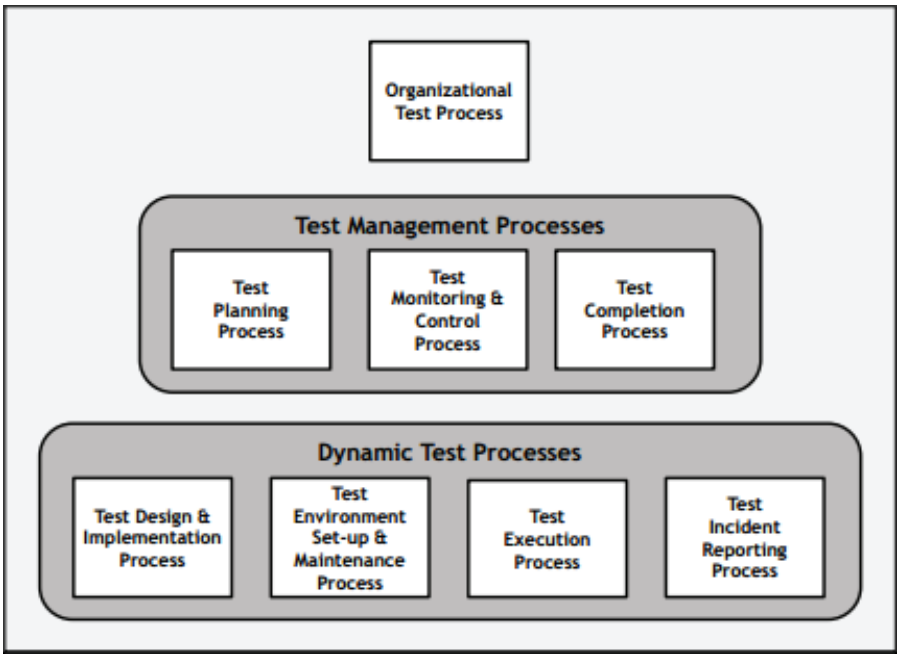

Figure 2.1 – Test Layers (ISO/IEC/IEEE 29119)

During the management layer, the test plan is generated. The test plan is responsible for providing essential information to be used during the dynamic phase. For instance, the design strategy, the context in which the project will be tested, the risks involved, the test design techniques to be applied, the test item and, as presented in the previous section, the test criteria.

2.2.3 ISO/IEC/IEEE 29119:2013 Dynamic Test Process

The test criteria may vary according to the context in which the software is expected to be executed, the software size, the time left in the project or other several reasons. Based on the selected criteria during the test plan, the test design to be conducted in the dynamic phase and the test environment will change. The test design is the activity responsible for designing test cases to be executed. The test environment is the set of facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform the testing of software. How these activities relate with each other is presented in Figure 2.2 retrieved from the ISO/IEC/IEEE 29119 part 2 (‘Software and systems engineering Software testing Part 2:Test processes’, 2013).

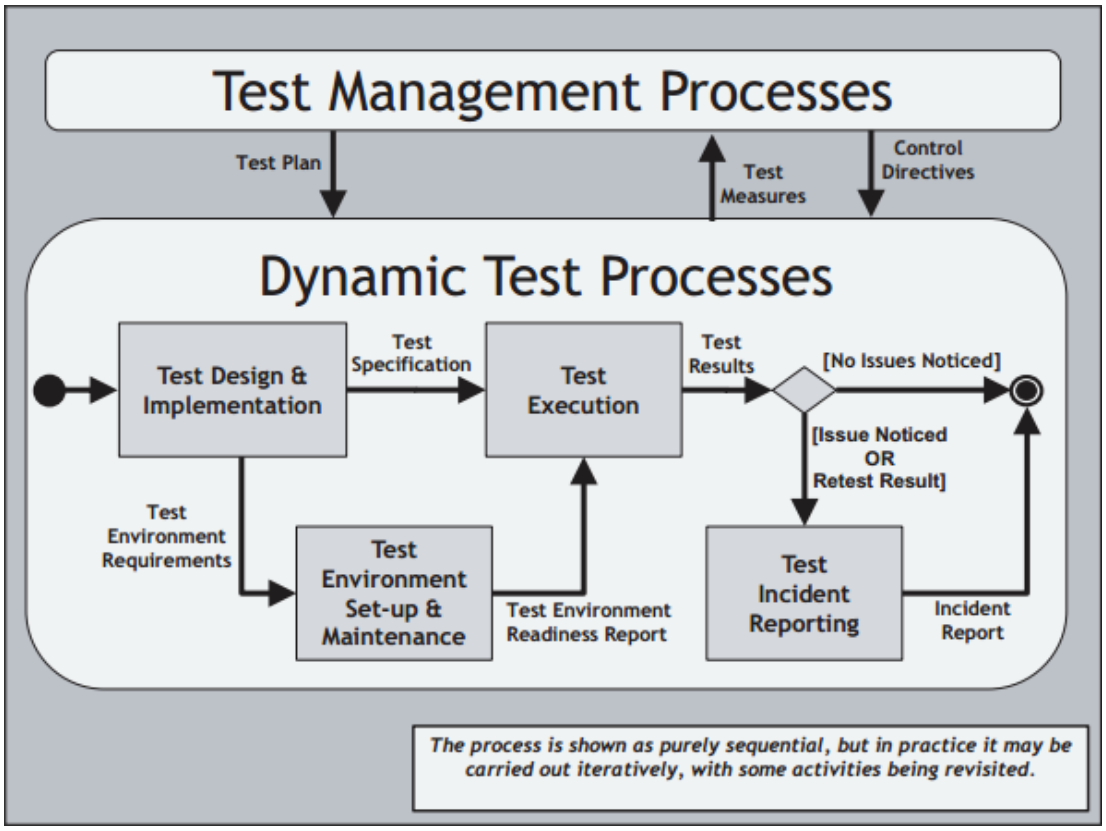

Figure 2.2 - Dynamic test process (ISO/IEC/IEEE 29119)

When applying the dynamic test process, the instructions defined in the test plan must be followed. Basically, it consists on preparing the test environment as specified, designing the test according to the chosen test criteria, executing the test cases with the defined test design technique accorded and reporting the test incidents.

In order to apply the test criteria specified in the test plan, the test design must set a list of preconditions, specify the test cases that achieve the proposed test coverage criteria and design the test procedures/test scripts/test suites, i.e. collections of test cases to be executed for a particular objective. This process can be observed in Figure 2.3 retrieved from ISO/IEC/IEEE 29119 part 2 (‘Software and systems engineering Software testing Part 2:Test processes’, 2013).

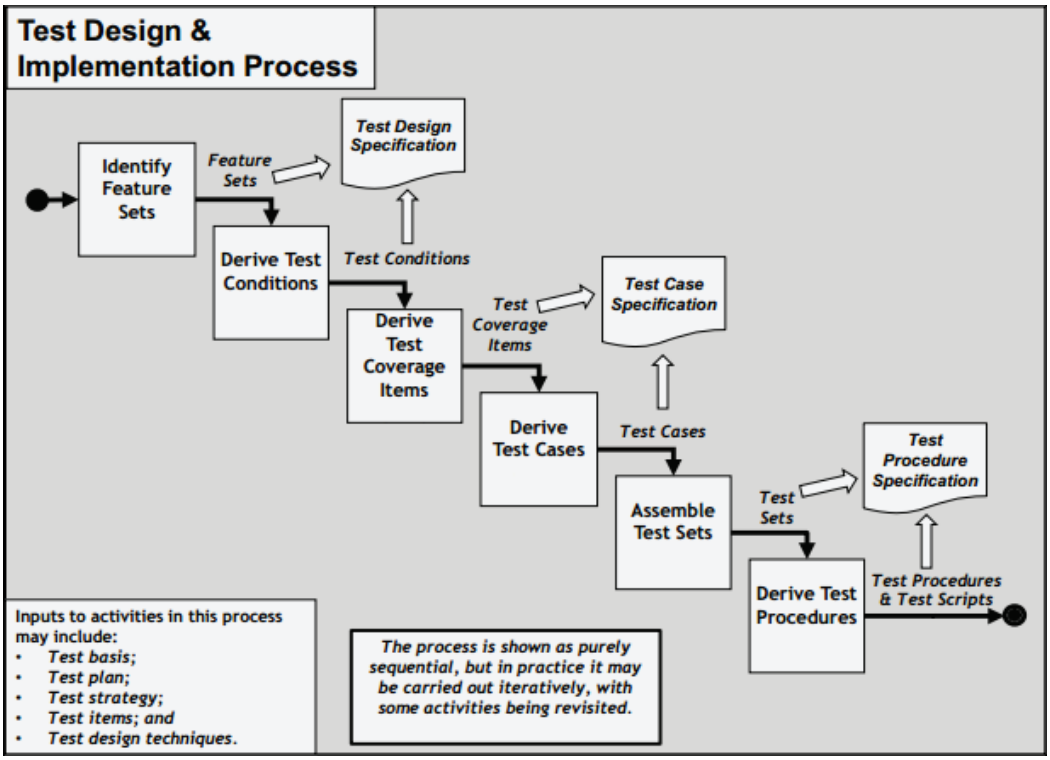

Figure 2.3 - Test design and implementation process (ISO/IEC/IEEE 29119)

With the test suite ready and test environment prepared, it is possible to execute the application and start looking for defects. Following the instructions on the test cases, the test item, which can be the entire system or just a simple functionality, receives a set of inputs. These inputs generate a set of outputs, which are then compared with the expected outputs/test oracle. If they do not match, this characterizes a test incident and must be reported. Summarizing, if the system presents any output different from the expected, this denote a failure. The activity of test execution can be observed in Figure 2.4.

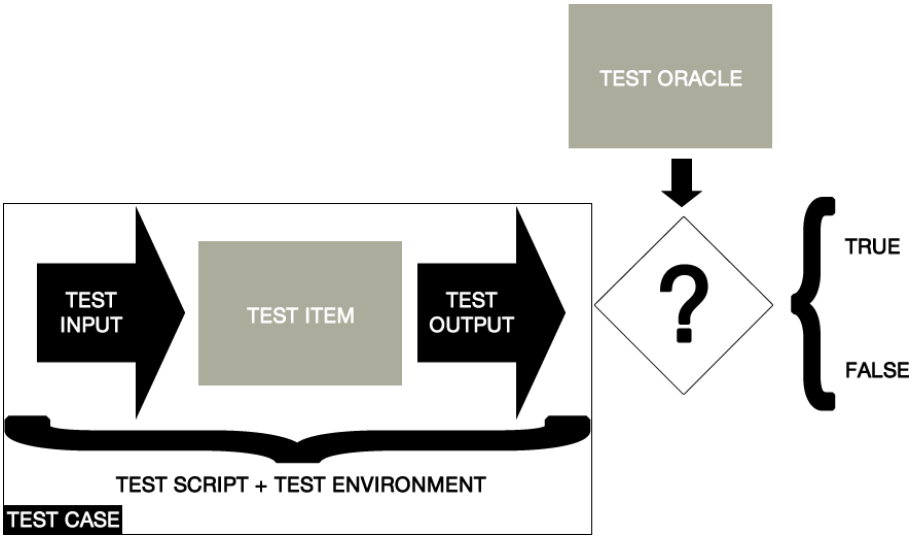

Figure 2.4 - Test execution process

Even if the test process is well established, the test criteria still remains an issue. Not always the chosen test criteria is something measurable, implying at a possible lack of coverage while designing the test cases. If the test cases do not achieve the proposed test criteria for the test plan, the entire test may be compromised. The more features the system presents or the wider is the system objective, the more complicated it will be to assure good test criteria. An example of this type of system is presented in details in the next section.

2.3 Ubiquitous Computing

In 1991, Mark Weiser envisioned that “the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.” (Weiser, 1991). With this idea, the ubiquitous computing started to give its first steps. Weiser believed that computers should not be limited to desktops or machines with direct user interaction. He defended the idea that a true powerful technology should be able to support the user in its everyday activities, even without its perception. Weiser defined this concept as invisibility and this type of system as ubiquitous.

Almost two decades later, ubiquitous systems became reality. Smart houses, GPS selecting a better path with less amount of traffic, mobile applications, wearable devices constantly being updated by information gathered from the internet, digital cameras capable of focusing the image by themselves and so on (Wei, 2014). In order to better understand its characteristics, Spínola et al. (2007) conducted a systematic review (Biolchini et al., 2005) to find the features that characterize ubiquitous software systems.

In their work, Spínola et al. identified ten characteristics that define a software system as ubiquitous. Not all ubiquitous software systems need to present all of these ten characteristics, but to be considered ubiquitous, it must present at least some of them:

- **Service omnipresence:** it allows the users to move around with the sensation of carrying the computing services with them;
- **Invisibility:** the ability of being present in objects of daily use, weakening, from the user’s point of view, the sensation of explicit use of a computer and enhancing the perception that objects or devices provide services or some kind of “intelligence”. With that, it is possible to find proper alternatives for traditional graphical interfaces used on desktop solutions in favor of more natural ways of data input in such a way that the interface itself will be minimally perceived by the user;

- **Context sensitivity:** ability to collect information from the environment where it is being used to improve the user's experience;
- **Adaptable behavior:** ability of, dynamically, to adapt itself and offer services according to the environment where it is being used, respecting its limitations;
- **Experience capture:** ability of capturing and registering experiences for later use;
- **Service discovery:** pro-active discovery of services according to the environment where it is being used. The application has to interact with the environment and allow the user to do the same, in order to find new services or information to achieve some desired target;
- **Function composition:** the ability of (based on basic services) creating the services required by the user;
- **Spontaneous interoperability:** the ability to change partners during its operation and according to its movement;
- **Heterogeneity of devices:** it provides mobility among heterogeneous devices. That is, the application could migrate among devices and adjust itself to each of them;
- **Fault tolerance:** the ability to adapt itself when facing environment's faults (for example, on-line/off-line availability).

Spínola et al. claim that the scope of ubiquitous systems is deeply related to the presented characteristics, however the absence of some aspects do not imply that a system is not ubiquitous. The ubiquity in a system can be met completely or partially according to the presence or absence of these aspects.

Some of those features like **Context Sensitivity**, **Adaptable Behavior** and **Experience Capture** are achieved by the ubiquitous software systems using sensors or usage logs, which enable the software system to collect data without user perception and adapt its functionalities to better adjust to different usage situations. A specific case of ubiquitous software system are context-aware software systems. These systems use environmental information to better serve the user in achieving his/her tasks.

2.4 Context-Aware Software Systems

Context-Aware Software Systems are a particular case of Ubiquitous Systems on which the characteristics of Context Sensitivity and Adaptable Behavior are mandatory. All other ubiquitous characteristics might be present in the system as well, but they are optional.

2.4.1 Glossary of Terms

The definitions of **Context** and **Context-Aware** were adapted from the definitions provided by Dey & Abowd (1999).

- **Context:** Any piece of information that may be used to characterize the situation of an entity (logical and physical objects present in the system's environment) and its relevant relations for the actor-computer interaction.
- **Context-Awareness:** Dynamic property representing a piece of information that can evolutionarily affect the overall behavior of the software system in the interaction between the actor and computer.
- **CASS:** Context-aware software systems.

2.4.2 CASS Particularities

Dey & Abowd (1999) define context as "any information that can be used to characterize the situation of an entity". An entity is a person, place, or object that is considered relevant to the interaction between an actor and an application, including the user and applications themselves, and a system is context-aware whether it uses the context to provide relevant information and/or services to the user, where relevancy depends on the user's task and perspective.

Dey and Abowd (1999) suggest organizing the context information into five dimensions, so answering the following questions the system can be aware of the context:

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

For instance, a mobile application can be seen as an example of a context-aware system. A context in which the user (who) is at the office (where) during a meeting (what) at the working time (when) because it is a user's routine (why) can be recognized by the system. The system then can gather the context data by the user profile, user calendar, clock time and other available resources. Based on the presented context, the system

can turn on the airplane mode in order to avoid unnecessary disturbance or detect that there is an incoming call and automatically avoid it.

However, this definition from Dey and Abowd (1999) did not seem to get the entire meaning of context and context-awareness. Suppose an application trying to execute a high memory consuming task. If it is context-aware, such application could end several secondary tasks in order to have more available memory.

This type of context-aware feature supports the interaction between the software and the hardware, but not necessarily the user. This concept was already discussed and is called actor-computer interaction, i.e. the actor can be a user, a functionality or even other system that interacts with the system.

It is possible to observe that this type of system presents some particularities, for instance, the ability of retrieving data using sensors, the skill of adapting its behavior according to the context without the user request and the interoperability with other systems and contexts. The next section addresses the challenges risen by the CASS particularities.

2.5 Challenges for Context-Aware Software Systems

The main issue regarding CASS is that contextual data is continuously changing (Dey & Abowd, 1999). This implies in several issues to be handled when considering such systems. Some authors listed possible challenges for the CASS domain, among them Winograd (2001) found the following issues:

- **Defining context and building context-aware models:** For the author, the definitions and models for CASS are still immature and task specific without the existence of Standards and support tools
- **Sensing and predicting context data:** Lack of predictability implies in possible contexts not being captured by the system. Bayesian models can be used to predict context.
- **Representing and storing context information:** The context representation should make the context interpretation easier and, whether possible, follow a standard.
- **Inferring context and adapting system behavior:** The interpretation is one of the main aspects of CASS and is still a challenge.
- **Evaluation of CASS:** An evaluation criteria must be defined for verification and validation of CASS, as well as measures for quality assurance. This also includes the software testing process, which needs to handle the context variation during the test.

- **Privacy Control:** The privacy of the contextual data of the users must be protected from malicious entities.

Although Winograd's study is from 2001, the challenges proposed by him are still valid. In a more recent work which is also a contribution of this dissertation, Matalonga et al. (2015a) proposed challenges for CASS that seem very similar to the ones proposed by Winograd. For instance, one of the challenges raised by Matalonga et al. is the **context variation**, which implies that several users, devices, services, usage scenarios and even hardware need to be considered at once and all together.

In addition, according to Matalonga et al. research, the **context variation can cause impact on the system behavior** implying that these contexts must be anticipated in order to make the system aware of all possible impacts it can suffer.

Concerning the **hardware resources** for such systems, to handle the context variation, resources like sensors, memory, GUI, battery, network and more must be present in the system. For this, the possible usage contexts must be known, otherwise the system might not have the tools to operate.

Observing the proposed challenges in this domain, the following chapter presents a secondary study conducted in the technical literature trying to understand how CASS are being tested. The point of this study is to observe how the technical literature handles some of the challenges presented before, mainly the **evaluation of CASS** and **context variation**.

2.6 Chapter Conclusions

This chapter introduced the ISO/IEC/IEEE 29119:2013 Testing Process, the concepts of Ubiquitous Computing and more specifically the Context-Aware Software Systems, which are a particular case of the ubiquitous systems. In addition, the existing challenges regarding this type of systems were discussed. The presented concepts are the basis for all further discussion in this dissertation.

Among the presented challenges, the difficulty of modeling and evaluating CASS due to the context variance was stated. The test process presented by the ISO/IEC/IEEE 29119:2013 do not seem to handle the context variance aspect, turning this an issue for the verification and validation of such systems.

In order to verify how the technical literature handle this issue, the next chapter presents a *quasi*-systematic literature review (qSLR) looking for what is known about test design techniques aiming at the context-aware aspect. The objective of this literature review is to find ways to evaluate/test CASS in order to find solutions for these proposed challenges.

3 Testing Context-Aware Software Systems: A *quasi*-Systematic Literature Review

This chapter presents the steps concerned with a quasi-Systematic Literature Review (qSLR) execution, the found results and their contribution to this dissertation. This qSLR results aim to characterize the state of the art regarding test design techniques for context-aware systems.

3.1 Introduction

The concepts and challenges presented in chapter 2 imply how difficult is to manage the verification, validation and testing of CASS. The cause of most of these challenges lies on the non-existence of standards and well-established models to describe the contexts and their variance. These factors suggest the lack of maturity of the CASS domain.

In order to observe how the technical literature deals with CASS, the proposed challenge of CASS evaluation considering the context variance was formalized and executed as a secondary study. The main goal is to find how CASS are being tested in the technical literature and observe if these approaches handle the context variance issue.

To achieve this goal, first we need to discover which techniques exist in the technical literature to test CASS. After that, the coverage of each identified technique can be compared to see if they provide any improvement when compared to the traditional testing techniques. Otherwise there will be no evidence of the quality obtained by the found techniques.

Based on this, this chapter continues the Prospection Phase of this research presenting the investigation conducted with the aid of CAcTUS Project and with the direct participation of this researcher, aiming to find the existing technologies for testing CASS. Further details of the *quasi*-systematic literature review process are also presented together with a synthesis of its results.

The research questions, the search string, the inclusion/exclusion criteria, the retrieved evidence and its contribution for this research are shown as well. Finally, a discussion detailing whether the context variance is handled and the test coverage obtained by each of the found approaches is presented, followed by the threats to validity found during the process.

For a complete overview of the *quasi*-systematic literature review about test design techniques for CASS conducted, the research protocol is available as a technical report. (Rodrigues et al., 2014).

3.2 Objectives

As presented in chapter 2, according to the challenges proposed by Winograd (2001) and Matalonga et. al. (2015a), the main difficulty of evaluating CASS is due to two factors: the immaturity of the CASS domain (resulting in the inexistence of standards and models) and the context variance (which implies that the evaluation/test process need to handle these possible contexts).

In order to observe how the technical literature deal with these challenges, a *quasi*-Systematic Literature Review was conducted following the guidelines proposed by Biolchini et al. (2005). More specifically, the *quasi*-Systematic Literature Review aimed to identify the different available testing techniques for CASS and the coverage levels that each of these techniques could reach.

Following the GQM approach (Basili et al., 1994), the research questions were derived aiming at the identification of how context-aware software systems are being tested and how the existing approaches can assure quality.

In order to achieve this goal, the following research questions were described:

- Which are the existing methods for testing context-aware software systems?
- What is the coverage obtained by each of them?

Together with the research questions and the goal to be achieved when the research questions gets answered, some metrics need to be collected in order to support the answering of the research questions. The complete GQM structure can be observed in Figure 3.1.

