Literature review of formal testing on Adaptive Systems

R. Caya, J.Neto

***Abstract*— This paper presents a review of the literature about the work on the model-based testing area to generate test cases for adaptive systems. We present a brief definition of the testing software activity, its importance as general parameter of software quality, and the particular need for having proper methods to perform it the domain of adaptive systems. We describe as well, the core principle bellow adaptive technology and present the reasoning for the need performing automatic testing for these systems. As result of our literature review we recover the most used formalisms and testing strategies used for adaptive technologies. Finally we present some insights and conclusions derived from the information we gather about the matter of model-based testing adaptive systems.**

***Keywords*— review, adaptivity, testing, state machine, test cases, software testing tools.**

I. INTRODUCTION

Testing is a significant part of software engineering and with the evolution of computer languages, methodologies and techniques that speed up the implementation of complex systems the task for testing and guaranteeing the quality has become increasingly demand [1]. There are several definitions for what is testing and what it aims according with different approaches and methodologies. Highly recommended documentation and specification about the testing process and its taxonomy are provided at [2] [3]. Even though, a in [4] Miller defines the goal of testing as:

"to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances."

In Miller's approach, a good test is one that has a high probability of finding an as yet undiscovered error, and a successful test is one that uncovers an as yet undiscovered error. As critical as it is, this process can be very time and resource consuming, being applied in different parts of the software developing cycle, from specification to maintenance. According to [1], by 2001, testing typically consume from 40% up to 50% of the software development effort.

Unfortunately, even being as critical as it is, the carry out of testing as formal and strict process is not usually integrated within the software development process [5]. As consequence, some of practitioners on the software industry consider it too expensive, case involves the production of artifacts useful for testing, but not considered at the beginning, i.e. clear or formal requirement specification. However, the guarantee of the quality of a software is a feature that different enterprises demand, some because it is the attribute that identify the trademark, and other because the critical nature of the system, in terms of reliability and robustness, established to be so, i.e. high-confidence medical cyber-physical systems [6].

Particularly, those technologies that manifest a mechanism that allows them to self-managing the decision over changes in its own configuration on run time, technologies that implement adaptive features, have being increasingly research in the past decade [7]. This interest might be consequence of the new needs exposed by the users of the, so called, "Information Society" that claim for flexible systems that can respond to the constant changes in their environments. The whole work at the Laboratório de Tecnologias Adaptativas (LTA) at the University of São Paulo focuses in the concepts, theories and mechanisms that model and implement adaptive behavior. Systems and devices that are complex, heterogeneous and perform dynamic changing behavior can be found in a very wide spectrum of areas, some examples are [8]: Virtual Reality applications, Multi-agent Systems [9], Simulations, Natural Language Processing, Human-Computer Interaction, Computing Theory, Organization and Orchestration of service-oriented software, Pattern Recognition strategies, and many more. An additional care must be taken when looking for adaptive systems and that is the proliferation of terminology [10] that tries to comprise the phenomenon that characterizes adaptivity: self-managing the changes in the configuration of the system (reflected by its behavior) at run time and as consequence of external stimuli.

The motivation for this work is to identify which elements from the testing field can assist developers to examine quality parameters in adaptive systems.

Our objective is to review literature to retrieve the main approaches followed in the area of testing to convey formalism, techniques, strategies that allow the specification of adaptive systems, followed by the generation of test cases that are adequate to guarantee the quality of the system.

The next sections in this document are organize as follows: Section II describe the main concepts related to this work, so it present testing as activity in software development, its classical approaches and formalisms. In the same way, we present the definition some of the well know strategies to conduct testing and to guarantee the quality of a software: Model-Based testing and Property-based testing. Section III we present the definition of Adaptive Technology and its basic features. Section IV describes the formal languages and models used in the literature to perform software testing for adaptive systems. Section V describes the results and insights

R. Caya, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, rosalia.caya@usp.br

J. J. Neto, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, joao.jose@poli.usp.br

we were able to reach after performing the literature review. Finally, Section VI state the conclusions of the review, some of the difficulties founded and possible future works in the endeavor of applying MBT to adaptive systems.

II. TESTING: DEFINITIONS AND APPROACHES

In this section we define testing as activity in the software development life cycle, the diffent approaches followed by researchers from several fields and some of the formalisms used for work in this area.

A. *Testing as activity*

The task of test a program is at least as old as computing itself. However, Software industry have suffer tremendous growth over the past three decades. Software applications have proliferated from the original task-oriented scientific computing domains into our daily lives in such a way that sometimes we do not realize that some kind of computation is performed when we do everyday tasks. Of course, very highly specialized and complex systems are coming to the rescue in different arenas, such as cyber-physical systems, virtual technology, and medical and military applications, as many others. This proximity implies that software is required to fulfill some quality attributes(to be usable, dependable, and safe) because of its impacts in our daily lives, such as economy, personal and national security, health, and safety. Three decades ago, testing accounted for about 50% of the total time and more than 50% of the total money expended in a software development project—and, actually the percentage is sort of the same today, with a tendency to grow. Actually, global competition, outsourcing, off-shoring, and increasing customer expectations have brought the concept of quality to the forefront. Developing quality products on tighter schedules is critical for a company to be successful in the new global economy. Traditionally, efforts to improve quality have centered around the end of the product development cycle by emphasizing the detection and correction of defects. On the contrary, the new approach to enhancing quality encompasses all phases of a product development process, from a requirements analysis to the final delivery of the product to the customer. Every step in the development process must be performed to the highest possible standard. As consquence, a software system goes through four stages of testing before it is actually deployed. These four stages, as mentioned in [11] and shown in Figure 1, are known as unit, integration, system, and acceptance level testing.