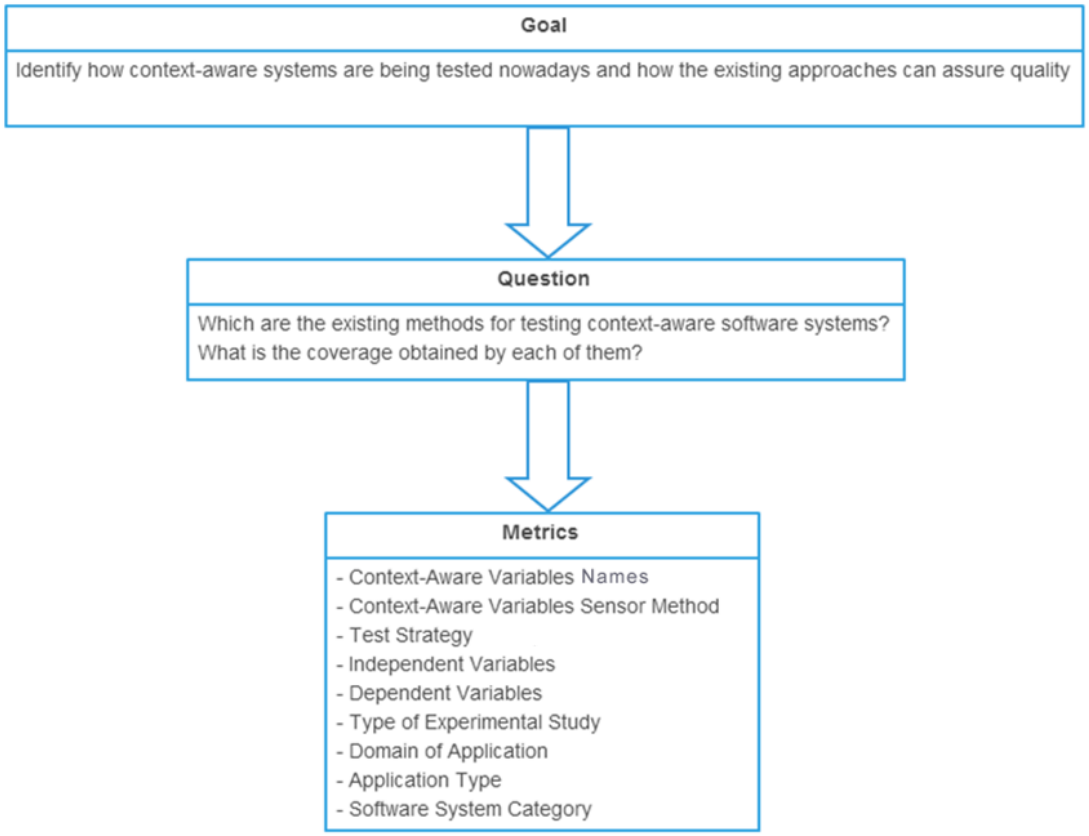

Figure 3.1 - The Goal, Question, Metric view of the research

3.3 Search String, Source and Studies Selection

The research protocol used for the *quasi*-systematic literature review process was based on Biolchini et al. (2005) and the search string was built according to the PICO approach (Pai et al., 2004).

This methodology splits the search string into four parts: Population of interest, Intervention, Comparison intervention (if applicable or available) and Outcome. Since this research is mainly a characterization, the comparison factor cannot be applied. Therefore, we can classify it as *quasi*-Systematic Literature Review (Travassos et al., 2008).

Until the final version of the protocol, the search string has evolved. Therefore, the majority of adjustments resulted from the protocol trials. Inclusion and exclusion of keywords based on the articles found during the trials are just example of this. The reader can observe the reasons behind these changes in the detailed protocol (Rodrigues et al., 2014). Here the final version of the search string is presented:

− Population: Sensibility to the context.

- Keywords: "context aware" OR "event driven" OR "context driven" OR "context sensitivity" OR "context sensitive" OR "pervasive" OR "ubiquitous" OR "usability" OR “event based” OR “self adaptive” OR “self adapt”.

− Intervention Control: Software testing.

- Keywords: "software test design" OR "software test suite" OR "software test" OR "software testing" OR "system test design" OR "system test suite" OR "system test" OR "system testing" OR "middleware test" OR "middleware testing" OR "property based software test" OR "property based software testing" OR "fault detection" OR "failure detection" OR "GUI test" OR "Graphical User Interfaces test".

− Comparison: None.

− Outcome Measure: Methodology.

- Keywords: "model" OR "metric" OR "guideline" OR "checklist" OR "template" OR "approach" OR "strategy" OR "method" OR "methodology" OR "tool" OR "technique" OR "heuristics".

From the final package of articles, only studies written in English and available on the web were selected. Three search engines were selected for the protocol execution: **Scopus**, **Web of Science** and **IEEExplore**. These three search engines were selected due to their consistency, which allows future replications if necessary and the number of indexed bases, amplifying the coverage. In addition, the technical literature presents comparisons of the performance of different search engines and the three selected ones are well-recommended and provide complementary results among them (Buchinger et al., 2014).

For the studies selection, inclusion/exclusion criteria were agreed among the researchers. The exclusion criteria is the counterpart of the inclusion criteria with the addition of the exclusion of articles older than 2000, since ubiquitous computing was not being developed at that time, just speculated. The inclusion criteria are as following:

- To talk about test strategies; or
- To talk about test design; or
- To talk about test methods; or
- To talk about test metrics; or
- To talk about testing measurement; or
- To talk about fault detection; or
- To talk about error detection; AND

- To present characteristics of context in context-aware software systems; or
- To present some characterization of context in context-aware software systems; or
- To analyze specific problems in sensing variables of context in either:
    - Human Computing Interaction
    - Software Systems' usability

After gathering all the studies retrieved by the search string, the inclusion/exclusion criteria were applied to the articles evaluated by three researchers. The decision of including or excluding the retrieved studies was made according to the scheme presented in Table 3.1 by analyzing the title and abstract.

Vote $_1$	**Vote** $_2$	**Vote** $_3$	***Decision***
Accept	Accept	Accept	***Accept***
Accept	Accept	Doubt	***Accept***
Accept	Reject	Reject	***Reject***
Doubt	Reject	Reject	***Reject***
Reject	Reject	Reject	***Reject***
Any other combination			***Group Discussion and acceptance only by consensus***

Table 3.1 - Acceptance criteria

The final package was retrieved from the databases in October 30th of 2014. From this execution, 1820 studies were recovered, which 110 made it through the inclusion/exclusion criteria. These then were discussed among the readers. After that, 75 were kept for full reading and only 11 were considered relevant for the research goal.

The results of the protocol were provided by the eleven selected papers:

- Satoh (2003),
- Tse et al. (2004),
- Ryan & Gonsalves (2005),
- Merdes et al. (2006),
- Jiang et al. (2007),
- Wang et al. (2007),
- Alsos & Dahl (2008),
- Wang et al. (2010),
- Amalfitano et al. (2013),

- Canfora et al. (2013) and
- Wang et al. (2014);

3.4 Studies Summary

Satoh (2003) presents a framework for the emulation of context variables in a ubiquitous software solution. The proposed architecture supports the development and testing of ubiquitous systems. Two examples of use are presented: UPnP protocol and Network mobility of a printer within a building. No formal evaluation of the framework was conducted though.

Tse et al. (2004) apply metamorphic testing to a set of valid inputs and outputs test cases to obtain new test cases. Tse et al. states that obtaining total coverage for context-aware systems while testing is not feasible, so they suggest an approach using metamorphic transformations. Using known relations between inputs and outputs, test oracles are created and used to compare the test results of the test cases generated using metamorphosis.

Ryan & Gonsalves (2005) conducts an experiment to evaluate the usability testing of context-aware applications. Two application (pc and mobile) scenarios were deployed into four different configurations (mobile native and html native) and executed without a formal technique (*ad-hoc*) by 12 users. These users had minimal experience using a Smartphone and/or mobile applications and conducted the experiment following a task list. The results show that users preferred the PC version over the mobile and that native mobile application had better results when considering the bandwidth usage perspective.

Merdes et al. (2006) proposes a XML-based-tool to handle the problem of resources availability in context-aware software systems. The XML layer gives the user the ability to configure run-time scenarios in which the test suite for the application will be executed. A case study is conducted to compare distinct test strategies according to the ranking provided by the proposed tool. The tool considers three dimensions during the evaluation: cost, test coverage and functionality protection.

Jiang et al. (2007) proposes a framework to manage and test applications on mobile devices. A tool based on the framework is presented and covers the testing process from the planning until the execution. When considering sensors readings, the tool can be used to perform middleware testing, i.e. simulate values for the sensors. Finally, to evaluate the tool, a controlled experiment was conducted to cover test planning and execution. The results compared the maintenance cost of the test cases generated by different testing tools.

Wang et al. (2007) proposes a framework to extend the coverage of a non-context-aware test suite. By identifying context variables that influence on the system, the framework generates test cases considering these variables. To validate the proposal, a case study was executed. A test suite was extended for a context-aware application considering location and user interest as context variables.

Alsos & Dahl (2008) proposes a case study to evaluate the usability of a specific context-aware system in the healthcare domain. Among the objectives of this study are: achieving extensive mobility, frequent context shifts, and the need for quick and effortless access to relevant information for immediate care situation, making hospitals suitable for the application of mobile and pervasive computing technology. To achieve the proposed goal, three distinct scenarios were prepared to be executed. The first concerns the location-awareness of the system, while the second and third ones do not make use of contextual information.

Wang et al. (2010), based on two of their previous works (Wang et al., 2007 and 2009) proposes the metric Context Diversity as a coverage predictor for context-aware test suites. The metric is calculated considering the number and values of the context-aware variables present in the system. In addition, a proof of concept is presented showing the context diversity value of different test suites being executed in a test item. The conclusion is that context diversity can be a good predictor of context-aware test suites coverage.

Amalfitano et al. (2013) suggests three kinds of techniques to test context-aware apps: Manually, Mutation-based and Exploration-based. This third one has been then investigated by the authors. They established event-patterns to be used in automatic black-box testing processes based on dynamic analysis of the mobile app. In this case, an app exploration technique may be used to define and execute test cases at the same time. Using the exploration technique, the author conducts a case study running two kinds of tests in some free apps. The first type tests only the GUI part of the app, the second one also tests how the app behaves according to alterations in the sensors that the app is related to. The conclusions indicate that test coverage is increased when considering context events.

Canfora et al. (2013) conducts a case study in order to evaluate user experience. Two distinct scenarios are proposed to the users. One of the devices to be used however only had 70% of its RAM memory available. The test cases are created using a native language described by the author and executed by an Arduino simulator. The device is placed on a rotating platform which moves in a controlled manner, simulating the movements affecting the display orientation and sensor settings. As a result, they concluded that their predictions about the users' impressions were correct.

Wang et al. (2014) extended their previous work (Wang et al., 2010) with a tool capable of automatically generating context-aware test cases using mutation algorithms. This case study enables them to confirm that context diversity is a good predictor of context-aware test suites coverage.

3.5 Answering the Research Questions

3.5.1 Which are the existing methods for testing CASS?

Even though some studies were recovered from the research protocol described in this chapter, very few evidence have been retrieved from its execution. Regarding the first research question, the techniques presented in the final package of studies can be divided into four distinct groups:

- Proposition and evaluation of frameworks without presenting information about context-awareness
  - Satoh et al. (2003)
  - Ryan & Gonsalves (2005)
  - Merdes et al. (2006)
  - Jian et al. (2007)
  - Alsos & Dahl (2008)
- Middleware testing
  - Canfora et al. (2013)
- Applied testing with lack of coverage prediction
  - Tse et al. (2004)
  - Wang et al. (2007)
  - Amalfitano et al. (2013)
- Context-Aware metric proposal
  - Wang et al. (2010)
  - Wang et al. (2014)

The selected studies from the first group propose a test approach/framework in which a context-aware software system is used in the evaluation process. Nevertheless, not necessarily the approach is focused on the context-aware feature.

The second group brings the idea of simulating the sensors values of context-aware software systems instead of actually get the information from the sensors, which is called Middleware testing in the technical literature. In such works, the authors write test cases considering the context variables as regular inputs by simulating and keeping their values fixed during the testing. Although this group is capable of handling a great

amount of distinct contexts and even the context variance issue, the found studies do not handle it.

The third group are the approaches classified as random testing, metamorphic testing and exploratory testing. These types of techniques can generate a large amount of valid test cases, however, since the results are generated without control, it is not possible to predict their coverage, making the test process less reliable.

The last group do not present any testing approach concerned with context-aware software systems. Nevertheless, they present metrics to measure the coverage obtained by the test cases considering the context variance. The metrics were also evaluated, giving more confidence of their applicability.

3.5.2 What is the coverage obtained by each of them?

Regarding the second research question, the only approach found in the technical literature was the metric of Context Diversity proposed by Wang et al. (2010). However, apart from the testing coverage proposal regarding context-awareness, no evidence of software testing technique focused on the context-aware feature was found. It means that context-aware systems are being tested as traditional ones.

3.6 Discussion

As mentioned in Chapter 1, as more specific features a system presents, as more effort will be needed to test it. Since CASS retrieves data from the environment and adapts its behavior, it is feasible to believe that this behavior must be taken into account during the test process and can imply on the increasing of effort. In order to verify this and whether the technical literature presents any approach able to handle the context variance in software testing, this qSLR was undertook.

According to the standards presented in the Chapter 2, the only classification in which it was possible to observe a pattern among the results was the ‘Test Design Technique’ according to the ISO/IEC/IEEE 29119:2013. Considering this classification, six studies ((Ryan & Gonsalves, 2005), (Merdes et al., 2006), (Jiang et al., 2007), (Wang et al., 2007), (Alsos & Dahl, 2008) and (Canfora et al., 2013)) were classified as Scenario Testing.

Scenario Testing is an approach that conducts the tester to obey a pre-stated scenario to test a specific feature. However, as mentioned in Chapter 2, one of the main challenges for the CASS testing is that the context variance must be considered and this testing approach forces the system to limit its functionalities to a specific scenario. This kind of approach treats context variables as single inputs with fixed values, not as a variable context influencing the entire test item, as it can be observed in Figure 3.2.

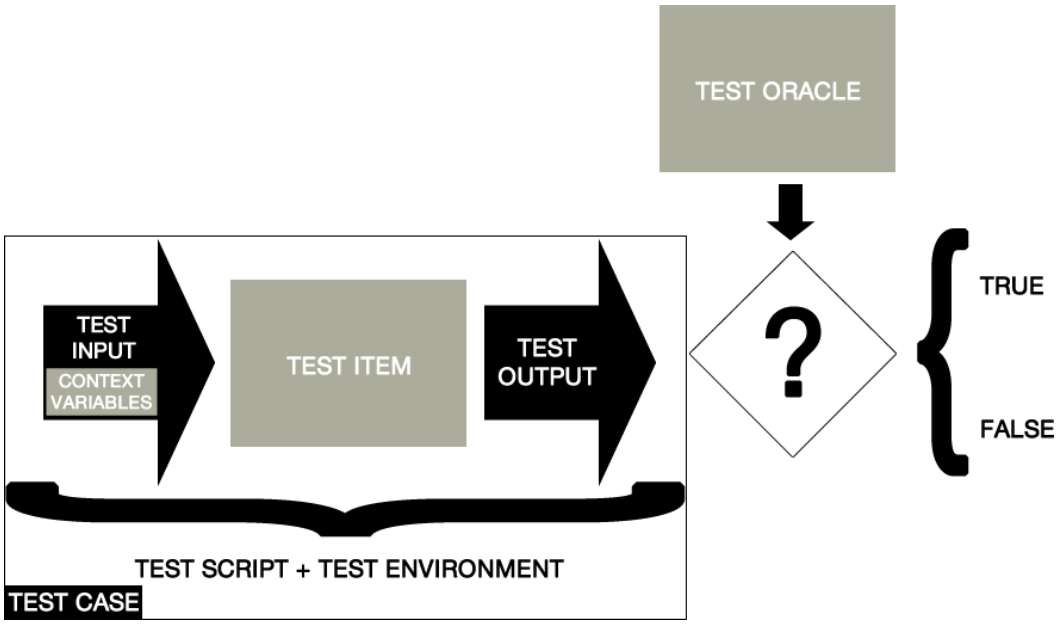

Figure 3.2 - Scenario Testing approach for context-aware systems

Two of the identified approaches ((Tse et al., 2004) and (Amalfitano et al., 2013)) propose test design techniques considering random factors such as metamorphic testing, random testing and exploratory testing. The qSLR was not able to identify proper evidence regarding the difference between context-aware software testing and traditional software testing. Still, randomized approaches are not a way to assure quality, since their coverage cannot be predicted nor does coverage warranty functional correctness (Wang et al., 2014).

A lack of consensus regarding several basic concepts has been observed. Five of the selected studies ((Tse et al., 2004), (Ryan & Gonsalves, 2005), (Alsos & Dahl, 2008), (Wang et al., 2010) and (Amalfitano et al., 2013)) provided no formal definition of context or context-awareness, so it was not possible to compare the definition presented in Chapter 2 with these studies. In addition, five of the studies ((Satoh, 2003), (Ryan & Gonsalves, 2005), (Alsos & Dahl, 2008), (Amalfitano et al., 2013) and (Canfora et al., 2013)) also do not provide the need of a Test Oracle in their testing process, making no clear how the test results are used in order to improve the software quality.

Another way to observe that context variance is not being addressed by the retrieved studies is that the ISO/IEC/IEEE 29119:2013 was capable of classifying almost all the studies regarding the Test Design Technique, suggesting that testing a context-aware software system is no different from the testing of a traditional system, or that no new approach has been presented so far. These classifications can be observed in Table 3.2 (Matalonga et al., 2015b).

ARTICLE	TEST DESIGN TECHNIQUE (ISO 29119)	TEST TYPE (AUTHOR'S OPINION)	TEST TYPE (ISO/IEC 25010)
Satoh2003	None	Interoperability Testing	Compatibility Testing, Interoperability Testing
Tse2004	Random Testing	Metamorphic testing	Functional Testing.
Ryan2005	Scenario Testing	Usability Testing	Usability Testing, Functional Testing
Merdes2006	Scenario Testing	Run-Time testing	Interoperability Testing
Jiang2007	Scenario Testing	None	Compatibility Testing
Wang2007	Scenario Testing	None	Functional Testing
Alsos2008	Scenario Testing	Usability Comparative Testing	Usability Testing
Wang2010	Syntax Testing	None	Compatibility Testing
Amalfitano2013	Error Guessing	Exploratory Testing	Procedure Testing
Canfora2013	Scenario Testing	User Experience Testing	Usability Testing
Wang2014	Branch Testing	Coverage-based Testing	Procedure Testing

Table 3.2 - Comparison among the recovered studies according to the ISO/IEC/IEEE 29119

The major problem in having several distinct definitions for the same concept is that maybe the different recovered studies might be researching different objects of study using the same names, resulting in noise and possible misjudgment while comparing the studies. In short, it is likely that they might not be comparable at all. In addition, it is important to consider that the table data was provided by the researchers' interpretation, which can differ whether applied by someone else.

As it was observed, the proposed software testing techniques do not consider context variance. In a real use scenario of CASS, the context can change before, during and after a user acting, and the system must be able to handle these variations.

Considering testing, the test process must enable the context to change freely as well, so that the software behavior in the field can be better reproduced during testing. In addition, test oracles/expected outputs must be aware of the possible contexts of usage, otherwise it will not be possible to make a comparison between the outputs and test oracles.

For instance, during the execution of one single test case, the outputs might change according to the context, even though the inputs are kept the same. Based on this, the hypothesis generated from this *quasi*-systematic literature review results is that the context variance influences not only the test item, but also the test oracle. Figure 3.3 illustrates the test execution process adapted according to this hypothesis.

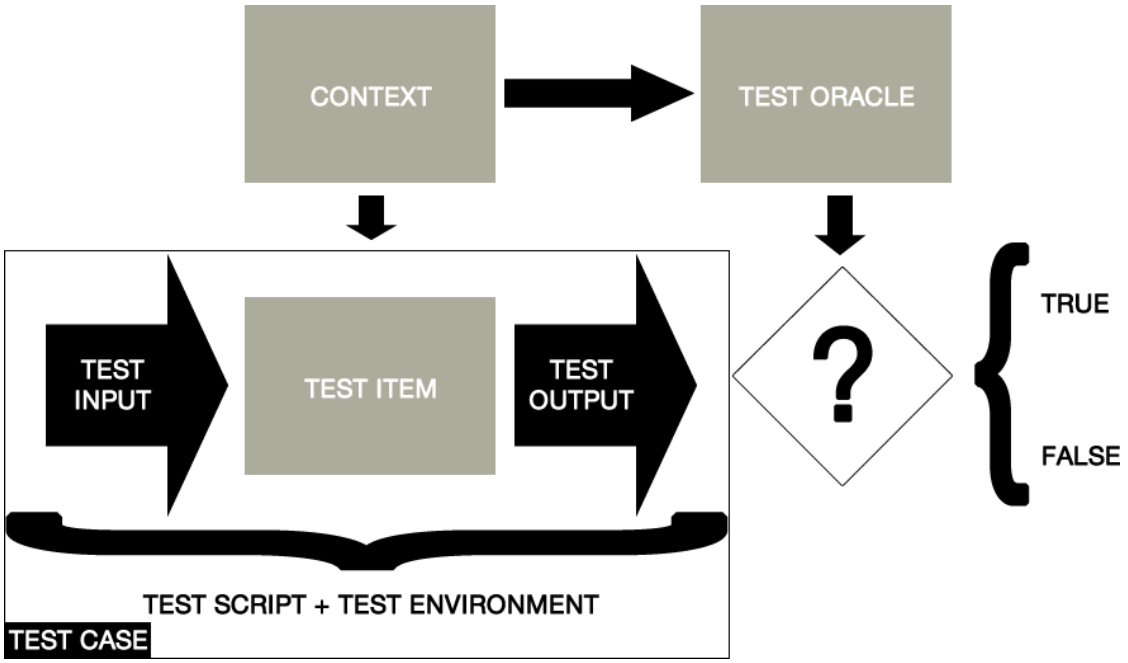

Figure 3.3 - Test execution process adapted from the CAcTUS Project hypothesis

In practice, it is possible to find some real examples of context-aware systems presenting failures based on context variance. For instance, the Android OS has a Camera application integrated with the operating system, which has the precondition of not executing with low battery (less than 10%) in order to save the remaining battery.

Nevertheless, if the user executes the application with more than 10% of battery and let the Camera application running until the mobile remains with less than 10% of battery and try to take a picture, the mobile freezes. This type of failure during the context variance indicates that this variation is not being considered while selecting the test coverage criteria, i.e. the perspective adopted is still to treat context variables as inputs.

In fact, trying to open the application with more than 10% of battery is possible and less is not possible, but the transition remains an issue.

3.7 Threats to Validity

Although the qSLR protocol followed well-established guidelines, a few threats to validity need to be taken into consideration. This section presents these threats and the actions taken in order to mitigate them.

- **Threat of missing literature:** The selection of three well-recommended and distinct databases and the execution of four distinct trials in order to improve the search string were the actions adopted to mitigate this bias. However, there is no way to affirm that there is no missing literature in this research. A snowballing could have helped to mitigate this bias.
- **Threat of selection bias:** Having all the three involved readers with the same power of decision and equal votes in the selection process helps to mitigate this bias. In addition, as presented in Table 3.1, when a study receives more than one Doubt vote, it was then discussed among the readers.
- **Threats of inaccuracy of data extraction:** This threat was handled by the iterative execution of the qSLR protocol during the four trials. The data extraction fields and process evolved during the protocol execution, however they were analyzed in pairs in order to mitigate this bias.
- **Bias on synthesis of information:** Using international standards / well established taxonomies as the ISO/IEC/IEEE 29119:2013 for the synthesis of information of the qSLR mitigates this threat.
- **Bias due to lack of control articles:** Since this qSLR was a characterization study, no control articles were used. Even calibrating the search string through the trial, this bias cannot be ignored.
- **Construct validity threat stemming from ISO/IEC/IEEE 29119 classifications:** The selection of the ISO/IEC/IEEE 29119 as the international standard to be used as taxonomy for the studies classification can be considered bias, since no comparison was made between the ISO/IEC/IEEE 29119 and other available taxonomies in the technical literature.

- **Construct validity on the classification of Test Types and Test Design Techniques of the selected primary sources:** The

classifications presented in Table 3.2 were made based on the three readers interpretations, which need to be considered as a bias.

3.8 Chapter Conclusions

This chapter presented a *quasi*-Systematic Literature Review (Travassos et al., 2008) conducted to find evidence about the particularities of testing context-aware software systems and how it differs from the traditional testing. The found studies consider context variables as a type of input in the test process, fixing the context variables values during the test process. The recovered approaches do not handle the context variance aspect of CASS, as presented in chapter 2.

In addition, using the ISO/IEC/IEEE 29119:2013 classification of Test Design Techniques and Test Type, it was possible to classify the found results, indicating that the testing for context-aware systems is no different from the traditional ones, or that no new approach was recovered from the literature by the proposed protocol.

A discussion was then raised considering the ISO/IEC/IEEE 29119:2013 dynamic test process and how to include the context variance into the described process. The proposed perspective is not achieved by any of the retrieved studies from the *quasi*-systematic literature review, encouraging the proposition of a test technique that takes that perspective into consideration.

Although no evidence was found in the computing domain to provide techniques for CASS testing considering context variance, the results have provided the perspective adopted by the computing domain so far. A hypothesis was then made in order to solve the issue. The next steps are to find if there are other domains with similar issues that share this same perspective. The next chapter brings evidence from other domains in order to propose a context-aware testing technique based on the hypothesis raised by this qSLR.

4 Towards a Context-Aware Test Process

Based on the results described in chapter 3, this chapter presents the issues found in other domains with similar properties when compared with testing for context-aware systems and how these domains managed to solve these problems. Based on the other domains retrieved concepts, a context-aware test process is proposed.

4.1 Introduction

In chapter 3, a *quasi*-systematic literature review was conducted in order to observe how context-aware systems have been tested. The retrieved results indicate that the issue of context variance is not being handled in the literature. Based on this, a hypothesis of how context can influence the test process has been asserted.

To identify some support to the asserted hypothesis, an *ad-hoc* web search was conducted in order to find domains presenting problems with similar characteristics of context influencing when compared to the proposed perspective. Therefore, two distinct domains were recovered and compared with the hypothesis.

After finding significant similarities between the proposed perspective and the found domains, the solutions observed in these domain were then adapted to the context-aware software testing scenario. To make it explicit, this chapter presents the steps conducted during this process, the findings and the adaptations made.

The following sections explain what have been observed in two distinct domains apart from the context-aware systems: Cybernetics and Organizational Resilience. Their perspectives are explained, as well as the similarities among their problems and the context-aware software systems. Finally, these approaches are adapted to the context-aware systems domain and then the obtained knowledge is evolved according to the CASS needs and defined in the form of a process.

4.2 Gathering Information from other Domains

A simple *ad-hoc* web search was conducted using some of the keywords presented in the search string in Chapter 3. The idea was to have a wider perspective on which domains deal with these same issues without following the formality of a qSLR or being limited by the search engines. Gathering keywords from all the levels of the PICO structure, it was possible to retrieve a domain called **Cybernetics**. In order to find it, a few keywords were used and adapted during the web search process, the keywords responsible for the findings were:

- “self adaptive” OR “self adapt”
- "system test" OR "system testing"
- "model" OR "approach" OR "technique"

After some research into the Cybernetics domain, the concept of **Resilience** has emerged among the discussions on how to handle system’s variances. Therefore, looking deeper into the Resilience perspective, it was possible to find a domain called **Organizational Resilience**, which applies the Resilience concepts at a social level of management.

Even though their definitions of context and system were distinct from the ones in context-aware systems, it was possible to observe similar types of issues to be handled: systems suffering influence from different sources and having to adapt to these changes. These two domains and their similarities with the presented problem are presented below.

4.2.1 Cybernetics

Wiener (1961) defined *cybernetics* as the science of control and communication, in the animal and the machine. The idea behind this definition is to know the main objective of the system and have the ability to control the system in order to maintain it. For instance, Wiener studied biological systems while trying to design smart missiles capable of following a target during the Second World War. Wiener believed that the only way to handle control and communication in a machine was simulating a biological system.

Later, Beer (1981) stated that cybernetic systems must be able of resuming a steady state after it has been disturbed in a way not envisioned by its designer. This can be observed as a system susceptible to suffer external disturbances and still maintain its behavior. As an example of application, Beer uses the cybernetics principles to support organizational management, i.e. how organizations can handle external adversities and keep working.

Adapted from the basic *cybernetics*, Harries-Jones (1988) defined the *new cybernetics* as the autonomous and self-organizational capabilities of complex systems. Based on Beer (1981) definition, Harries-Jones not only stated that this type of system must be able to recover itself from disturbances, but also be able to evolve based on these disturbances whether necessary, i.e. with or without disturbances, the system is supposed to behave as expected. This perspective can be observed in Figure 4.1.

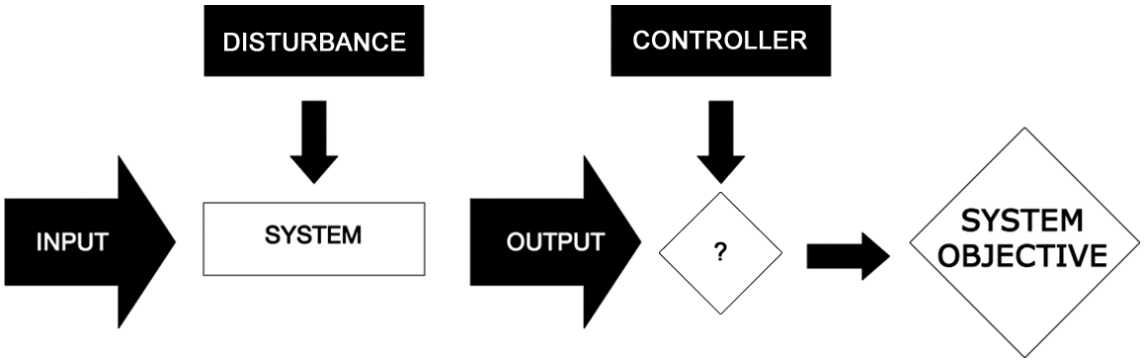

Figure 4.1 - Cybernetic system behavior perspective

In respect to disturbances, Ashby defined them as something that “moves a system from a state to another” (Ashby, 1956). Later, realizing that Ashby’s definition was too general, Forman and Godron adapted this definition to “an event that causes a significant change from the normal pattern in a system.” (Forman & Godron, 1986).

Therefore, a cybernetic system can be seen as a system that deals with inputs and outputs (effects from a disturbance or stimulus in the environment). A disturbance is a factor that needs to be handled in a way that does not affect the outputs. For this, it is supposed to have a controller to regulate the impact of the disturbance in order to maintain the system objective.

4.2.1.1 Mapping Cybernetics to Context-Aware Systems

Making a comparison between Cybernetics and CASS, the concept of disturbance referred in the Cybernetics domain can be interpreted as a context variation in the CASS domain. For instance, a human body can be seen as an enormous cybernetic system with several cybernetic subsystems. Consider specifically the human thermal system as one of these. It aims to keep the individual body temperature about 36.5 Celsius degrees.

If a disturbance lowers the temperature, the individual body starts to consume fat in order to keep it warm, and if a disturbance raises the temperature, the individual body sweats in order to cool it down. In that way, the fat and sweat are the instruments used by the controller (human thermal system) in order to keep the system objective (the individual body around 36.5 Celsius degrees).

In addition, it is important to remember that temperature variance during long periods can compromise the system. In this example, the only possibility of the thermal system to control the temperature is based on the body’s own temperature, i.e. it does not consider other possible factors. If the person has a fever for instance, even with the thermal system trying to lower the temperature, the issue will only be solved once the immune system (another system) deals with the fever cause.

This illustrates that even being able to control known features, a cybernetic system does not adapt to the changes in order to evolve the system. In a context-aware system, a possible way to solve the issue would be finding a way to keep the entire system working even with the temperature variance.

The presented example shows similarities and divergences between CASS and Cybernetic Systems. However, the systems described in the Cybernetics domain can be social, biological or mechanics, and the systems proposed as context-aware are exclusively software systems. Although the concepts seemed to be related, no practical approach was found in the cybernetics domain that could deal with the disturbances apart of controller actions.

4.2.2 Organizational Resilience

One of these concepts is proposed by Ashby (1956) as the *Law of Experience*. It states that the more ways you have to make a system resilient, the more resilient it will be. It means that, if a system has several ways to handle a disturbance, it will be more likely to not present failures regarding this disturbance.

In fact, several domains exist based on the Resilience concept, for instance the Organizational Resilience (which handles the social systems), Resilience Engineering (which handles the constructions), Ecological Resilience (which handles the environmental systems), and others.

Following Ashby's path, several authors defined resilience according to their domain of action. Holling (1973) defined it as a persistence of systems measure and of their ability to absorb changes and disturbances and still maintaining the same relationships between populations or state variables.

Folke et al. (2010) defined it in a simpler way saying that resilience is the capacity to change in order to maintain the same identity. Finally, Walker & Salt (2012) stated resilience as the ability of a system to absorb disturbances and still retain its basic functions and structure.

The system identity is how the system is supposed to behave, similar to Beer's definition (1981). The Cybernetics domain calls it System Objective. The Cybernetics domain expects the system output to be always the same, independent of the disturbances. The Organizational Resilience domain wants to compare the system identity with the output in order to keep improving the system until it can handle all possible disturbances by itself. A representation of this can be observed in Figure 4.2.

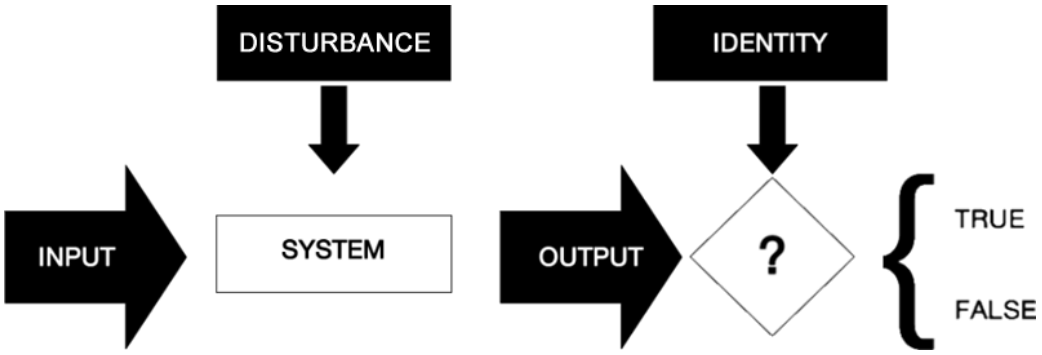

Figure 4.2 - Resilient systems behavior perspective

4.2.2.1 Mapping Organizational Resilience to Context-Aware Systems

In contrast to Cybernetics, Organizational Resilience shows a few practical approaches to help the system to maintain its identity. For instance, Walker & Salt (2012) defined the concept of “threshold” as being the level of “disturbance capable of changing the system identity”. In order to maintain this identity, the resilient practices are used to identify the thresholds so they could be avoided. Not all disturbances are capable of reaching thresholds, but all thresholds are reached by disturbances.

Returning to the human body system example, it can be also interpreted as a resilient system. For instance, the system identity is to keep itself healthy. The temperature change is observed as a disturbance and the temperature change for long periods can compromise the human body system healthy, being considered then a threshold, i.e. if the system temperature stays different from 36.5 Celsius Degrees for long periods of time, the system identity is lost.

According to Walker & Salt (2012), a system resilience can be divided into two groups: General Resilience and Specific Resilience. The first is the “system’s capacity to manage a disturbance and still avoid reaching a threshold”. It means that the General Resilience is concerned with the system’s identity. The second one is the “resilience of some part of the system to particular kinds of disturbance”.

In the human body system, when it gets fever, even with all sweating effort by the thermal system, the temperature can only get lower by the aid of the immune system. It means that when the system gets a fever, the Specific Resilience of the thermal system suffered a disturbance, but the General Resilience of the system did not. It will only happen if the immune system cannot handle the fever cause after a long period. In the case where the human body system gets the same virus that can cause a fever twice, the immune system already have learned how to handle that disturbance and it will not be a potential threshold anymore.

As exemplified, to assure the system General Resilience, the resilience practices try to identify thresholds in order to avoid them. Walker & Salt (2012) proposed techniques to support thresholds identification. Being able to identify the spots that compromise the Specific Resilience and, as a result, the General Resilience, it will be possible to avoid them. Nevertheless, some thresholds might not be found in this process. The proposed process consists of four steps:

1. **List the known thresholds**

   In the first step, the known thresholds are listed. They might be known for tacit knowledge, previous experience or just guessing.

   In the human body system example: A virus that already had infected the system is a known threshold, since the immune system already knows how to handle it.

2. **Enumerate the thresholds of potential concern**

   Based on the thresholds found in the first step, look for thresholds that can affect the general resilience directly.

   In the human body system example: A virus that causes fever can compromise the thermal system Specific Resilience. However, a disease that makes the heart to stop compromises the entire system identity directly (General Resilience), since all other system parts will fail if this happens. Therefore, this kind of disease is a threshold of potential concern.

3. **Reproduce the system in a conceptual model**

   The third step creates a model showing how the system is supposed to behave, i.e. each possible system's state or usage situation is listed in separated boxes. These boxes are connected among them with two types of arrows, grey and black. The grey ones represent passive transitions, i.e. transitions happening without an actor intervention, as the battery consumption for instance. The black ones represent an actor intervention, like a user pressing a button. This model is very similar to State Machines models usually used in Software Engineering (Pressman, 2010).

   In the human body system example: Figure 4.3 shows the conceptual model for the thermal system example.

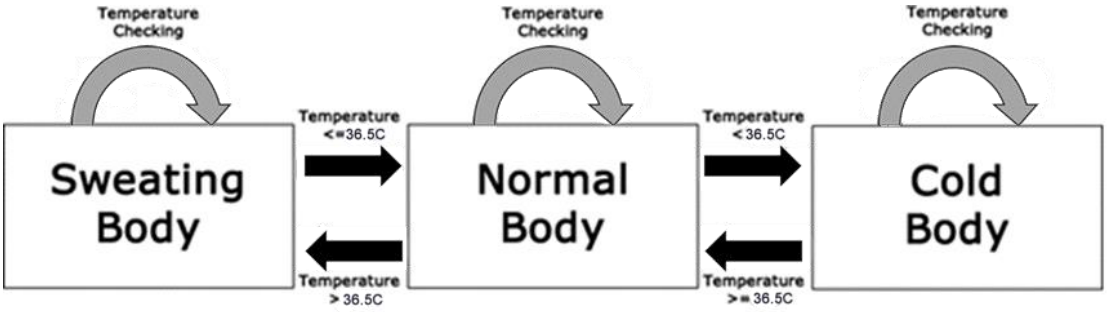

Figure 4.3 - Thermal system conceptual model

4. Reproduce the system in an analytical model

The fourth step consists of listing every action the system can do and describe its impacts to the system, if possible with the aid of experts in the area. Analyzing these two generated mental models, it is possible to find unknown thresholds to be considered from now on.

In the human body system example: Table 4.1 shows the analytical model for the thermal system example.

ACTIONS	IMPACT ON THE SYSTEM
SWEAT	Reduce body temperature
CONSUME FAT	Increase body temperature

Table 4.1 - Thermal system analytical model

The idea behind this process is to know as many disturbances and thresholds as possible, so actions can be prepared to make the system to avoid them, so it will maintain its identity. Remembering the similarities and differences between CASS and Cybernetics presented in the last section, Table 4.2 includes the Organizational Resilience domain in the comparison.

DOMAIN / ATTRIBUTE	CONTEXT-AWARE SOFTWARE SYSTEMS	CYBERNETICS	ORGANIZATIONAL RESILIENCE
Definition	A system with the dynamic property of adapting its behavior according to the context (Adapted from CAcTUS Project).	“A system capable of resuming a steady state after it has been disturbed in a way no envisioned by its designer.” (Beer, 1981)	“The ability of a system to absorb disturbances and still retain its basic function and structure.” (Walker & Salt, 2012).

Objective	Be able to adapt its behavior in order to keep the system working as expected based on the contextual variances (Adapted from CAcTUS Project).	“With or without disturbances, the system is supposed to behave as expected.” (Harries-Jones, 1988).	Maintain the system general resilience (system identity).
How it calls the factors that can compromise the system objective?	Context	Disturbance	Threshold
Factor definition	Any piece of information that may be used to characterize the situation of an entity (logical and physical objects present in the system’s environment) and its relevant relations for the actor-computer interaction (Adapted from CAcTUS Project).	“An event that causes a significant change from the normal pattern in a system.” (Forman & Godron, 1986)	“Disturbance capable of changing the system identity.” (Walker & Salt, 2012).
How it anticipate these factors in order to achieve the proposed objective?	Software testing and Verification and Validation techniques	Controller actions	Thresholds avoidance techniques
What are the limitations of the way used to handle the factors?	Lack of coverage, no evidence was found to support the existence of a technique to test CASS considering the context variance.	The controller must be aware of the possible changes, otherwise it will not be able to take actions to make the system recover.	The techniques help the thresholds identification, but not assure coverage. By knowing them, controller actions can be performed beforehand.

Table 4.2 - Comparison between CASS, Cybernetics and Organizational Resilience

Considering the similarities among Organizational Resilience and CASS, the next section explores these common features and try to adapt the disturbances and thresholds identification process to the CASS reality.

4.3 Adapting other Domains Concepts to CASS

The previous section discussed other domains aside from CASS trying to solve similar issues: guarantee the functionality of systems that are likely to be affected by undesired factors. As presented in Table 4.2, the differences among these factors are: the **context** changes the way the system behave and the **disturbance** may influence the system behavior, leading it to reach a **threshold**.

In specific cases, a **context** can be observed as a **disturbance** or **threshold** if the system is not prepared to handle it. For instance, an Android Camera application is not supposed to launch with less than 10% of battery left. However the Android OS cannot handle the situation of opening the Camera application with little more than 10% of battery left and keep it running until the battery goes below 10%. This situation crashes the application. The battery consumption is a **disturbance**, and in the context of the camera application, the transition from 10% to 9% of battery during the application execution is a **threshold**.

This example illustrates that, when considering situations where the system is presenting failures, the three presented domains (CASS, Cybernetics and Organizational Resilience) share exactly the same issue, i.e. the testing of CASS should not be different from the testing of Cybernetics and Organizational Resilience.

Nevertheless, the concept of testing does not exist in the Cybernetics and Organizational Resilience domains. In Cybernetics, a controller must judge what to do every time a **disturbance** affects the system. In Organizational Resilience, the **disturbances** are evaluated and strategies are developed to make the system avoid possible **thresholds**. In the thermal system example, this would be similar to taking a vaccine to prevent a disease that could cause fever.

Since the different possible **contexts** that may affect CASS could lead to situations that cause the systems to fail, we assume this kind of situations can be handled in the same way Organizational Resilience deals with **disturbances** that can lead to **thresholds**.

Even with distinct objectives, both domains have interest on knowing which factors (**context** and **disturbances**) influence the system and how they do. Based on this, the following section proposes to adapt the thresholds identification process from the Organizational Resilience domain to the CASS testing.

4.4 A Context-Aware Test Suite Design

As presented in section 2.2.2, the testing process adopted by the ISO/IEC/IEEE 29119:2013 consists of an organizational level, a management level and a dynamic level. At the dynamic level, it is possible to observe the proper execution of the test itself based on the test plan generated in the management level, obeying the rules stated in the organizational level.

The assumption behind this proposal is that it is possible to adapt the threshold identification approach presented in the last section in order to consider the context variance during testing. It is also possible that other perspectives could be chosen to guarantee the expected coverage during the test process. In that case, the proposed process would be complementary, achieving the coverage of contextual variance.

In order to propose an approach capable of handling the context variation based on the thresholds identification technique proposed by Walker & Salt (2012), two examples of CASS are presented and used as application scenarios. The thresholds identification process is then adapted to reach the CASS domain needs.

The following sections present the examples chosen to observe the testing technique behavior, the construction methodology adopted for the adaptation of the thresholds discovery approach to the CASS testing technique and the process evolution during the trials. The final version of the CASS testing process was named CATS Design – A Context-Aware Test Suite Design.

4.4.1 Material Selected for CATS Design Evaluation

During the adaptation process from the Organizational Resilience domain to the testing of CASS, several aspects of the CATS Design process needed to be evaluated in order to guarantee that they were actually contributing on the improvement of testing coverage of CASS.

For this evaluation, two examples of applications were selected and three trials were needed in order to make the CATS Design process to achieve the intended purposes, which is improving the test coverage of context-aware systems. The examples selection was made by convenience, trying to choose real world applications considering the CASS reality. This section presents a summary of each of these two projects.

4.4.1.1 Smart House

The first attempt to adapt the Organizational Resilience model to the testing of context-aware systems was made using an *ad-hoc* approach with a non-formalized project. It describes a system for a smart house proposed by Schilit (1994) and adapted for research purposes. The example is presented below.

"A client requests a system for turning his/her house into an intelligent house, so that s/he can see what happens inside it, whether s/he is in the house, or outside, say, at work. Besides, the client wants the house lights that automatically switches on and off when a person enters and leaves a room; that the garden waters automatically every day, at a certain time; room temperature regulate itself reaching a maximum of 24◦C, approximately. Lastly, the client wants that during holiday periods, lights and television sets switch on and off periodically, and windows open and close automatically so that the house seems in use, and thus, the owner prevents burglary."

4.4.1.2 Smart Camera

The Smart Camera system was proposed by the author and designed to be an embedded system for a digital camera focused on making the user experience with the device even more comfortable and optimized. The system was projected to maximize the user capacity of taking photos with better quality and minimum time spent, using sensors and intelligent behavior to adjust the software to better adapt to the sensors readings.

This system allows the camera to adjust its behavior according to the environment in order to save battery and optimize the photo taking activity. The user is able to take single pictures, several pictures with one single command and make the camera follow a selected target to always have the best possible focus.

The goal of this project was to be used as a testing project to improve the CATS Design process. This project was never meant to become a real project, however, a requirements documentation and a non-context-aware test documentation were developed in order to support the CATS Design construction process. This documentation can be found in APPENDIX A – Smart Camera Requirements and APPENDIX B – Smart Camera Non-Context-Aware Test Suite.

4.4.2 CATS Design Construction Methodology

In order to observe if the adaptations made from the Organizational Resilience domain to the testing of context-aware systems were consistent, the CATS Design process was applied to the Smart House example once and to the Smart Camera project twice.

Based on the generated test suite, the non-context-aware test suite of the Smart Camera project was then compared to the context-aware one generated by the CATS Design process. From this, the difference of coverage between the CATS Design process and the traditional testing techniques has been observed.

Having a consistent model ready, it was then applied to a new project, the CAUS – Context-Aware University System (Castellanos, 2015). This last step is a proof of concept used to evaluate the final shape of the CATS Design process and is presented in the next chapter. This methodology can be observed in Figure 4.4. The next section shows the process construction.