The objective of testing is to deliver a reasonably stable, roboust system that can satisfy the needs of the user and at the same time meeting a tight schedule, to discovering most of the faults, and to verifying that fixes are working and will not result in new faults. In general, software testing is a highly labor intensive task, it comprises a number of distinct activities, and it consumes a lot of resources: time and human. This high cost is one of the reasons why test automation is very attractive. With the help of proper tools the durations of those tasks can be shortened, and the skills of est engineers can be focus in designing and developing better and refined strategies to generate automated test cases instead of manual and extenuating work.

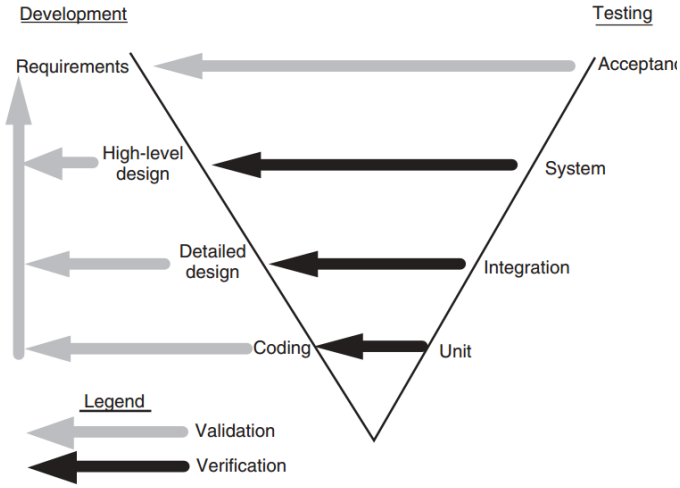

Figure 1. Software Testing Four Stages presented at [11].

B. *Classical Approaches*

As stated before, seems that test automation may be of great help, but it comes with a cost: sufficient time and resources need to be allocated for the development and maintaining of an automated test suite. Different approaches consider the sources from which it will be designed, and others consider the moment in the software life-cycle in which testing will be applied.

In the first case, test cases need to be designed by considering information from several sources, either from the requirements specification of a system (functional or black box testing) or from source code (structural or white box testing). In specification-based testing a key element is that the requirements had been specified in a formal manner, so they are a precise and detailed description of the system's functionality and constraints so the can be a the point of reference for test data selection and adequacy.

In the second case, it is necessary to distinguish between two similar concepts related to software testing: validation and verification. In case of validation, it establishes the correspondence between a system and users' expectations. In this type of testing a key element is the specification of what functionality is needed and which additional properties the system should posses. The objetive of validation is to determine if the entire system meets the customer's needs and expectations. In case of verification, a key element is building a high-level description (abstraction) of the functionality the system is supposed to have. Then, the whole testing process is about making sure of the correspondence of an implementation of the software with its specification.

In case of this work we are interested in study the automation of functional test cases towards verification of a system. This can be done by property extraction or model-

based testing.

C. *Model-based testing:*

The goal of model based testing is to show that the implementation of the system behaves compliant with this model. Model based testing uses a concise behavioral model of the system under test, and automatically generates test cases from the model [12].

As mention before, the goal of testing is to identify differences between the behavior of the implementation and the intended behavior of a system under test (SUT), as expressed by its requirements [13]. Usually, the model of the SUT is given as a black box that accepts inputs and produces outputs, but also exists testing processes that work with white and gray box approaches [3]. Model-Based Testing (MBT) is an area of testing in which the work relies on explicit abstract formal behavior models that encode the behavior of the SUT or its execution environment or sometimes both of them [14]. In this context, abstraction is not only beneficial, but also necessary for hiding unnecessary programming details as well as reducing the complexity of the system. However, it is also essential for the test model to be detail enough in order to generate effective test cases. The model must describe possible input/output sequences, and is linked to the implementation by a conformance relation. Finding the right level of abstraction for the test model is one of the challenges in MBT and is for now regarding to the test team.

The basic idea, and main practical advantage, of MBT is that from a formal or semi-formal model complete test cases can be generated. Another expected advantage of MBT is that it is hope to mitigate some well know problems of deriving tests "by hand", most of all because of its dependency on the approach of single engineers [3]. The ideas of MBT date back to 70's with studies about specification-based testing [15] [4]. Based on these studies, software system now can be specified in more rigorous, understandable, clear ways, which has brought great chances to improve automated functional testing techniques. Meanwhile, software development has evolve from standalone systems to complex, heterogeneous, distributed, real-time, and component based systems. This evolution joined by the advances reached in the area of formal verification have led to an increased level of interest from the academic field as well as from the industry.