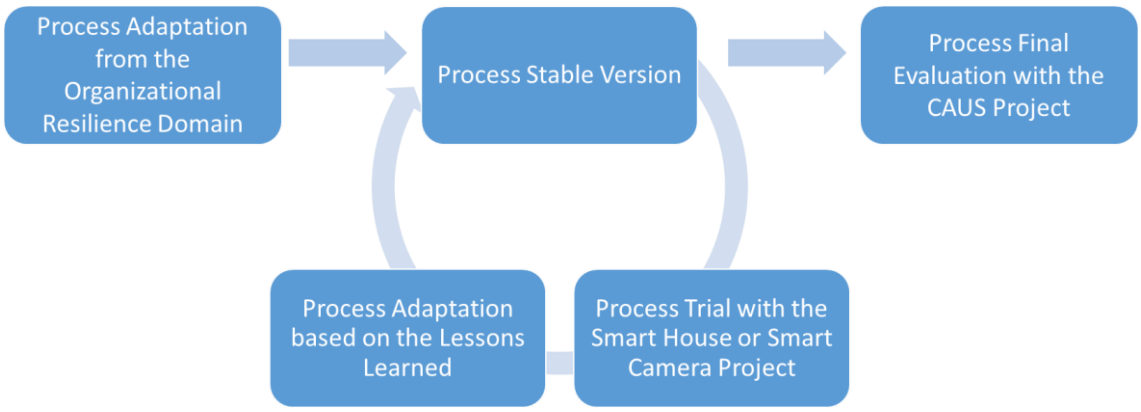

Figure 4.4 - CATS Design Construction Methodology

4.4.3 Initial Version

The four steps presented by Walker & Salt (2012) to find possible thresholds were adapted for CASS. The general adaptations are presented below and the first version of the process can be observed in Figure 4.5.

1. **List the known thresholds**

   This step was divided into two, one concerned with the identification of thresholds in the requirements documentation and other by applying tacit knowledge to support their identification. By doing so, it intends to reduce the chances of missing a threshold by an error in the requirements or by a human mistake.

2. **Enumerate the thresholds of potential concern**

   In Organizational Resilience, thresholds of potential concern are the ones more likely to make the system to lose its identity (General Resilience). In CASS testing, it was considered the context variables values that are more likely to cause changes in the system. For instance, a mobile compass application can read the Wi-Fi signal, but it is not concerned with it. The changes in the user position are more relevant in this scenario.

3. **Reproduce the system in a conceptual model**

   This step was kept the same, since it helps the discovery of unknown context variables and their respective values.

4. **Reproduce the system in an analytical model**

   This step was kept the same for the same reason of step 3. However, the list of actions was trade for a list of context variables in order to help their discovery.

The first version of the CATS Design process is concerned with the ability of finding thresholds and discovering how such thresholds can actually affect the system's behavior. Therefore, no test case template has been proposed in the first version of the process yet. The main point was to confirm if the process adaptation were capable of finding context variables not identified beforehand and their respective values.

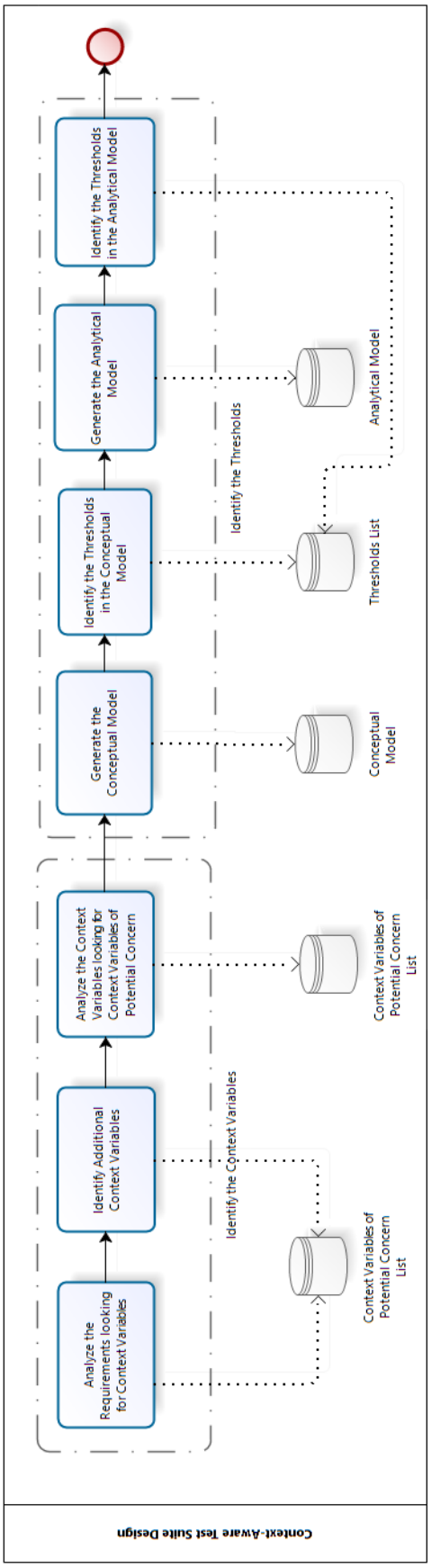

Figure 4.5 - CATS Design Process version 1

4.4.4 Construction Trials

This section presents the steps conducted during the CATS Design construction and the process evolution. The trials were applied during the CATS Design process adaptations, so the steps are not always consistent among them. However, this divergence was important for the process construction.

4.4.4.1 Trial 1: Smart House

Based on the process previously presented in Figure 4.5 and using the Smart House example, a list of context variables was identified (see Table 4.3). Some of the context variables were collected from the requirements specification and are presented in the first row, the other ones were proposed by the author based on his tacit knowledge and are shown in the second row. The context variables of potential concern are marked in bold.

Context Variables List
• Water Pump State • **User Presence** • Temperature Level • **Time Instant** • Electrical System
• Lights state • TV state • Windows State • **Internet Connection** • **Water** • **Power**

Table 4.3 - Context variables from the Smart House example

Exemplifying the concept of context variables of potential concern, **Water** and **Power** were considered to be context variables of potential concern in this example. If the system lacks water or power, several functionalities will not be completed. The **Time Instant** activates and deactivates several functionalities and are likely to influence other system's functionalities.

In addition, the **User Presence** actions can coincide with **Time Instant** actions causing conflict. Finally, state changes must be notified to the user by the **Internet Connection**, so it needs to be considered a context variable of potential concern as well.

Having selected these context variables, the thresholds identification phase begin with the creation of both conceptual and analytical models, respectively. The conceptual model is presented in Figure 4.6 (freehand) and Figure 4.8 (transcribed). The analytical model is presented in Figure 4.7 (freehand) and Table 4.4 (transcribed).

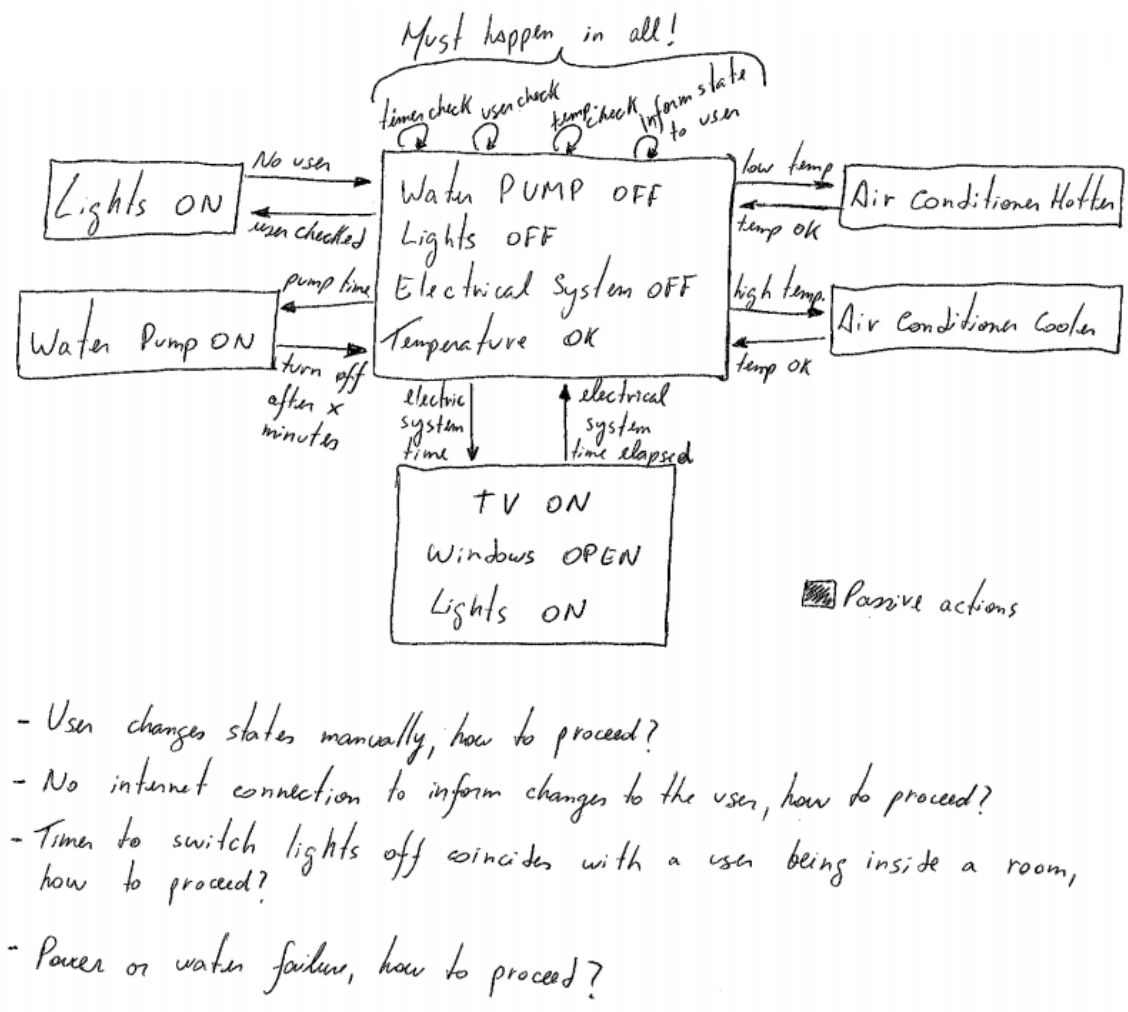

Figure 4.6 - Conceptual model for the Smart House Project

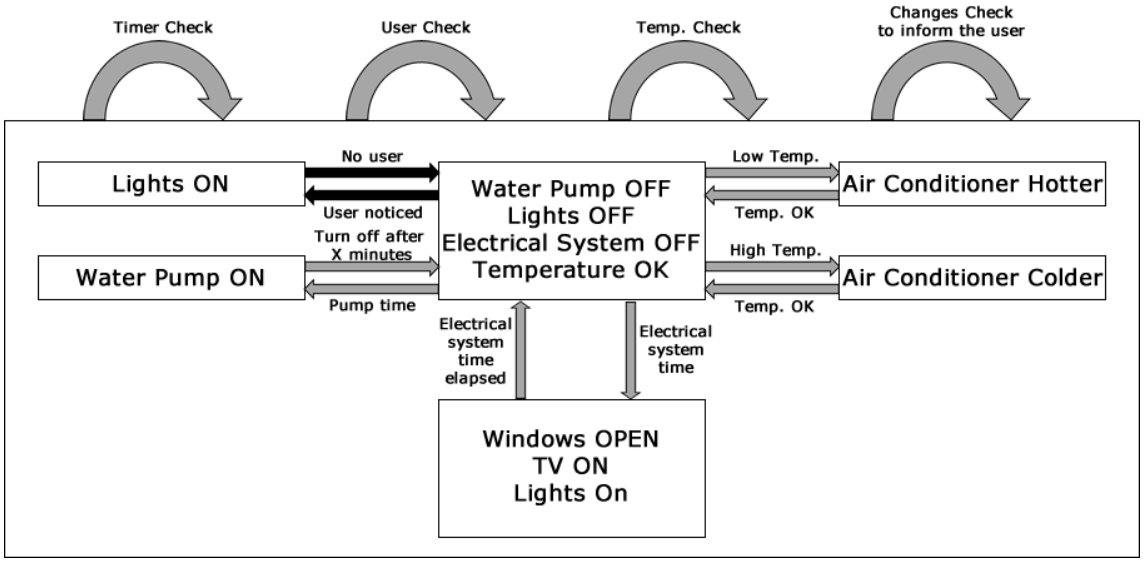

Figure 4.7 - Transcribed Conceptual model for the Smart House Project

Context Variable	Effect
User presence	Turn ON lights if find presence
Timer	Turn several states ON/OFF as programmed, for instance water pump, lights, TV and windows (OPEN/CLOSED).
Temperature Level	Check if the temperature is above or below the programmed. Changes the environment to a colder state if the temperature is too high or to a hotter state if it is too cold.
Electrical System	Controls the state of TV, Windows and lights.
Water Pump state Lights Windows TV	ON/OFF according to the timer
Internet Connection	Send the house states to the user

Figure 4.8 - Analytical model for the Smart House Project

Context Variable	Effect
User Presence	Turn ON lights if finds a user presence
Timer	Turn several states ON/OFF as programmed, for instance water pump, lights, TV and windows (open/closed).
Temperature Level	Check if the temperature is above or below the programmed. Changes the environment to a colder state if the temperature is too high or to a hotter state if it is too cold.
Electrical System	Controls the state of TV, Windows and Lights
Water Pump State Lights Windows TV	ON/OFF according to the timer
Internet Connection	Send the house states to the user

Table 4.4 - Transcribed Analytical model for the Smart House Project

Since this first version did not have the test suite design features yet, the final step was to list the thresholds found in the process, which are shown below. In this iteration, it was possible to observe that the step when the context variables of potential concern were identified had no purpose, since the models had to consider all the possible context variables.

- **Thresholds for the Smart House Project**
  - Try to turn ON the **water pump** without **water**
  - Try to accomplish **any function** without **power**
  - Try to accomplish **any function** without **internet connection**
  - Try to accomplish **any timer function** with an **user interference**

4.4.4.1.1 Trial 1: Process Adaptations

In the first trial, similarly to the Organizational Resilience domain, all context variables were identified and the ones that seemed to be more relevant were classified as of potential concern. However, all listed context variables are relevant. Context variables such as pressure, that could be measured, but have no relevance for the system, were already excluded. So the first impression was that there is no point of classifying the context variables of potential concern.

In addition, some thresholds were made by the combination of two or more context variables, so not always the problem was inherent to the context variables of potential concern. However, since the used scenario was not represented by a complete requirements documentation, the step of classifying context variables of potential concern was not removed in this first iteration.

Another issue considered was that having a list of thresholds was not enough to prepare a test suite. Based on this, the idea in the second iteration is to adapt a non-context-aware test suite to become context-aware using the list of thresholds. Finally, the pre-requisite of having a non-context-aware test suite for the project was added in order to compare the coverage of the processes.

Moreover, the initial idea was not to have a proper test case template for the CATS Design, but instead, adapt an already existing non-context-aware test suite to become context-aware. For this, each test case from the non-context-aware test suite had to receive a test oracle (considering the thresholds and context variables), a thresholds list and a context variables list. This second version can be observed in Figure 4.9.

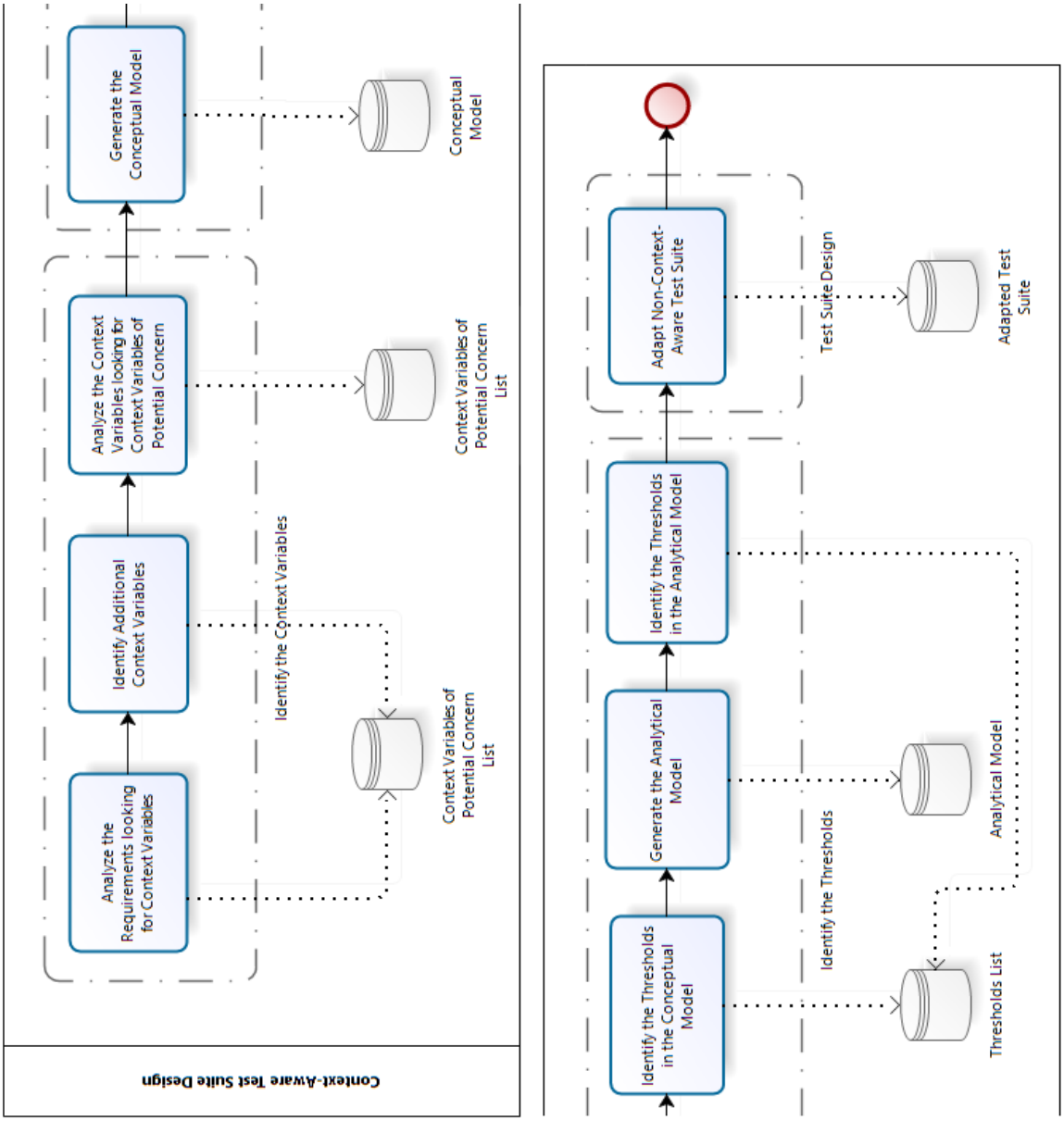

Figure 4.9 - CATS Design Process version 2

The next section shows the second trial, conducted with the Smart Camera Project, which received the step of adapting a non-context-aware test suite into a context-aware one.

4.4.4.2 Trial 2: Smart Camera

The second trial evolved based on the first evaluation. Although the concept of context variables of potential concern seemed to be useless for the process, it was kept for one more trial of experimentation. Together with it, the last step of the process received a more detailed description of what to do with the thresholds list, i.e. adapt a non-context-aware test suite to become context-aware, instead of just having it as a result. However, it does not have a proper test case template yet.

This second version of the process was applied to the Smart Camera project (see APPENDIX A – Smart Camera Requirements). Figure 4.10 shows the execution of steps 1, 2 and 3, characterizing the first phase: Context variables identification. Step 1 is the identification of context variables from requirements, Step 2 is the inferring of context variables by the tester and Step 3 the identification of context variables of potential concern. The context variables are transcribed below.

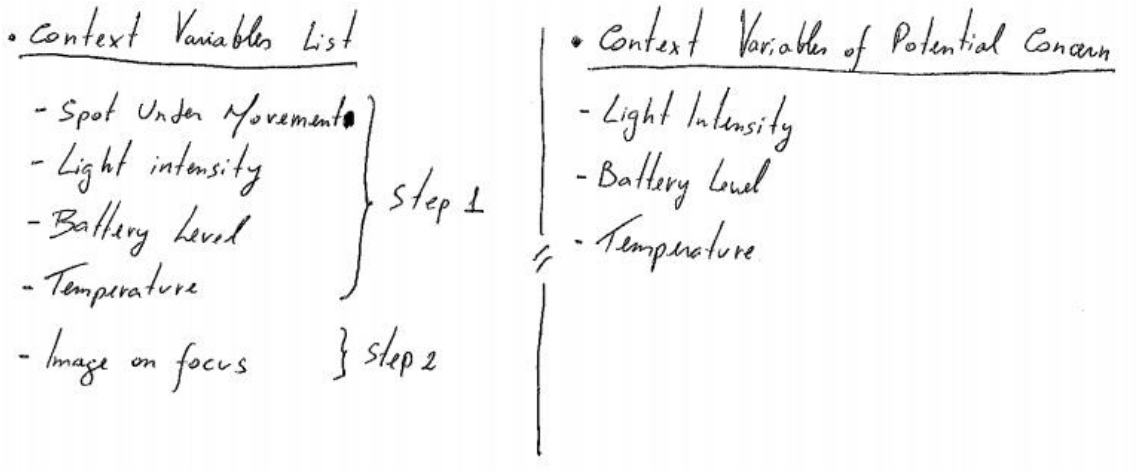

Figure 4.10 - Context variables identification for the second trial

- **Context Variables – Step 1**
  - Spot Under Movement
  - Light Intensity
  - Battery Level
  - Temperature
- **Context Variables – Step 2**
  - Image on focus
- **Context Variables of Potential Concern – Step 3**
  - Light Intensity
  - Battery Level
  - Temperature

From these context variables, the conceptual and analytical models were derived with the aid of the requirements documentation. Figure 4.11 shows the conceptual model and Figure 4.13 the analytical model. Figure 4.12 shows the transcribed conceptual model and Table 4.5 shows the transcribed analytical model.

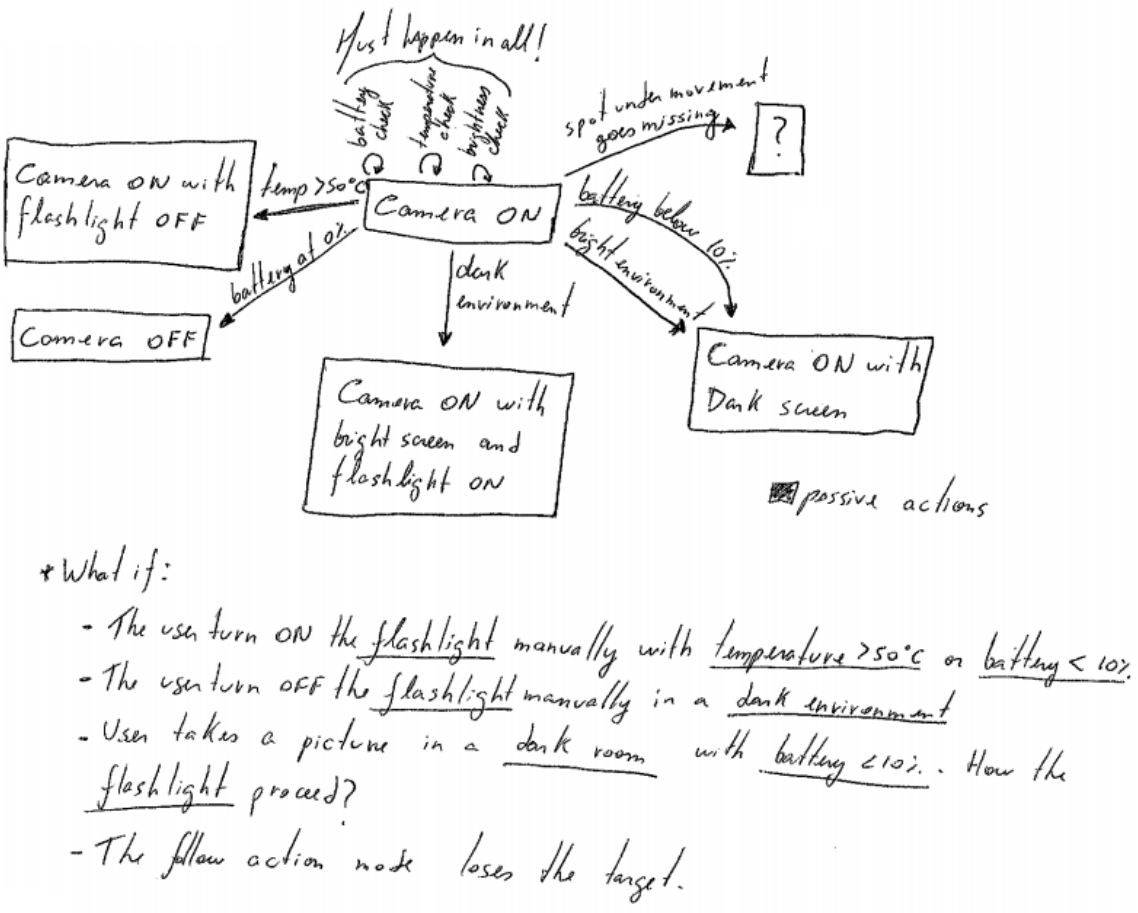

Figure 4.11 - Conceptual model for the second trial

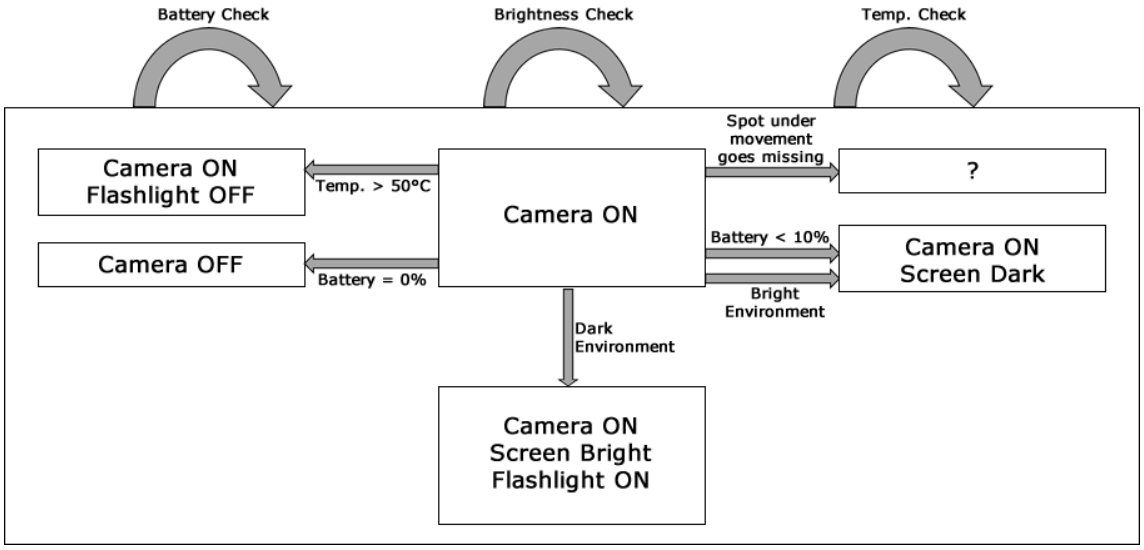

Figure 4.12 - Transcribed Conceptual model for the second trial

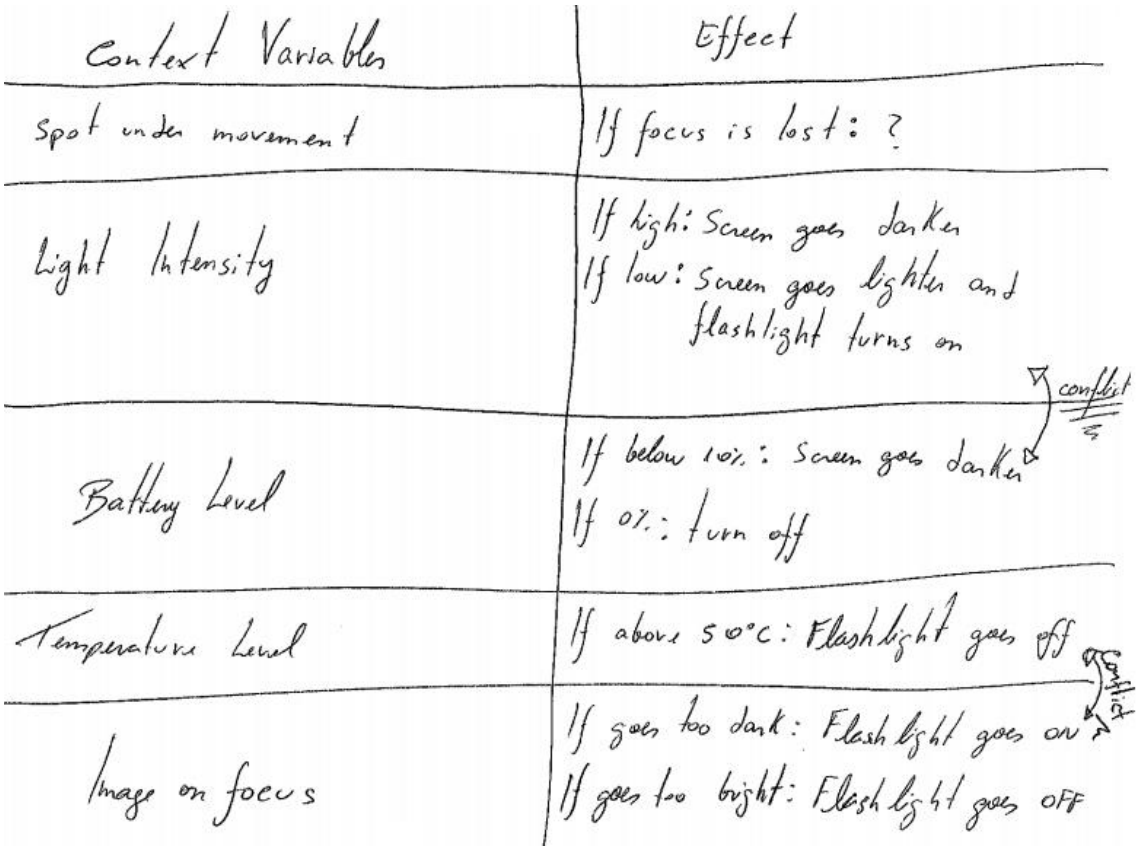

Figure 4.13 - Analytical model for the second trial

Context Variable	Effect
Spot Under Movement	If focus is lost:?
Light Intensity	If high: Screen goes darker If low: Screen goes brighter and flashlight goes ON
Battery Level	If below 10%: Screen goes darker If 0%: Turn OFF
Temperature Level	If above 50 °C: Flashlight goes OFF
Image on Focus	If goes too dark: Flashlight goes ON If goes too bright: Flashlight goes OFF

Table 4.5 - Transcribed Analytical model for the second trial

From the models, the thresholds were derived. Having the list of context variables and thresholds, the provided test documentation for the Smart Camera project was adapted with new test oracles including the context variables and thresholds to them. The test documentation for the Smart Camera project can be found in the APPENDIX B – Smart Camera Non-Context-Aware Test Suite.

4.4.4.2.1 Trial 2: Process Adaptations

From these results, it becomes clear that the step of classifying context variables of potential concern had no use for the process. Also, this second trial led to the observation that not always a non-context-aware test suite will be provided. And even if it is provided, the findings in the list of thresholds might support the creation of test cases that may not exist in the non-context-aware test suite. Therefore, the CATS Design process must have a proper test case template.

A first test case template was designed based on the traditional testing techniques. The fields included in the template are:

- **Test Suite ID:** Identification of the test suite
- **Test Case ID:** Identification of the test case
- **Use Case Base:** Use Case used as basis for the test case
- **Test Objective:** Objective of the test case
- **Precondition:** List of conditions that must be satisfied before the test case is executed
- **Test Inputs:** List of inputs of the test case
- **Test steps:** Steps to be performed during the test case execution
- **Test Expected Outputs:** List of test oracles of the test case
- **Post-Condition:** List of conditions that must be satisfied after the test case is executed

In addition, the step of finding context variables of potential concern was removed and the pre-requisite of having a non-context-aware test suite was removed as well. Another relevant point was that not always is possible to create a full list of test oracles, implying in a test case with no expected output. This occurs because of a requirements problem, but is documented as a test result. These adaptations can be observed in Figure 4.14.

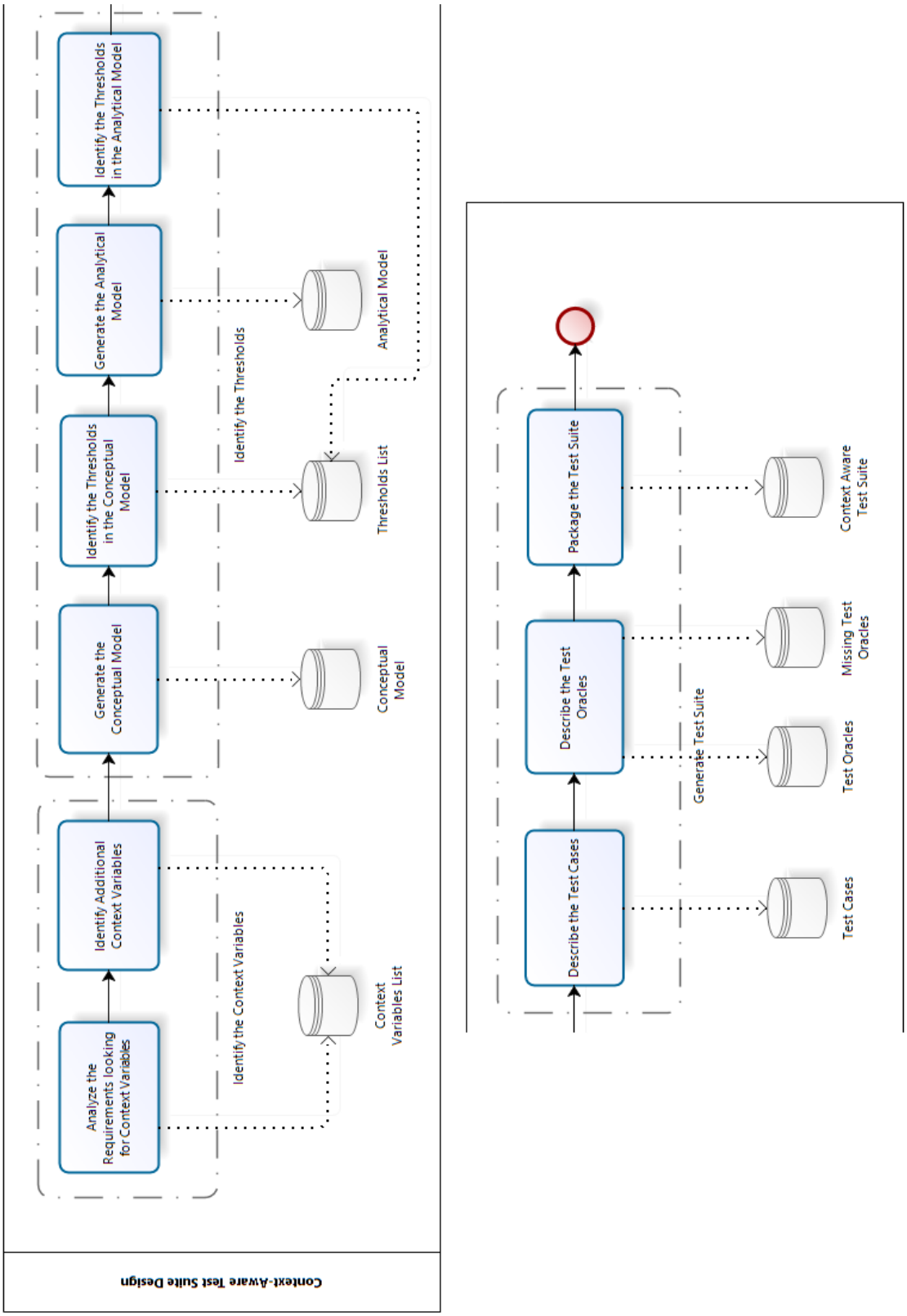

Figure 4.14 - CATS Design Process version 3

4.4.4.3 Trial 3: Smart Camera

Once the first steps remained the same, the context variables had no change between the second and third trial. However, with a deeper look into the requirements, the variables **Image of Focus** and **Spot under Movement** were noticed to be handling the same concept. So these two variables were merged into one as shown in Figure 4.15 and transcribed below.

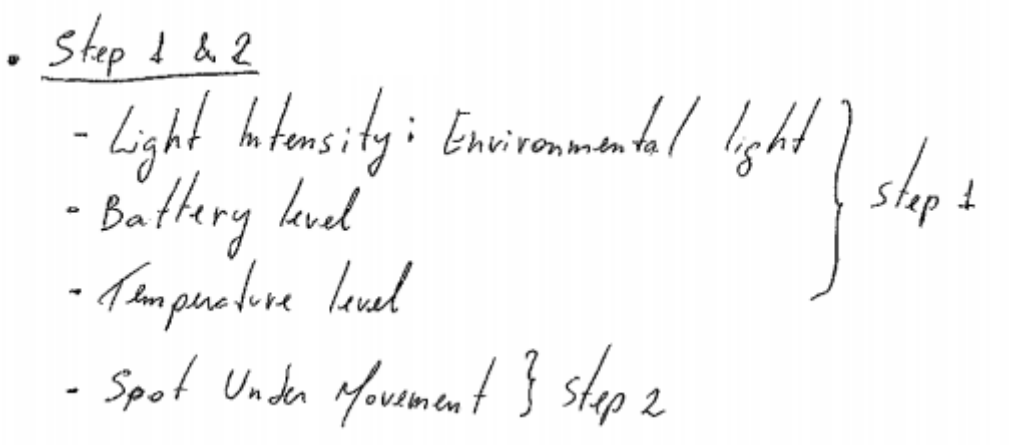

Figure 4.15 - Context Variables Identification phase for the third trial

- **Context Variables – Step 1**
  - Light Intensity: Environmental light
  - Battery Level
  - Temperature Level
- **Context Variables – Step 2**
  - Spot Under Movement

The conceptual model and analytical model remained the same. In order to exercise the models design, they were rebuilt for the third trial. Even though they seemed distinct from the ones in the second trial, the identified thresholds did not change. This exemplifies that there is no unique way to create the models. The conceptual and analytical models are presented respectively in Figure 4.16 and Figure 4.18 and transcribed in Figure 4.17 and Table 4.6. The list of all the findings of the process is shown in Figure 4.19 and is transcribed below it.

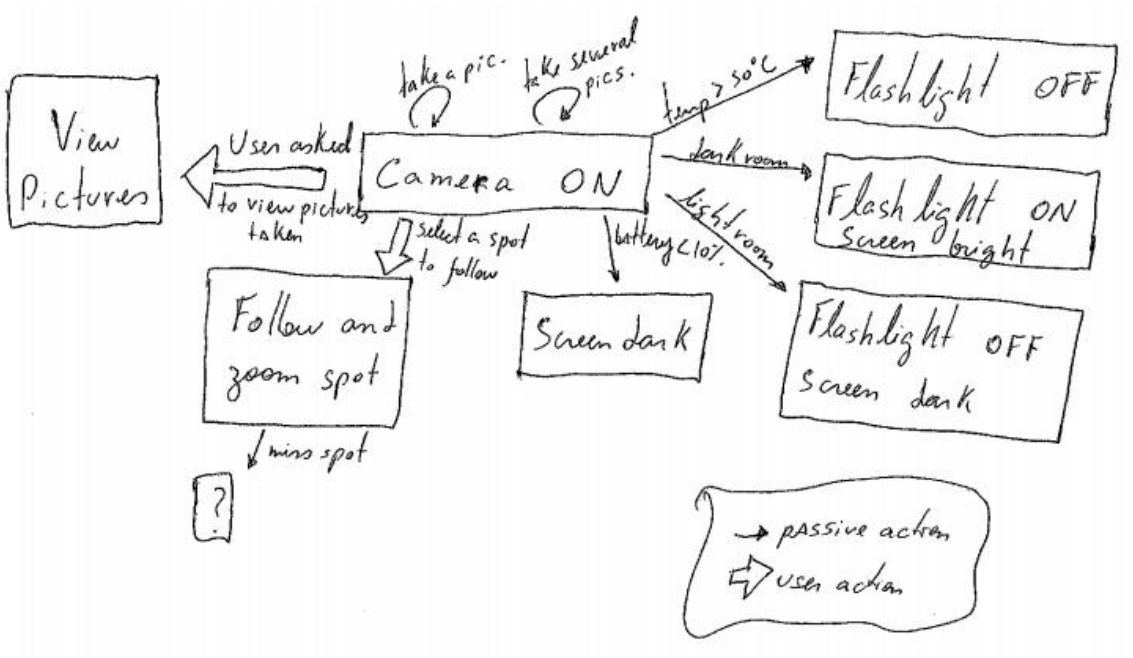

Figure 4.16 - Conceptual model for the third trial

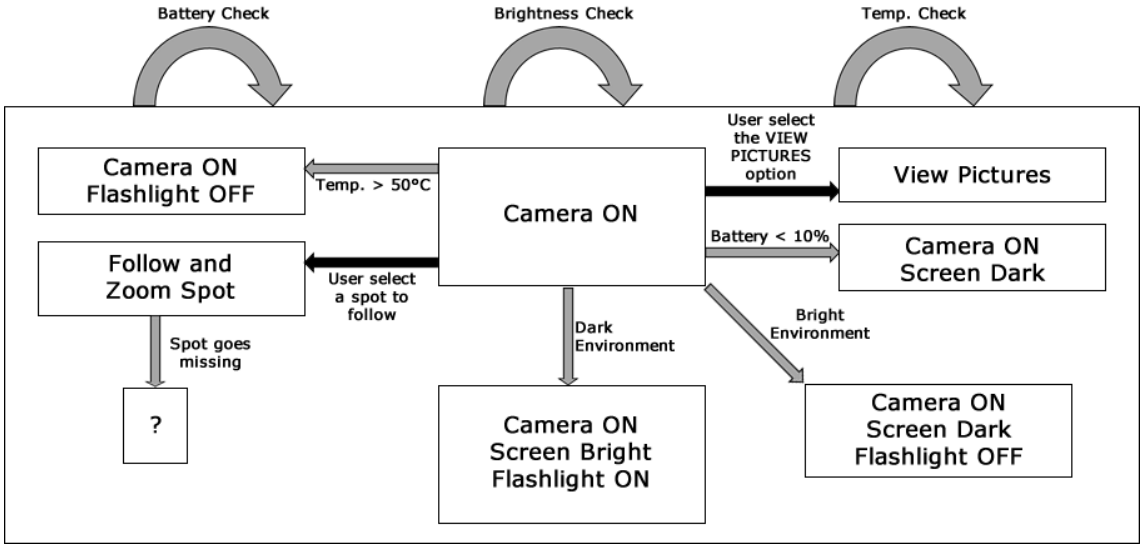

Figure 4.17 - Transcribed Conceptual model for the third trial

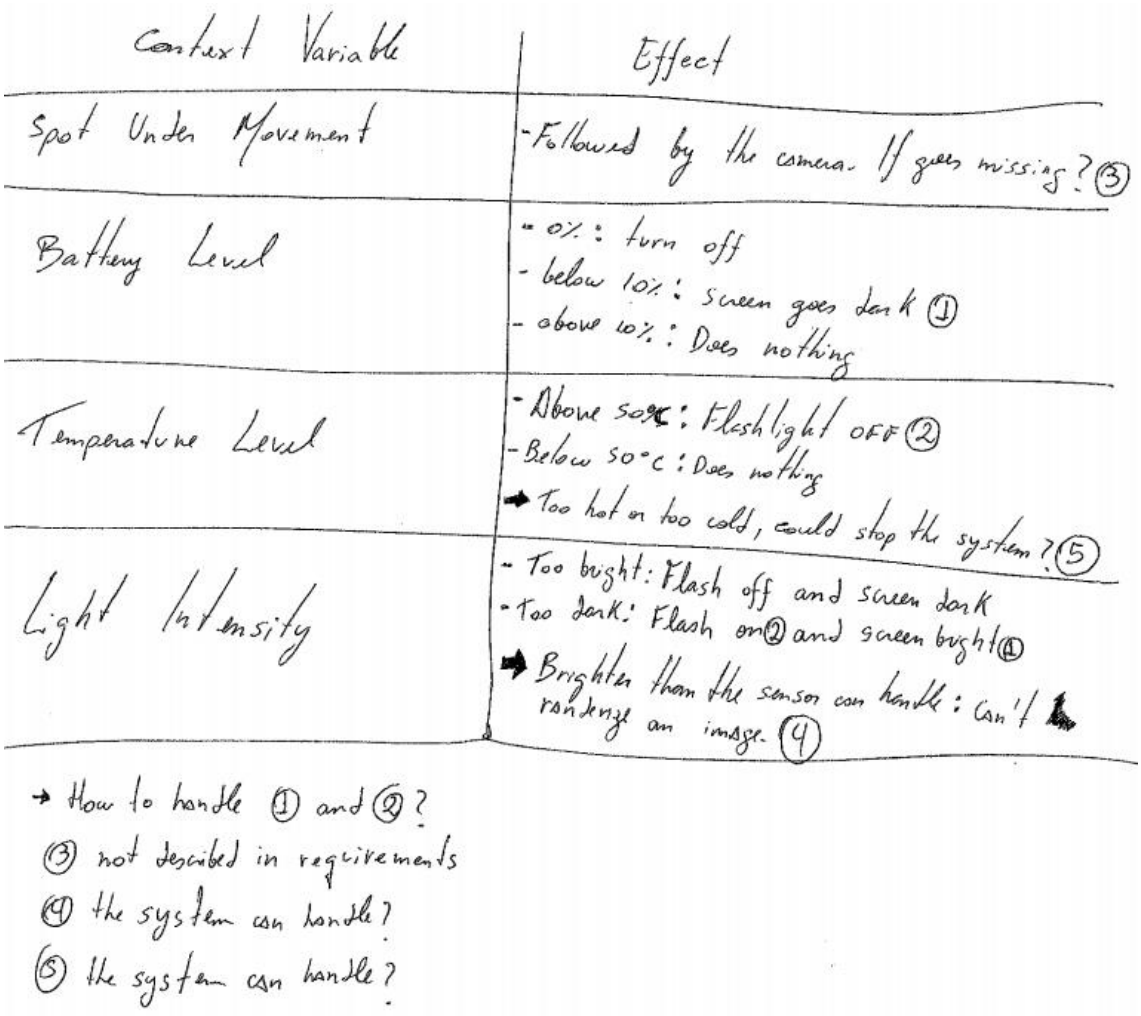

Figure 4.18 - Analytical model for the third trial

Context Variable	Effect
Spot Under Movement	Followed by the camera. If goes missing: ?
Battery Level	If 0%: Turn OFF If below 10%: Screen goes dark If above 10%: Does nothing
Temperature Level	If above 50 °C: Flashlight goes OFF If below 50 °C: Does nothing If too hot or too cold, could stop the system?
Light Intensity	If too bright: Flashlight goes OFF and Screen goes dark If too dark: Flashlight goes ON and Screen goes bright

Table 4.6 - Transcribed Analytical model for the third trial

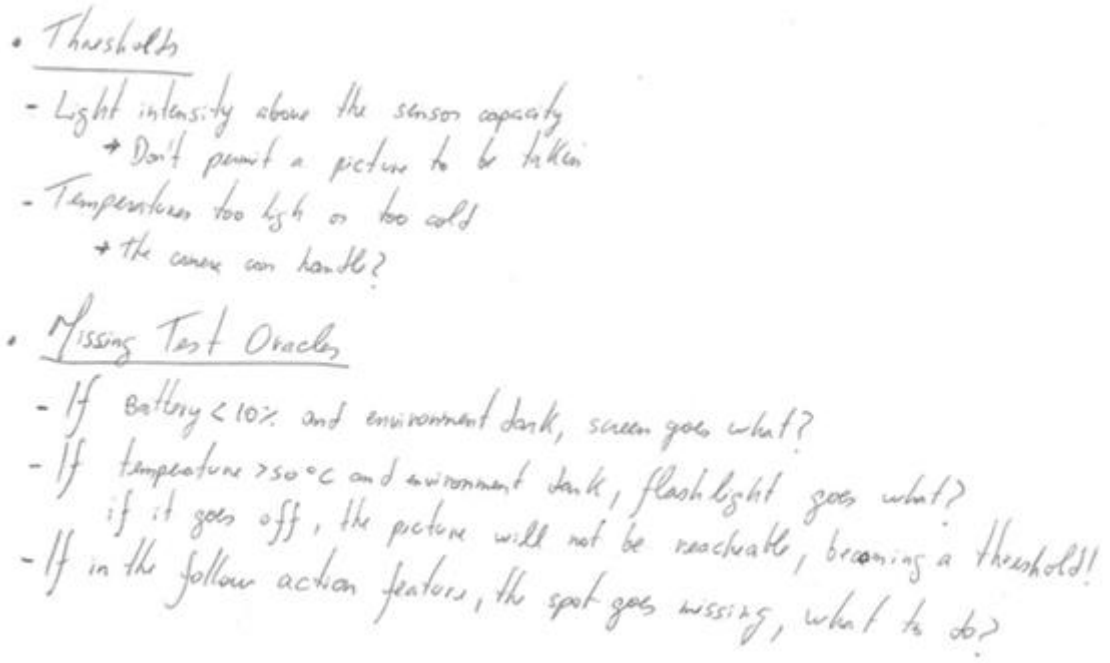

Figure 4.19 - Findings for the third trial used to generate the test suite

- **Thresholds**
    - Light Intensity above the sensor capacity
        - Do not permit a picture to be taken
    - Temperature too high or too cold
        - The camera can handle?
- **Missing Test Oracles**
    - If battery < 10% in a dark environment, what to do with the screen?
    - If temperature > 50 °C in a dark environment, what to do with the flashlight?
    - In the follow action feature, if the spot goes missing, what to do?

The last phase of the process in the third trial was to gather the findings and create a test suite. First, the test cases were created, then the test oracles. As mentioned before, the thresholds and context variables were not explicitly described. Table 4.7 shows an example of a test case in this version of the process.

Test Suite ID:	***CATS001***
Test Case ID:	***TakePicHL***
Use Case Base:	***Take a Picture***
Test Objective	*Verify the Take a Picture feature with environmental light high*
Precondition:	*The camera must be turned ON* *The environment must be bright*
Test Input:	*Press the picture icon*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The User press the picture icon.*
Test Inputs:	Press the picture icon
Test Expected Outputs:	*Take a picture in which it could be possible to recognize the objects in the picture* *The flashlight will not be activated* *The screen will be dark*
Post-Condition:	*The camera takes the picture and record it in the internal memory.*

Table 4.7 - Test case generated in the third trial

The problem with this test case template is that the concept of free context variation cannot be documented, since the steps characterize a controlled scenario to be followed. Moreover, the context variables and thresholds are not explicitly described, generating ambiguity in the interpretation.

4.4.4.3.1 Trial 3: Process Adaptation

The removal of the concept of thresholds of potential concern provided no changes in the process results and turned the process to be more objective. In this way, the context variables identification remained unchanged after this trial, as well as the conceptual and analytical models.

The test suite generation however suffered some adaptations. The template created to describe the test cases limits the contextual variances in each test case, which is not the objective of the process. The context in CASS needs to change freely and the test aspect must take this into account.

Therefore, a new test case template was created. In this template, a traditional test case is described with the test steps as it was in the previous template. Nevertheless, three fields were added to the template:

- **Relevant Context Variables:** List of context variables exerting influence in the actual test case.
- **Known Thresholds:** Identified thresholds that must be considered in the actual test case.
- **Test Expected Outputs for each Threshold:** System expected behavior when facing each threshold.

With these changes, the scenario (test steps) is maintained and all identified thresholds are described, as well as how the system is supposed to behave when facing each one of them. In this way, the tester has the power to choose when, between the test steps, a context change must happen. The post-condition field was merged with the Test Expected Outputs since for each possible context change, the Test Expected Outputs already show the post-conditions. Table 4.8 presents the SmartCamera example applied to the new proposed test case template.

Test Case ID:	***CATS001***
Test Objective	*Verify the Take a Picture feature*
Precondition:	*The camera must be turned ON*
Test Input:	*Press the picture icon*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The User press the picture icon.*
Relevant Context Variables:	1. *Battery Level* 2. *Temperature Level* 3. *Light Intensity*
Known Thresholds:	a. *Battery below 10%* b. *Temperature above 50°C* c. *Bright environment* d. *Dark environment*

	e. Battery below 10% and Dark environment *f. Temperature above 50°C and Dark environment*
Test Expected Outputs for each Threshold:	*a. Screen goes dark* *b. Flashlight goes off* *c. Screen goes dark and Flashlight goes off* *d. Screen goes bright and Flashlight goes on* *e. Not specified* *f. Not specified*

Table 4.8 - SmartCamera example in the CATS Design new test case template

Another important aspect to be observed in this new template is that thresholds that in the previous version had no oracle provided by the requirements are now described in the test case. Besides, now they present the expected output as **"not specified"**, indicating that there is a scenario that needs to be tested and the requirements do not provide information of how to handle it.

Observing this, the step of describing test oracles was put before the test cases description, so these missing spots can be identified earlier. The final version of the CATS Design process is presented in Figure 4.20 and described in details in the next section.

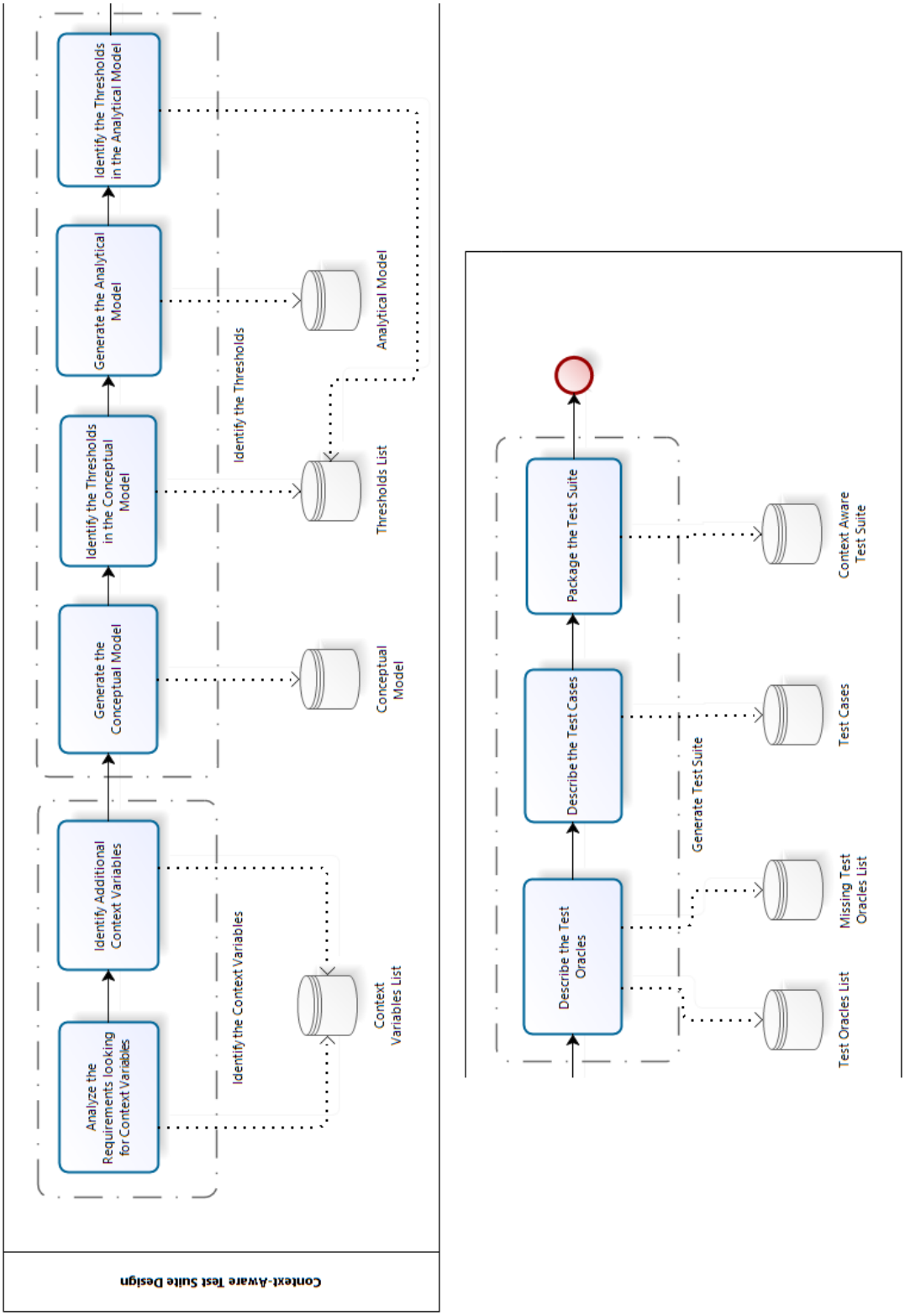

Figure 4.20 - CATS Design Process Final Version

4.4.5 Final Version

Based on the adaptations presented in the previous sections, the process for context-aware test suite design is described. Using the ISO/IEC 24774:2010 (ISO/IEC, 2010) process model together with the task template provided by the research group of software quality at COPPE/UFRJ, the process has been described as follows.

- **Purpose**

To define a context aware test suite considering the context variations in which the test item is immersed. This generated test suite aims to complement the coverage obtained by traditional test design techniques that were not designed to take into account the context variation.

- **Expected Results**
  - A List of Context Variables that may affect the behavior of the test item.
  - A List of Thresholds identifying the contexts where the Test Item may change its supposed behavior.
  - A Context-Aware Test Suite containing the test cases and test oracles.

- **Activities**

Activity: Identify the Context Variables

Description: In order to find how the context is influencing the Test Item, the context elements must be recognized beforehand. This task aims at discovering the elements composing the possible contexts in which the Test Item will be executed.

Task: Analyze the Requirements looking for Context Variables	
Description	The Tester reviews the Requirements documentation and enumerates the identified Context Variables – Variables that have their values updated even without a direct request from the user – that may exert influence over the Test Item, i.e. any variable that can change its value without a direct request of the user.
Pre-Task	-
Input Criteria	Requirements Documentation

Output Criteria	All the context variables present in the Requirements documentation must be present in the list of Context Variables.
Responsible	Tester
Participants	-
Required Artifacts	Requirements Documentation
Produced Artifacts	List of Context Variables
Support Tools	-
Post-Task	Identify Additional Context Variables

Task: Identify Additional Context Variables	
Description	The Tester together with additional domain experts identify additional context variables, apart from the requirements, by performing a deeper judgment into the list of Context Variables based on their own knowledge.
Pre-Task	Analyze the Requirements looking for Context Variables.
Input Criteria	The list of Context Variables
Output Criteria	Once the Tester and the additional experts involved in the testing process are convinced that all possible context variables that may influence the software are present in the list of Context Variables.
Responsible	Tester
Participants	Experts in the Software Domain (Optional)
Required Artifacts	List of Context Variables
Produced Artifacts	List of Context Variables Updated
Support Tools	-
Post-Task	Generate the Conceptual Model

Activity: Identify the Thresholds

Description: Once the contextual information influencing the Test Item is known, this activity will generate models that show how each context variable impacts the system.

Task: Generate the Conceptual Model	
Description	The Tester must write down every instance/state the Test Item can have based on the combination of context variables, based on the Requirements and the list of Context Variables. Each state/instance of the system should be indicated as a rectangle, passive actions (without actor's intervention) should be indicated with grey arrows and active actions (with actor's intervention) with black arrows.
Pre-Task	Identify Additional Context Variables
Input Criteria	The list of Context Variables and the Requirements Documentation.
Output Criteria	All the possible usage situations of the Test Item affected by a context variable (or combination of them) must be listed.
Responsible	Tester
Participants	-
Required Artifacts	Requirements Documentation and list of Context Variables
Produced Artifacts	Conceptual Model.
Support Tools	Any graphical editor or paper and pen.
Post-Task	Identify the Thresholds in the Conceptual Model.

Task: Identify the Thresholds in the Conceptual Model	
Description	The relations identified in the Conceptual Model are used to find possible Thresholds for the Test Item. The Tester must check if any of the described transitions might suffer the influence of the context variables described in the list of Context Variables. If any transitions were supposed to not happen normally based on one or more context variables change, this need to be described in the list of

	Thresholds. This description must include the affected transition, what context variables cause this transition to be affected and which values these variables as supposed to have in order to cause that effect.
Pre-Task	Generate the Conceptual Model
Input Criteria	The Conceptual Model, the list of Context Variables and the Requirements Documentation, so the Tester can check the relation between the context variables and the transitions, and also their possible values.
Output Criteria	All the possible usage situations of the Test Item must be checked, including the transitions between them.
Responsible	Tester
Participants	-
Required Artifacts	Requirements Documentation, Context Variables List and Conceptual Model.
Produced Artifacts	List of Thresholds
Support Tools	-
Post-Task	Generate the Analytical Model

Task: Generate the Analytical Model	
Description	This task describes how each context variable is supposed to interact with the Test Item. The Tester must right down a list with every context variable present in the list of Context Variables and describe how each of these variables impact the Test Item behavior.
Pre-Task	Identify the Thresholds in the Conceptual Model.
Input Criteria	The list of Context Variables and the Requirements Documentation.
Output Criteria	All the context variables expected influence over the Test Item must be described.
Responsible	Tester

Participants	-
Required Artifacts	Requirements Documentation and list of Context Variables.
Produced Artifacts	Analytical Model.
Support Tools	Any text editor or paper and pen.
Post-Task	Identify the Thresholds in the Analytical Model.

Task: Identify the Thresholds in the Analytical Model	
Description	This task will use the descriptions of the Analytical Model to identify possible Thresholds of the Test Item. Analyzing the Analytical Model content, any context variable (or combination of them) that is supposed to make the Test Item to behave in a way that it is not supposed to be, must be described in the list of Thresholds. This description must include the feature affected, what context variables cause this feature to be affected and which values these variables as supposed to have in order to cause that effect.
Pre-Task	Generate the Analytical Model
Input Criteria	The Analytical, the list of Context Variables and the Requirements must be provided, so the Tester can check the relation between the context variables and the system behavior, and also their possible values.
Output Criteria	All the possible usage influences of the context variables over the Test Item must be checked.
Responsible	Tester
Participants	-
Required Artifacts	Requirements, list of Context Variables and Analytical Model.
Produced Artifacts	List of Thresholds
Support Tools	-

Post-Task	

Activity: Generate the Test Suite

Description: After discovering the context factors that may affect the system and how they do it, this activity proposes how a context aware test suite is generated.

Task: Describe the Test Oracles	
Description	This task uses the generated products of CATS Design to create the Test Oracles considering the context variation. For each instance and transition of the Test Item described in the Conceptual Model that might be affected by the identified Thresholds, a Test Oracle must be described based on the Requirements. If some instance or transition does not have a clear Test Oracle stated in the Requirements, it must be documented as a *Missing Test Oracle* and Requirements must be updated. Otherwise, the test oracle must be described as NOT SPECIFIED in the Test Case.
Pre-Task	Identify the Thresholds in the Analytical Model
Input Criteria	The context variables and thresholds must be known at this point.
Output Criteria	All the possible usage situations of the Test Item must have a Test Oracle.
Responsible	Tester
Participants	-
Required Artifacts	Requirements Documentation, list of Context Variables, Conceptual Model and list of Thresholds.
Produced Artifacts	List of Test Oracles and list of Missing Test Oracles
Support Tools	Any text editor or paper and pen
Post-Task	Describe the Test Cases

Task: Describe the Test Cases	
Description	This task describes explicitly the instances and transitions presented in the Conceptual Model as Test Cases. The Test Case template must include the Test Oracles developed in the process, the identified Thresholds that may affect each of the Test Cases and also all the Context Variables that may influence each Test Case, even the ones that do not necessarily generate a Threshold.
Pre-Task	Describe the Test Oracles
Input Criteria	The context variables, thresholds and test oracles must be known at this point.
Output Criteria	All the possible usage situations of the Test Item must have a Test Case.
Responsible	Tester
Participants	-
Required Artifacts	Requirements Documentation, list of Context Variables, Conceptual Model, list of Thresholds and list of Test Oracles.
Produced Artifacts	Test Cases
Support Tools	Any text editor or paper and pen.
Post-Task	Package the Test Suite

Task: Package the Test Suite	
Description	This task puts together the generated products of the process.
Pre-Task	Describe the Test Cases
Input Criteria	All the Test Cases must be described, including their Test Oracles, Thresholds and Context Variables.
Output Criteria	The Test Suite.
Responsible	Tester

Participants	-
Required Artifacts	Test Cases
Produced Artifacts	Context Aware Test Suite
Support Tools	-
Post-Task	-

4.5 Chapter Conclusions

This chapter presented the evolution process of the CATS Design, a context-aware test suite design approach. The development of this process started with an *ad-hoc* web search conducted using the keywords presented in chapter 3 qSLR, looking for solutions from other domains for issues similar to the ones existing in the CASS domain.

During this search, a domain called Cybernetics was encountered. The Cybernetics domain was concerned with disturbances that could make the system not work as expected. To handle this issue, a controller needed to take decisions of what to do every time a disturbance appeared.

Inside the Cybernetics domain, the concept of resilience was identified. Based on it, the domain of Organizational Resilience was introduced with the same issues of the Cybernetics domain. However, the way of handling this issue by the Organizational Resilience domain was to discover the possible disturbances that could affect the system (called thresholds in this domain) and instrument the system how to avoid them.

Although the Organizational Resilience presented an approach to identify and avoid the thresholds, the idea for CASS testing was just to identify them, so the same approach could be adapted for the CASS reality. Therefore, the process of thresholds identification by the Organizational Resilience domain was adapted to the CASS domain and introduced in the ISO/IEC/IEEE 29119:2013 test process presented in chapter 2.

In addition, the methodology used for this adaptation was presented and two CASS examples were chosen to support this adaptation. Finally, after three trials of adaptations, a stable version of the process was presented as the CATS Design approach – Context-Aware Test Suite Design.

The final version of CATS Design process can generate a complete set of test cases, test oracles, thresholds and context variables based on the requirements specifications and also supported by the tester domain knowledge. The next chapter presents the stable version of the CATS Design process being used to support the testing planning to a real system as a proof of concept, as well as the threats to validity observed during its application.

5 Context-Aware Test Process Evaluation

This chapter presents the proof of concept of the final version of CATS Design process as described in chapter 4. In addition, a more detailed discussion about the evaluation results and threats to validity are presented.

5.1 Introduction

This chapter presents the evaluation of CATS Design, the CASS testing approach proposed in this dissertation. The objective is to analyze the CATS Design process to observe its coverage regarding the testing of CASS. For this proof of concept, a third party software project (developed as an undergrad project at the ORT University in Uruguay) has been selected by convenience.

The chosen project represents a ubiquitous system called CAUS – Context-Aware University System – proposed by Castellanos (2015), which also provides the requirements specification and a non-context-aware test suite, which helps the evaluation of CATS Design.

The system uses QR Code and other sensors to supply the user with information about the university environment, offices and other users. Besides the details of the CAUS project and the CATS Design proof of concept, this chapter provides a detailed discussion about the results and the conclusions generated by them, the threats to validity observed during the CATS Design usage and other relevant considerations.

5.2 CAUS: Context-Aware University System

The CAUS system was developed as a bachelor's undergrad project and aimed at applying the ubiquitous computing concepts into a university environment. By adapting the work of Abowd et al. (1996), which had already proposed a context-aware university system, yet with no formal requirements specification. Based on this, Castellanos (2015) defined the CAUS system, which is a mobile application that recovers information from QR Codes, Wi-Fi signal and other sensors in order to provide contextual information to the user.

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

The system's communication approach uses the mobile device camera for QR code reading. The application is involved with several actors, different from the Smart Camera (Chapter 4, Section 4.4.1.2), which was involved with just one. In addition, the system is supposed to be functional only inside the university dependencies, characterizing even more the concept of context-awareness.

The features incorporated into the system include the localization of the user and offices, management of user and offices, request for users and offices availability and messages exchanging. The requirements specification is available online (Castellanos, 2015).

5.3 Proof of Concept

The CATS Design process presented in chapter 4 can be quickly summarized as following:

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

This proof of concept using the CAUS project was conducted in May 14th of 2015. Using the requirements specification (Castellanos, 2015) as the process' input, the CATS Design process was executed. During the phase of *Context Variables Identification*, two context variables were recovered directly from the requirements specification (Step 1). Four others were inferred in the step of additional variables identification (Step 2). The explanation for the selection of each of the presented context variables is presented below.

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

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

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

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

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

Not only were the use cases used to generate the instances of the conceptual model, but also the non-functional requirements. Together with the context variables identified in the first step phase of the CATS Design process, the relations connecting the different states of the CAUS project were created. The conceptual model is presented in Figure 5.1 and transcribed in Figure 5.2.

2) Thresholds Identification

- ***Step 3: Create a conceptual model***

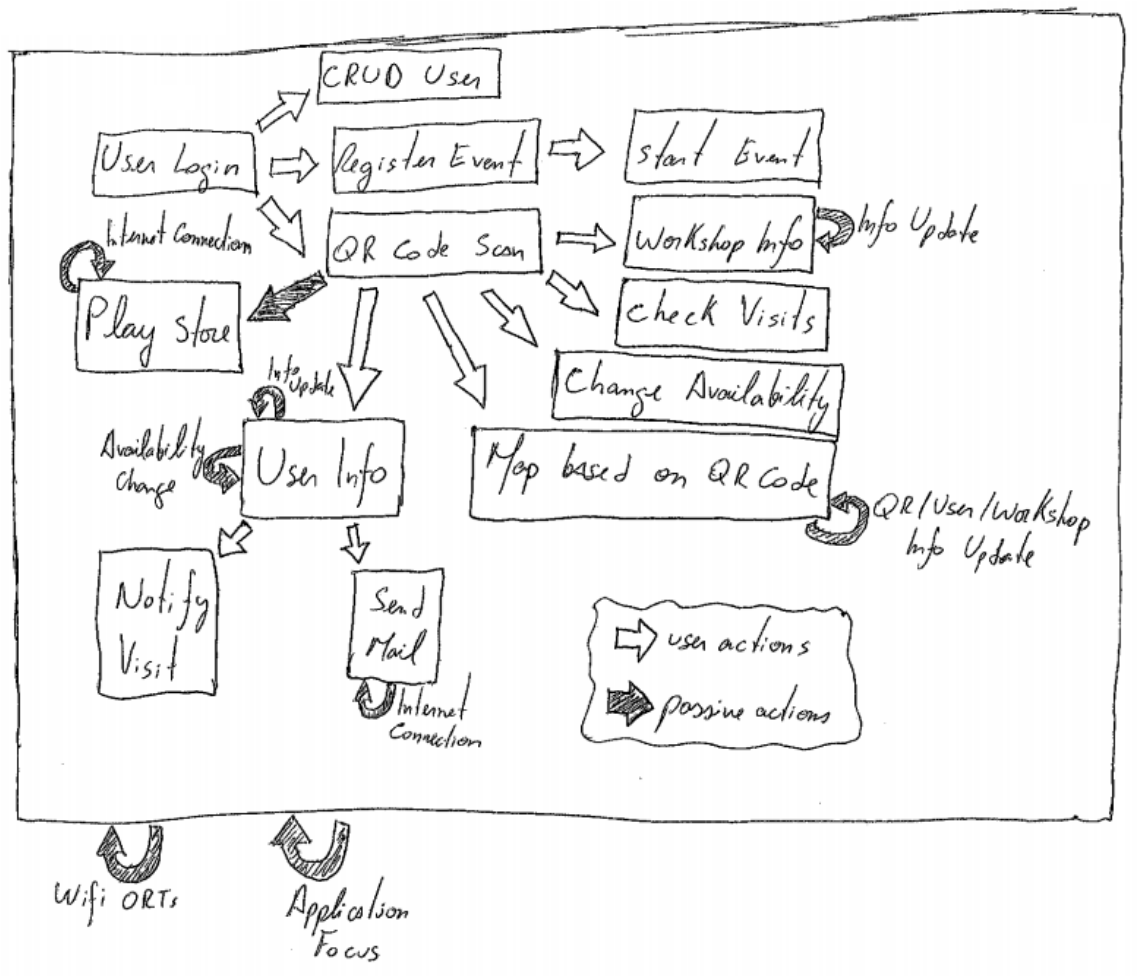

Figure 5.1 - Conceptual Model for the CAUS Project

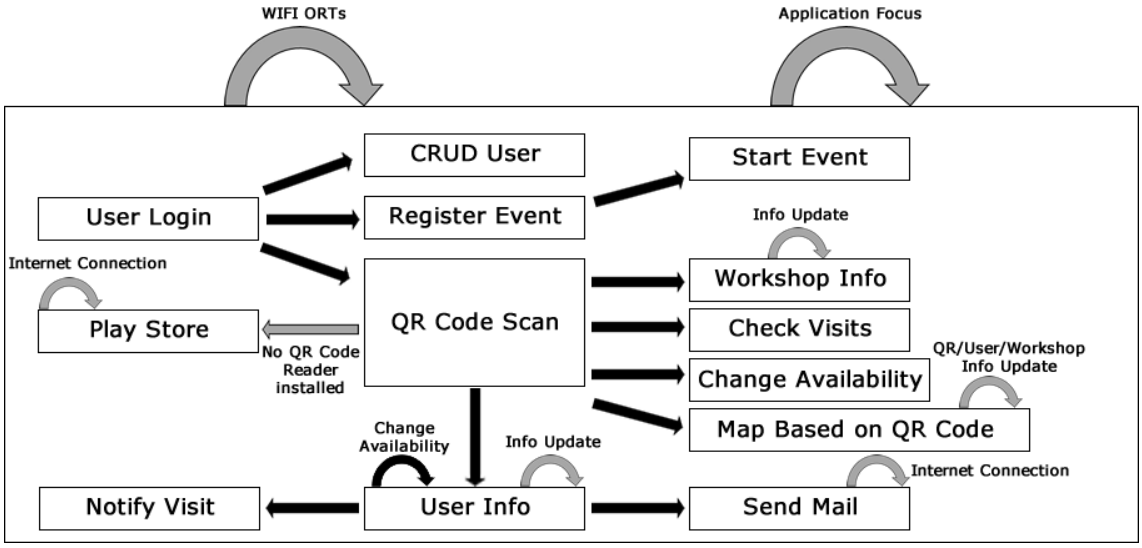

Figure 5.2 - Transcribed Conceptual Model for the CAUS Project

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

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

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

Since the **information update** is not an important context variance while the user is, for instance, sending an email, the identified thresholds were also described with the information of when they are important to be handled. Continuing with the thresholds identification phase, steps 5 and 6 are responsible for the creation and interpretation of the analytical model. These steps allowed to find three additional thresholds to be incorporated to the test suite. The analytical model is shown in Figure 5.3 and transcribed in Table 5.1. The thresholds are presented below as well.

- ***Step 5: Create analytical model***

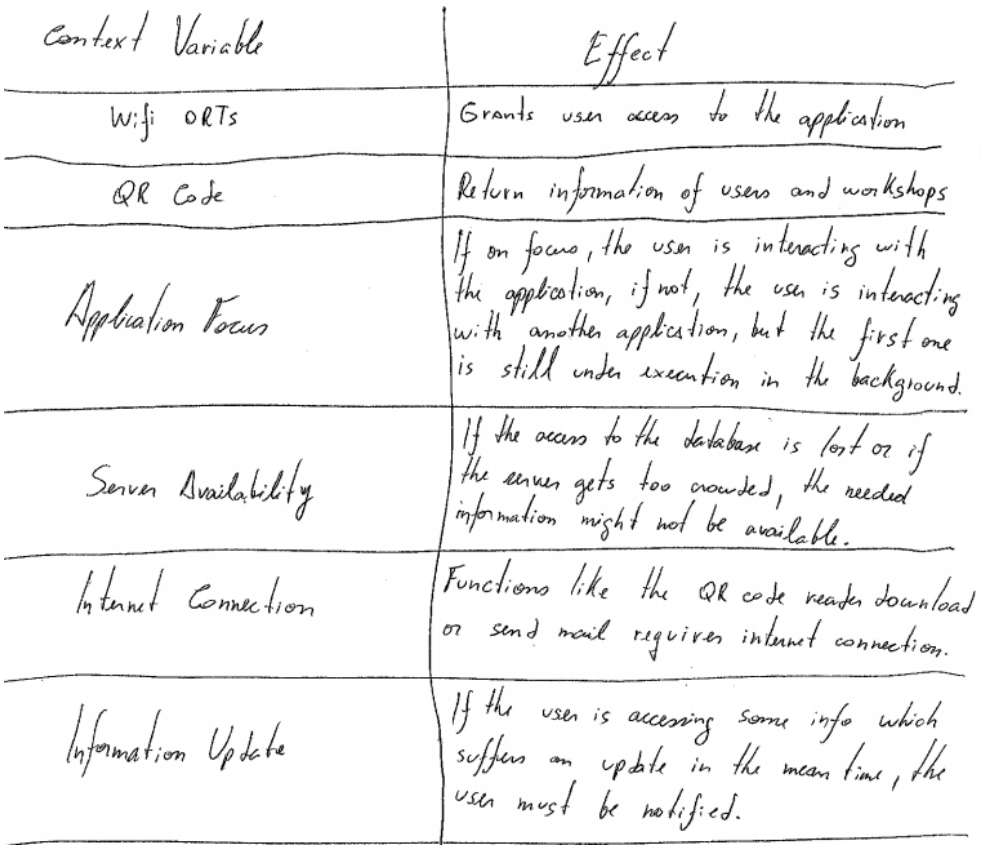

Figure 5.3 - Analytical Model for the CAUS Project

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

Table 5.1 - Transcribed Analytical Model for the CAUS Project

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

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

Starting the third and last phase of the CATS Design process, the context variables and thresholds are already known. In addition, the conceptual model, the analytical model and the requirements specification are available artifacts.

Analyzing the list of thresholds, the seventh step describes how the system is supposed to behave when each one of the thresholds is faced, i.e. the test oracles concerning the thresholds. If necessary, the lack of expected behavior is documented. This step is presented in Table 5.2.

3) Test Suite Generation

- ***Step 7: List test oracles***

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

Any transition or state	Server goes offline	Not specified
Any transition or state	Server limit is exceeded	Not specified

Table 5.2 - Test oracles description for CAUS Project

As discussed before, the threshold concept only makes sense when considering the feature in which the threshold is identified. There is no reason for considering the **loss of internet connection** during the **QR Code Scan** since the user and the server are connected by the same Wi-Fi and the information is available even without the internet. However, when considering features like **send mail**, then the **loss of internet connection** must be taken into consideration. The missing test oracles are described as "Not specified" and are documented as well.

- ***Step 8: Create test cases***

Once the test oracles are known, step 8 is intended to generate the test cases aiming at covering the identified contexts that are relevant to the system. Those contexts are the **transitions considered in the conceptual model and the thresholds presented in the thresholds list**.

Each test case must describe the relevant context variables, the relevant thresholds and the test oracles for the listed thresholds. An example of generated test case is presented in Table 5.3. The complete list of test cases for the CAUS proof of concept can be found in the APPENDIX C – CAUS Context-Aware Test Suite.

Test Case ID:	***CATS - TC001***
Test Objective	*Verify the login feature*
Precondition:	-
Test Input:	*A valid user and password*
Test steps:	1. *User connect to University Wi-Fi* 2. *User executes the application* 3. *User provides credentials to access the application* 4. *The system shows the menu for the user*
Relevant Context Variables:	1. *University Wi-Fi* 2. *Application Focus* 3. *Server Availability*
Known Thresholds:	a. *Loss of University Wi-Fi connection* b. *Loss of focus*

	c. *Server goes offline* d. *Server limit is exceeded*
Test Expected Outputs for each Threshold:	a. *Notify the user and close the application* b. *Continue the application from where it stopped* c. *Not specified* d. *Not specified*

Table 5.3 - Test case example for the CAUS Project

Notice that no alternative flow has been described, for instance providing invalid credentials for the login feature. This kind of coverage can be obtained by other means of software testing, the focus of the CATS Design is the context variation, and this is why it is intended to be complementary to other testing techniques.

- ***Step 9: Package test suite***

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

5.4 Discussion

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

The second phase, responsible for generating the models, has also indicated to be useful for thresholds identification. First, during the CATS Design process construction, the same example (SmartCamera) had two distinct interactions, generating distinct models, and still the conclusions generated by the models were the same. In this proof of concept, the models once more revealed to be complementary, when the conceptual and analytical models separately could not list the impact of all context variables found in phase one. However, using them together was possible to identify how each of them impact the system.

By the third and final phase, the process of creating test oracles based on the context variables, thresholds and requirements have revealed to be useful and helped

finding faults from the requirements as well, indicating that the CATS Design approach not only deals with the testing aspect, but also with some verification of the requirements documentation.

Although the aim of the test process is not to point omissions on the requirements specification, it is interesting to observe the verification ability of the CATS Design regarding the completeness of the requirements documentation for CASS.

Finally, the test case template final version is able to address scenarios without freezing the context, i.e. without treating the context variables as inputs. Instead, all possible relevant contextual changes are described together with the test scenario, and the Tester have the freedom to model the test environment as s/he wishes and change the context variables values at any time, knowing how the system is supposed to behave at any relevant configuration identified by the CATS Design process.

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

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

5.5 Threats to Validity

It is important to observe that even though the CATS Design process provides a simple and useful improvement for CASS testing, due the lack of real projects, the presented results have some threats to validity to be considered.

The trials of adaptation were executed using incomplete requirements or requirements created by the own author, which can bias the observations. The final version of the process was evaluated with an undergrad final project chosen by convenience, not with industrial projects, which can be seen as a conclusion threat to validity as well.

The generated models of the CATS Design process can generate multiple interpretations according to the process applier. However, this threat was mitigated twice. Once when the conceptual and analytical model revealed to be complementary and the second time when the same project (SmartCamera) generated two distinct models, but both of them provided the same conclusions regarding thresholds.

The adaptations made from the Organizational Resilience domain seemed to support the process of finding thresholds, but there is no evidence that this is the best option for the context-aware domain. Moreover, the generated test suite was not formally evaluated with an experiment or even due time restrictions, which can be seen as a threat.

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

5.6 Chapter Conclusions

This chapter presented the methodology for evaluation of the CATS Design process. A proof of concept was presented, as well as the project chosen to be part of the process with a deeper discussion about the findings and a list of threats to validity.

Although the process was not empirically experimented, the proof of concept provides information that increases the belief in the applicability of the CATS Design approach in real CASS. In addition, the generated models also provide a better understanding of the impact of contextual variance during the system execution.

Finally, during the step of test oracles identification based on the thresholds, it was noticed that the CATS Design approach can be used not only for testing, but for verification of the completeness of the requirements documentation regarding context-awareness. This chapter was important to understand the applicability of the process and its limitations. The next chapter presents the conclusions and plans of future work.

6 Conclusions and Future Work

This chapter presents the conclusions and contributions generated by this research. Moreover, the limitations of the results achieved are provided and a few suggestions of future work are proposed.

6.1 Introduction

This research presented the issues on addressing the testing of context-aware software systems. It was stated that traditional testing approaches could reduce the system efficiency when applied to ubiquitous systems (Ducatel et al., 2003). Since CASS is a type of ubiquitous systems, applying traditional testing techniques to them would reduce their efficiency as well.

Therefore, a *quasi*-Systematic Literature Review was conducted to find evidence about how to test this type of systems. Nevertheless, the found techniques did not have a predictable coverage or could not handle the context-aware feature properly, specifically the aspect of free context variance.

Based on this, a research was performed aiming to find similar issues in other domains. The Cybernetics and Organizational Resilience domains were then identified and studied. A process for handling context-aware issues was then adapted from the Organizational Resilience domain and tailored to the context-aware domain, becoming the CATS Design process. CATS Design was then evolved using CASS toy projects until its final version. Finally a proof of concept was conducted in order to evaluate the process final shape.

This chapter presents the contributions generated by this dissertation, a more detailed discussion about the limitations encountered in each of the generated contributions and how this work can be improved in the future.

6.2 Contributions

Firstly, together with the state of the art presented in chapter 2, a discussion about the existing challenges in the context-aware area for the industry was also presented. The proposed challenges increase the idea of why the industry must be concerned about solutions for this field, since context-aware applications keep getting developed with no specific technique to test them. Some of these challenges were published in the SAST Workshop during the CBSoft Conference in 2015 (Matalonga et al., 2015a).

In chapter 3, a *quasi*-systematic literature review research protocol was prepared to identify the state of the art regarding testing techniques for context-aware systems.

This protocol is available online as a technical report (Rodrigues et al., 2014). This technical report contributed to the body of knowledge regarding software testing, ubiquitous systems and context-aware software systems.

The qSLR also generated another result, which was the verification of the ISO/IEC/IEEE 29119:2013 test design techniques against the findings of the qSLR. During the research, was possible to observe that rarely the authors consistently classify the used test design techniques. Moreover, when they do, not always they match the international standard classifications. Still the ISO/IEC/IEE 29119:2013 standard was able to classify almost all of the identified approaches, indicating that testing context-aware systems is no different from testing traditional systems or that the existing approaches still do not explore correctly the context-aware feature. These results were discussed in chapter 3 and presented in the SPICE Conference in 2015 (Matalonga et al., 2015b).

The chapter 4 presented the main contribution of this dissertation, a context-aware test suite design process, the CATS Design. A testing process supporting the test of the context-aware feature, not yet addressed in the technical literature recovered from the *quasi*-systematic literature review presented in chapter 3. The proposed process allows the test cases to keep the context varying freely, different from the recovered approaches of the qSLR.

In addition, this research provides an initial evaluation, as a proof of concept, of the proposed context-aware test suite design, presented in chapter 5. This evaluation can be a starting point to motivate future applications and researches of the CATS Design approach.

6.3 Limitations

From the contributions provided by this research, the results of the *quasi*-systematic literature review were the basis for all of them, since it provided the state of the art in the field investigated in this dissertation. Considering it, all the threats to validity presented in the *quasi*-systematic literature review protocol (Rodrigues et al., 2014) need to be considered as limitations for this research as well.

A complete list of these limitations was presented in chapter 3, for instance the **inaccuracy of data extraction** resulting in misunderstanding of the results, the **bias on synthesis information** since some of the chosen taxonomies, as the ISO/IEC/IEEE 29119, are used to classify studies according to the reader interpretation and the **construct validity of the studies quality**, since the selected studies might not be comparable.

Moreover, in respect to the CATS Design process, resilience domain might not be the adequate domain to gather a process to be adapted for the context of context-aware software testing. Even though, the results achieved indicate an improvement in the test coverage concerning context-awareness.

It is also important to consider that all the evaluations made with the CATS Design process were represented as a proof of concepts. None of them used a real industrial project, which limits the capacity of generalization of the results.

6.4 Answers to the Research Questions

As presented in Chapter 1, this dissertation's research questions are:

- Is it possible to design a test approach considering context variance?
- If yes, does it improve the test coverage when compared with the traditional testing techniques?

Taking the results of the *q*SLR presented in Chapter 3, all of the identified testing approaches rely on constraining the context variables values during the test execution. Based on this, a search was conducted by looking for similar problems into other domains. An adaptation of a solution from the Organizational Resilience domain was proposed in Chapter 4 in order to provide an approach for testing without constraining context variance.

The results presented in Chapter 5 indicate that the proposed approach is able to support the identification of test cases considering the context variance. In addition, the proposed approach aids the verification and validation of the requirements documentation completeness regarding the context-aware feature.

Based on this result, we can assert that it is possible to design a test approach considering the context variance. In respect to coverage, the proposed technique coverage is based on the contexts revealed during the models analysis, i.e. the coverage is concerned exclusively with the context-aware feature.

Therefore, our hypothesis is that the proposed approach can provide a good coverage regarding the context-aware features; however it does not offer a good coverage when considering other features like the amount of available use cases. It can indicate that the CATS Design approach is a complementary technique, which allows to deal with the context variance together with traditional ones. It provides a good context-aware coverage for testing that can contribute to increase the test coverage of context-aware systems being already tested with other testing techniques.

6.5 Future Work

Considering the quick evolution of the technology, it is important to update the results of the *quasi*-systematic literature review in the nearest future. Not only to gather updated information, but also to fulfill possible gaps, such as the absence of testing techniques that allow the context to vary freely.

It would be interesting to apply the CATS Design process into a real industry project by means of a case study or a controlled experiment. This would give the process more reliability and would help to improve it.

Finally, a proposal for executing the test cases generated by the CATS Design process could be developed and incorporated to the process. Although the test suite generated provides enough information for testing the context-aware feature, the way of testing it might influence the test results.

For instance, despite deciding to verify the thresholds just in the first step of each test case being a possible usage situation of the CATS Design, the coverage would not be as good as when applying the threshold verification at each step.

6.6 Chapter Conclusions

This last chapter presented the importance of contributing for the body of knowledge on software testing by proposing a solution to deal with an open problem in software engineering: the testing of CASS. Moreover, the limitations encountered in each of the steps of the dissertation and possibilities of future work were discussed.

The material produced as technical papers and reports was also briefly presented, showing the distinct paths of rationale that this research was able to trace in order to achieve its actual state. It was also stated the contribution of the first steps of the dissertation for a wider project (CAcTUS Project) that intends to provide further solutions for the context-aware systems domain.

In respect of CASS testing, the proposed approach can still evolve and its use in a real CASS project would be of great importance for its further maturity. Assuring a way to measure properly the coverage attained by the CATS Design is also an important path to consider, probably using the measurement Context Diversity proposed by Wang et al. (2010) or other similar one, since it was still not evaluated.

The main point to observe in this dissertation, aside the CATS Design proposition, is that the field of testing CASS is still young and presents other challenges of research. As stated by Ducatel (2003), ubiquitous systems might lose efficiency and efficacy whether dealt with traditional software technologies.

References

Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R. & Pinkerton, M. (1996). Cyberguide : A Mobile Context-Aware Tour Guide. *Baltzer Journals 3*: 1–21.

Alsos, O.A. & Dahl, Y. (2008). Toward a best practice for laboratory-based usability evaluations of mobile ICT for hospitals. *Proceedings of the 5th Nordic conference on Human-computer interaction building bridges - NordiCHI '08* 3.

Amalfitano, D., Fasolino, A.R., Tramontana, P. & Amatucci, N. (2013). Considering Context Events in Event-Based Testing of Mobile Applications. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*.

Ashby, W.R. (1956). An introduction to cybernetics. *An introduction to cybernetics.*

Basili, V.R., Caldiera, G. & Rombach, H.D. (1994). The goal question metric approach. *Encyclopedia of Software Engineering 2*: 528–532. Retrieved from http://maisqual.squoring.com/wiki/index.php/The Goal Question Metric Approach

Beer, S. (1981). *Brain of the Firm: The Managerial Cybernetics of Organization*. John Wiley & Sons. Retrieved from https://books.google.ca/books?id=bVK3AAAAIAAJ

Biolchini, J., Mian, P.G., Candida, A. & Natali, C. (2005). Systematic Review in Software Engineering. *Engineering 679*: 165–176. Retrieved from http://www.cin.ufpe.br/~in1037/leitura/systematicReviewSE-COPPE.pdf

Buchinger, D., Cavalcanti, G.A. de S. & Hounsell, M.D.S. (2014). Mecanismos de busca acadêmica: uma análise quantitativa. *Revista Brasileira de Computação Aplicada 6*(1).

Camuffo, M., Maiocchi, M. & Morselli, M. (1990). Automatic software test generation. *Information and Software Technology 32*(5): 337–346. Retrieved from http://www.sciencedirect.com/science/article/pii/095058499090003A

Canfora, G., Mercaldo, F., Visaggio, C.A., D'Angelo, M., Furno, A. & Manganelli, C. (2013). A case study of automating user experience-oriented performance testing on smartphones. *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013* 66–69.

Castellanos, S. (2015). Proyecto CAUS - Context Aware University System - Especificación de Requisitos. Retrieved 1 May 2015, from http://fi.ort.edu.uy/innovaportal/file/2231/9/esre_prototipo_caus.summary.pdf

Delamaro, M.E., Maldonado, J.C. & Jino, M. (2007). *Introdução ao teste de software*. Rio de Janeiro: CAMPUS.

Dey, A.K. (2001). Understanding and using context. *Personal and Ubiquitous Computing 5*: 4–7.

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

Dijkstra, E.W. (1972). The Humble Programmer. *Commun. ACM 15*(10): 859–866. Retrieved from http://doi.acm.org/10.1145/355604.361591

Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J. & Burgelman, J.-C. (2003). Ambient intelligence: From vision to reality. *IST Advisory Group Draft Report, European Commission*.

Folke, C., Carpenter, S.R., Walker, B., Scheffer, M., Chapin, T. & Rockström, J. (2010). Resilience thinking: integrating resilience, adaptability and transformability. *Ecology and Society 15*(4): 20.

Forman, R.T.T. & Godron, M. (1986). *Landscape Ecology*. Wiley. Retrieved from https://books.google.com.br/books?id=ZvNEVs2MWqcC

Glass, R.L. & Hunt, A. (2006). *Software Conflict 2.0: The Art And Science of Software Engineering*. Developer.* Books.

Goodenough, J.B. & Gerhart, S.L. (1975). Toward a Theory of Test Data Selection. In *Proceedings of the International Conference on Reliable Software*. New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/800027.808473

Goodenough, J.B. & Gerhart, S.L. (1977). Toward a Theory of Testing: Data Selection Criteria. In Prentice-Hall (ed.), *Current trends in programming methodology* (Vol. vol. 2 R.). Englewood Cliffs, NJ.

Harries-Jones, P. (1988). The Self-Organizing Polity: An Epistemological Analysis of Political Life Laurent Dobuzinskis Boulder: Westview Press, 1987, pp. 223. *Canadian Journal of Political Science 21*(02): 431–433.

Holling, C.S. (1973). Resilience and stability of ecological systems. *Annual review of ecology and systematics* 1–23.

Jiang, B., Long, X. & Gao, X. (2007). MobileTest: A tool supporting automatic black box test for software on smart mobile devices BT - 29th International Conference on Software Engineering, ICSE'07 - 2nd International Workshop on Automation of Software Test, AST'07, May 20, 2007 - May 26, 20 IEEE Computer Society Technical Council on Softwar.

Leeds, H.D. & Weinberg, G.M. (1961). *Computer programming fundamentals*. McGraw-Hill New York.

Malik, N., Mahmud, U. & Javed, Y. (2007). Future challenges in context-aware computing. *Proceedings of the IADIS International Conference* 306–310.

Matalonga, S., Rodrigues, F. & Travassos, G. (2015a). Challenges in Testing Context Aware Software Systems. In *9th Workshop on Systematic and Automated Software Testing 2015*. Belo Horizonte, Brazil.

Matalonga, S., Rodrigues, F. & Travassos, G. (2015b). Matching Context Aware Software Testing Design Techniques to ISO/IEC/IEEE 29119. In *15th International Conference SPICE Conference*. Gothenburg, Sweden.

Mathur, A.P. (2008). *Foundations of Software Testing* (2nd ed.). Addison-Wesley Professional.

Merdes, M., Malaka, R., Suliman, D., Paech, B., Brenner, D. & Atkinson, C. (2006). Ubiquitous RATs: How Resource-Aware Run-Time Tests Can Improve Ubiquitous Software System. *6th International Workshop on Software Engineering and Middleware, SEM 2006* 55–62.

Mota, S. (2013). *UMA ABORDAGEM PARA ESPECIFICAÇÃO DE REQUISITOS FUNCIONAIS DE UBIQUIDADE EM PROJETOS DE SOFTWARE*. Universidade Federal do Rio de Janeiro.

Myers, G.J. & Sandler, C. (2004). *The Art of Software Testing* (2nd ed.). John Wiley & Sons.

Pai, M., Mcculloch, M., Gorman, J.D., Pai, N., Enanoria, W., Kennedy, G., … Colford, J.M. (2004). Systematic reviews and meta-analyses: an illustrated, step-by-step guide. *Natl.Med J India 17*: 86–95.

Pressman, R. (2010). *Software engineering : a practitioner's approach*. New York: McGraw-Hill Higher Education.

Rocha, A.R.C., Maldonado, J.C. & Weber, K.C. (2001). *Qualidade de software: teoria e prática*. Prentice Hall. Retrieved from https://books.google.com.br/books?id=gBtGAAAAYAAJ

Rodrigues, F., Matalonga, S. & Travassos, G.H. (2014). Systematic literature review protocol: Investigating context aware software testing strategies. Rio de Janeiro. Retrieved from http://www.cos.ufrj.br/~ght/cactus_pr012014.pdf

Ryan, C. & Gonsalves, A. (2005). The effect of context and application type on mobile usability: An empirical study. In *Conferences in Research and Practice in Information Technology Series* (Vol. 38).

Satoh, I. (2003). Software testing for mobile and ubiquitous computing. *The Sixth International Symposium on Autonomous Decentralized Systems, 2003. ISADS 2003.* (Section 2).

Schilit, B., Adams, N. & Want, R. (1994). Context-aware computing applications. *Workshop on Mobile Computing Systems and Applications*.

Software and systems engineering Software testing Part 1:Concepts and definitions. (2013). *ISO/IEC/IEEE 29119-1:2013* 1–64.

Software and systems engineering Software testing Part 2:Test processes. (2013). *ISO/IEC/IEEE 29119-2:2013(E)* 1–68.

Spínola, R.O. (2010). *UMA ABORDAGEM PARA ESPECIFICAÇÃO DE REQUISITOS FUNCIONAIS DE UBIQUIDADE EM PROJETOS DE SOFTWARE*. Universidade Federal do Rio de Janeiro.

Spínola, R.O., Massollar, J. & Travassos, G.H. (2007). Checklist to Characterize Ubiquitous Software Projects. *XXI Simpósio Brasileiro de Engenharia de Software* (1991): 39–55. Retrieved from http://www.lbd.dcc.ufmg.br:8080/colecoes/sbes/2007/SBES03.pdf

Spínola, R.O., Pinto, F.C.R. & Travassos, G.H. (2008). Supporting requirements definition and quality assurance in ubiquitous software project. In *Communications in Computer and Information Science* (Vol. 17 CCIS).

Systems and Software Engineering Vocabulary. (2009). *ISO/IEC 24765:2009* 1–410.

Tang, L., Yu, Z., Zhou, X., Wang, H. & Becker, C. (2011). Supporting rapid design and evaluation of pervasive applications: Challenges and solutions. In *Personal and Ubiquitous Computing* (Vol. 15).

Travassos, G.H., Santos, P.S.M. dos, Mian, P.G., Neto, A.C.D. & Biolchini, J. (2008). An Environment to Support Large Scale Experimentation in Software Engineering. In *13th IEEE International Conference on Engineering of Complex Computer Systems (iceccs 2008)*. IEEE.

Tse, T.H., Yau, S.S., Chan, W.K., Lu, H. & Chen, T.Y. (2004). Testing context-sensitive middleware-based software applications. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.* IEEE.

Walker, B. & Salt, D. (2012). *Resilience practice: building capacity to absorb disturbance and maintain function*. Island Press.

Wang, H. & Chan, W.K. (2009). Weaving Context Sensitivity into Test Suite Construction. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. Auckland: IEEE.

Wang, H., Chan, W.K. & Tse, T.H. (2014). Improving the Effectiveness of Testing Pervasive Software via Context Diversity. *ACM Trans. Auton. Adapt. Syst. 9*(2): 9:1–9:28.

Wang, H., Zhai, K. & Tse, T.H. (2010). Correlating context-awareness and mutation analysis for pervasive computing systems. In *Proceedings - International Conference on Quality Software*.

Wang, Z.W.Z., Elbaum, S. & Rosenblum, D.S. (2007). Automated Generation of Context-Aware Tests. In *29th International Conference on Software Engineering (ICSE'07)*.

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

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*.

Wiener, N. (1961). *Cybernetics; or, Control and Communication in the animal and the machine*. M.I.T. Press.

Winograd, T. (2001). Architectures for Context. *Human-Computer Interaction*.

Yamada, S. & Osaki, S. (1985). Cost-Reliability Optimal Release Policies for Software Systems. *IEEE Transactions on Reliability R-34*(5): 422–424.

APPENDIX A – Smart Camera Requirements

This appendix presents the requirements documentation for the Smart Camera Project, used for the evaluation of CATS Design Process.

A.1 Introduction

A.1.1 Project Purpose

The purpose of this document is to present a description of the SmartCam System. It intends to explain the main idea and features of the system in a resumed way and how the system will react to external stimuli. This document is intended for both stakeholders and developers of the system.

A.1.2 Scope of the Project

This software system shall be an embedded system for a digital camera focused in making the user experience with the device even more comfortable and optimized. This system shall be designed to maximize the user capacity of taking photos with better quality and minimum time spent using sensors and intelligent behavior to adjust the software to better adapt to the sensors readings.

A.2 Overall Description

A.2.1 Functional Requirements Specification

This section outlines the use cases. The system has only one actor, who is the camera user.

Use case: Take a Picture

Brief Description:

The User tries to take a picture.

Initial Step-By-Step Description:

Before this use case can be initiated, the User has already turned the camera ON.

1. The User selects the camera icon on the screen.
2. The User waits for the camera to be ready.
3. The User presses the picture icon.
4. The camera takes the picture and records it in the internal memory.

Use case: Take Several Pictures

Brief Description:

The User tries to take several pictures in a row.

Initial Step-By-Step Description:

Before this use case can be initiated, the User has already turned the camera ON.

1. The User selects the camera icon on the screen.
2. The User waits for the camera to be ready.
3. The User presses the multipicture icon.
4. The camera takes the picture and records it in the internal memory.

Use case: Take a Picture with Follow Action

Brief Description:

The User tries to take a picture using follow action.

Initial Step-By-Step Description:

Before this use case can be initiated, the User has already turned the camera ON.

1. The User selects the camera icon on the screen.
2. The User waits for the camera to be ready.
3. The user presses the LCD screen over the item to be followed by the camera.
4. The camera zooms in the selected spot and follows it if it moves.
5. The User presses the picture icon.
6. The camera takes the picture and records it in the internal memory.

Use case: Visualize Pictures

Brief Description:

The User tries to visualize the pictures already taken.

Initial Step-By-Step Description:

Before this use case can be initiated, the User has already turned the camera ON.

1. The User selects the memory icon on the screen.
2. The camera shows the pictures saved in the internal memory.
3. The user selects the picture he/she wants to see.

A.2.2 Non-Functional Requirements

The camera in which the SmartCam System operates shall have at least 4GB of internal memory space. The physical machine to be used must also have a luminosity

sensor and a flash light attached to it. The battery level and temperature must be readable values for the system as well. The camera needs a touch screen in order to execute the camera functionalities.

A.2.3 Functional Requirements

1. During the camera operation, the device must be able to capture the light intensity and adjust the screen level of luminosity based on it.
2. If the battery level goes below 10%, the user must be informed and the screen must go to the lower luminosity level in order to save battery.
3. If the image on which the camera is focused on is detected to be too dark by the camera, the camera must turn on the flash light automatically.
4. If the camera internal temperature goes higher than 50℃, the flash light must be disabled and the user must be informed.

APPENDIX B – Smart Camera Non-Context-Aware Test Suite

This appendix presents the non-context-aware test documentation for the Smart Camera Project, used for the evaluation of CATS Design Process. This documentation was used to verify the coverage obtained by the CATS Design Process.

B.1 Introduction

B.1.1 Project Purpose

This Software Test Documentation provides a description of test plan of the SmartCam System. This first draft only includes functional testing for the features presented in the Software Requirements Specification document. It covers the general methods made use of in the tests conducted for the system.

Since SmartCam is a product for different kinds of customers (various groups of age and technical knowledge) and also must adapt its behaviors in order to better provide its features to these heterogeneous amount of users, this document was necessary to assure a certain level of quality of those features before they reach the final customers.

B.1.2 Scope of the Project

This software system is an embedded system for a digital camera focused in making the user experience with the device even more comfortable and optimized. This system is designed to maximize the user capacity of taking photos with better quality and minimum time spent using sensors and intelligent behavior to adjust the software to better adapt to the sensors readings.

B.2 Test Plan

B.2.1 Software to be tested

This part is aimed at identifying the items to be tested by the test cases. The software to be tested is the SmartCamera, which requirements documentation is in the APPENDIX A – Smart Camera Requirements.

B.2.2 Test Strategy

The functionalities of the system will be tested considering the conformity with functional and non-functional requirements, in order to verify if the use cases are working as described and obeying the functional requirements.

Input values will be created according to the given specification. For each use case, a test case will be generated considering all functional requirements involved on it. The generated outputs will be compared with the expected ones as described in the requirements documentation.

B.2.3 Test Procedure

B.2.3.1 Functional Use Cases

B.2.3.1.1 Take a Picture

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakePic***
Use Case Base:	***Take a Picture***
Test Objective	*Verify the Take a Picture feature*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The User presses the picture icon.*
Test Inputs:	Press the picture icon
Test Expected Outputs:	*Take a picture*
Post-Condition:	*The camera takes the picture and record it in the internal memory.*

B.2.3.1.2 Take Several Pictures

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakeSevPic***
Use Case Base:	***Take Several Pictures***
Test Objective	*Verify the Take Several Pictures feature*
Precondition:	*The camera must be turned ON*

Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The User presses the multipicture icon.*
Test Inputs:	Press the multipicture icon
Test Expected Outputs:	*Take several pictures*
Post-Condition:	*The camera takes the pictures and record it in the internal memory.*

B.2.3.1.3 Take Picture with Follow Action

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakePicFA***
Use Case Base:	***Take a Picture with Follow Action***
Test Objective	*Verify the Take Several Pictures feature*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The User presses the LCD screen over an item on movement.* 4. *The User waits for the camera to zoom into the selected spot.* 5. *The User presses the picture icon.*
Test Inputs:	Press over the spot Press the picture icon
Test Expected Outputs:	*Take a picture of the spot under movement with zoom*
Post-Condition:	*The camera takes the picture and records it in the internal memory.*

B.2.3.1.4 Visualize Pictures

Test Suite ID:	***FuncTest001***
Test Case ID:	***VisPic***
Use Case Base:	***Visualize Pictures***

Test Objective	*Verify the Visualize Pictures feature*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the memory icon on the screen.* 2. *The camera shows the pictures saved in the internal memory.* 3. *The user selects a picture to see.*
Test Inputs:	Select a picture.
Test Expected Outputs:	*The system shows the picture.*
Post-Condition:	*The picture is shown to the user.*

B.2.3.1.5 Light Intensity Low

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakePicLIL***
Use Case Base:	***Take a Picture***
Test Objective	*Verify the Take a Picture feature with light intensity variation*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The camera is positioned in a low lighted environment.* 4. *The User presses the picture icon.*
Test Inputs:	Reduce light intensity Press the picture icon
Test Expected Outputs:	The flash light goes ON The screen goes lighter Take a picture
Post-Condition:	*The camera takes the picture and record it in the internal memory.*

B.2.3.1.6 Light Intensity High

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakePicLIH***
Use Case Base:	***Take a Picture***
Test Objective	*Verify the Take a Picture feature with light intensity variation*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The camera is positioned in a very illuminated environment.* 4. *The User presses the picture icon.*
Test Inputs:	Increase light intensity Press the picture icon
Test Expected Outputs:	The screen goes darker Take a picture
Post-Condition:	*The camera takes the picture and record it in the internal memory.*

B.2.3.1.7 Battery Low

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakePicBL***
Use Case Base:	***Take a Picture***
Test Objective	*Verify the Take a Picture feature with battery level variation*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The camera battery goes below 10%* 4. *The User presses the picture icon.*
Test Inputs:	Battery reduction below 10% Press the picture icon

Test Expected Outputs:	System Warning of Low Battery The screen goes darker Take a picture
Post-Condition:	*The camera takes the picture and record it in the internal memory.*

B.2.3.1.8 High Temperature

Test Suite ID:	***FuncTest001***
Test Case ID:	***TakePicHT***
Use Case Base:	***Take a Picture***
Test Objective	*Verify the Take a Picture feature with high temperature*
Precondition:	*The camera must be turned ON*
Test steps:	1. *The User selects the camera icon on the screen.* 2. *The User waits for the camera to be ready.* 3. *The camera temperature goes higher than 50℃.* 4. *The User presses the picture icon.*
Test Inputs:	Temperature over 50℃ Press the picture icon
Test Expected Outputs:	System Warning of High Temperature Take a picture without flash light
Post-Condition:	*The camera takes the picture and records it in the internal memory.*

APPENDIX C – CAUS Context-Aware Test Suite

This appendix presents the context-aware test suite for the CAUS Project generated using the CATS Design Process.

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

Test Case ID:	***CATS - TC002***
Test Objective	*Verify the QR code scan feature*
Precondition:	*Be logged into the system*
Test Input:	*A valid QR code for CAUS*
Test steps:	*1. The User selects the QR code scan option*

	2. *The system executes the QR reader application* 3. *System reads the QR code* 4. *The system shows the menu for the user*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *QR Code*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *QR Code unreadable*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Notify the user and do not proceed the transition*

Test Case ID:	***CATS - TC003***
Test Objective	*Verify the play store feature*
Precondition:	*Be logged into the system*
Test Input:	*A valid QR for CAUS*
Test steps:	1. *The User selects the QR code scan option* 2. *The System notice that there is no QR code reader installed in the mobile* 3. *The System redirects the user to the play store and suggests a QR code reader* 4. *The User selects a QR code reader* 5. *The System install the QR code reader and resume the application*
Relevant Context Variables:	1. *University WI-FI*

	2. *Application Focus* 3. *Server Availability* 4. *Internet connection*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Loss of internet connection*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC004***
Test Objective	*Verify the map feature for user location*
Precondition:	*Be logged into the system* *Have read a QR code*
Test Input:	*Select the map option*
Test steps:	1. *The User selects the map option* 2. *The System shows the user location*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified*

	d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC005***
Test Objective	*Verify the workshop information feature*
Precondition:	*Be logged into the system* *Have read a QR code*
Test Input:	*Select a workshop*
Test steps:	1. *The User selects a workshop* 2. *The System shows the workshop information*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC006***
Test Objective	*Verify the user information feature*
Precondition:	*Be logged into the system* *Have read a QR code*
Test Input:	*Select a user*
Test steps:	1. *The User selects another user* 2. *The System shows a menu for the possible actions*

Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded*
Test Expected Outputs for each Threshold:	a. *Notify the user and close the application* b. *Continue the application from where it stopped* c. *Not specified* d. *Not specified*

Test Case ID:	***CATS - TC007***
Test Objective	*Verify the user schedule information feature*
Precondition:	*Be logged into the system* *Have read a QR code* *Have selected a user*
Test Input:	*Select a schedule*
Test steps:	1. *The User selects the schedule option* 2. *The System shows a calendar with the schedule is shown to the user*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped*

	c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC008***
Test Objective	*Verify the user availability information feature*
Precondition:	*Be logged into the system* *Have read a QR code* *Have selected a user*
Test Input:	*Select the availability option*
Test steps:	1. *The User select the availability option* 2. *The System shows the user availability*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC009***
Test Objective	*Verify the send mail feature*
Precondition:	*Be logged into the system* *Have read a QR code* *Have selected a user*
Test Input:	*Send email*

Test steps:	1. *The User selects the send mail option* 2. *The System shows a menu to be completed* 3. *The User completes the menu and selects the option Send* 4. *The System sends the email*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Internet Connection*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Loss of internet connection*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC010***
Test Objective	*Verify the notify visit feature*
Precondition:	*Be logged into the system* *Have read a QR code* *Have selected a user*
Test Input:	*Select the notify visit option*
Test steps:	1. *The User selects the notify visit option* 2. *The System shows a form to be completed* 3. *The User completes the form and selects the option Send* 4. *The System saves the notification*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus*

	3. *Server Availability*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified*

Test Case ID:	***CATS - TC011***
Test Objective	*Verify the change availability feature*
Precondition:	*Be logged into the system* *Have read a QR code*
Test Input:	*A valid state for availability*
Test steps:	1. *The User selects the update availability option* 2. *The System shows the possible options* 3. *The User selects an option* 4. *The System saves the user new availability*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified*

Test Case ID:	***CATS - TC012***
Test Objective	*Verify the check visits feature*

Precondition:	*Be logged into the system* *Have read a QR code*
Test Input:	*Select check visit*
Test steps:	1. *The User selects the visits history option* 2. *The System shows the notifications*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information Update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC013***
Test Objective	*Verify the CRUD user feature regarding creation*
Precondition:	*Be logged into the system*
Test Input:	*Valid user information*
Test steps:	1. *The User selects the register option* 2. *The System shows the form* 3. *The User completes the form and selects the Save option* 4. *The System saves the new user*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability*
Known Thresholds:	a. *Loss of University WI-FI connection*

	b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified*

Test Case ID:	***CATS - TC014***
Test Objective	*Verify the CRUD user feature regarding read and delete*
Precondition:	*Be logged into the system*
Test Input:	*Select user management and delete*
Test steps:	1. *The User selects the user management option* 2. *The System shows a list of the system users* 3. *The User selects a user and selects the option Delete* 4. *The System removes the user register*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information Update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC015***
Test Objective	*Verify the CRUD user feature regarding update*
Precondition:	*Be logged into the system*
Test Input:	*Valid user information*
Test steps:	1. *The User selects the user management option* 2. *The System shows a list of the system users* 3. *The User selects a user and selects the option Update* 4. *The System shows the form with the user information* 5. *The User updates the form and select the option Update* 6. *The System saves the user information*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information Update during the user access*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified* e. *Not specified*

Test Case ID:	***CATS - TC016***
Test Objective	*Verify the register event feature*
Precondition:	*Be logged into the system*
Test Input:	*Valid event information*
Test steps:	1. *The User selects the create event option*

	2. *The System shows a form* 3. *The User completes the form and select the option Create Event* 4. *The System creates the event*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded*
Test Expected Outputs for each Threshold:	a. *Notify the user and close application* b. *Continue application from where it stopped* c. *Not specified* d. *Not specified*

Test Case ID:	***CATS - TC017***
Test Objective	*Verify the start event feature*
Precondition:	*Be logged into the system*
Test Input:	*Select start event*
Test steps:	1. *The User selects the see events option* 2. *The System shows a list of events* 3. *The User selects an event and selects the option Start Event*
Relevant Context Variables:	1. *University WI-FI* 2. *Application Focus* 3. *Server Availability* 4. *Information Update*
Known Thresholds:	a. *Loss of University WI-FI connection* b. *Loss of focus* c. *Server goes offline* d. *Server limit is exceeded* e. *Information Update during the user access*

Test Expected Outputs for each Threshold:	*a. Notify the user and close application* *b. Continue application from where it stopped* *c. Not specified* *d. Not specified* *e. Not specified*