However, the methods and tools used in MBT are dependent of the creation of the formal model for the system under test. This way, MBT does not replace traditional test design, but supports them in a way that allows testers to improve their understanding of the domain and to perform tests more effectively and efficiently.

Nowadays, a variety of models and different notations are available to present the system from different perspectives. Some formal notations use Finite State Machines, grammars and set theory to present precise semantics. Some other prefer more visual representations, such as diagrams and tables, to facilitate communication with different audiences. Some of the most known models are:

- UML
- Statecharts and Extended Finite State Machines
- Petri nets
- Tasks models
- Z, Larch, VDM, Pre/Post Conditions, Estelle, and λ-calculus, and π-calculus

The multitude of specification models and its semantics allow different degrees of conduciveness for analysis and impose a difficulty for people looking to develop encompassing technology for model-based testing. In [16] some of the typical misleading expectations, misunderstandings and pitfalls are point out as:

- MBT solves all problems;
- MBT is just a matter of tooling;
- Models are always correct Test case explosion will occur.

Nowadays, the research in this field has ignite. Several international conferences (ACM TAAS, IEEE ICCAC, SEAMS, Adaptive) have been created on the specific topic, aiming to explore, develop, implement and compare different techniques according to the features of the SUT. In the same way, some of the most important events of the area of Software Development have incorporated the MBT and formal verification as a track of interest.

III. ADAPTIVE SYSTEMS

Adaptation is a promising approach to manage the complexity embedded in contemporary systems [12] [7]. An adaptive system is an entity able to adapt autonomously, reconfigure at run-time, to internal dynamics according to sensed conditions in the environment to achieve a particular goal. [12] [17]. The increasingly complexity and heterogeneity in nowadays systems make difficult to apply static strategies to reflect dynamic behavior, so adaptive methods and technology has become an important research topic in many diverse application areas [18]. One of the more recently growing fields in research related with adaptivity is in Software Engineering [18] under the terminology of *SaS* (Self-adaptive Systems). This way has become necessary to include adaptivity in the research agenda of technological community.

Basically, an adaptive system, as shown in Figure 2, comprises two parts: the managed system (also called system layer, managed resources, core function, underlying subsystem as presented in [7]) and the managing system (or architecture layer, autonomic manager, adaptation engine, reflective subsystem, adaptive layer, as presented in [7]) .

The underlying subsystem deals with the domain functionality and the adaptive layer deals with the adaptations of the managed system to achieve particular quality objectives. The key underlying principle of adaptive systems is complexity management through separation of concerns.

A. *Adaptive Technologies at LTA*

The Laboratório de Tecnologias Adaptativas (LTA) at the Politechnical School of the University of São Paulo have develop several researches studying adaptive technology since the late 90s. As result several formalisms, techniques, methods, tools and applications were develop based on the incorporation on the features of adaptive behavior. In these researches, adaptivity is applied in different areas, such as formal languages [19], human-computer interaction [20], formal models [21] [22], data mining, robotics [23], and so on, proving the potential and versatility that the basic and intuitive concept of adaptivity has. In the context of the works at the LTA, adaptivity is understood as the ability a system has to respond to external stimuli, deciding and applying dynamic self-reconfigurations in its structure [8] and behavior [22], taking into account the historical information stores from previous executions [19]. This way the term adaptive technology refers to the application of adaptivity to practical and concrete purposes, meaning that an adaptive device is a rule-driven device able in an independent way of applying changes in that same ruleset and processing the corresponding actions.

One interesting feature about adapting entities is that even when most of the time different research laboratories do not have close collaborations the approach about the foundations of adaptivity seem concomitant. For example, the core formulation developed and currently use to define adaptive devices at LTA is in close concordance with the general one presented at the begging of this section. In the formalism it is possible to identify two major components [22]: the underlying device (usually a non-adaptive one), and the adaptive mechanism, responsible for the incorporation and well performance of adaptivity. The fact that exist a major concordance about the core formalism below adaptive technology is very important because allow us to gather several researches that otherwise could remain as disseminated efforts. This work is actually one of several efforts aiming at helping in the integrations of the theory of adaptive technologies and other fields in computing.

However, the importance of study adaptivity does not hold only in creating adaptive devices, it also covers the understanding of the different features and properties implied by it. For example, to correctly incorporate and create adaptive devices on-the-fly, through strategies as negotiation in Multi-Agent Systems or orchestration in Service-Oriented Software, it will be necessary to identify the properties this dynamic entities should satisfy. Furthermore, researching and exploring literature about formal representation allows developing appropriate tools to apply technology available for non-adaptive software.

It also allow us to distinguish very closely related but different terms by understanding its differences, as is the case with adaptability and adaptivity.

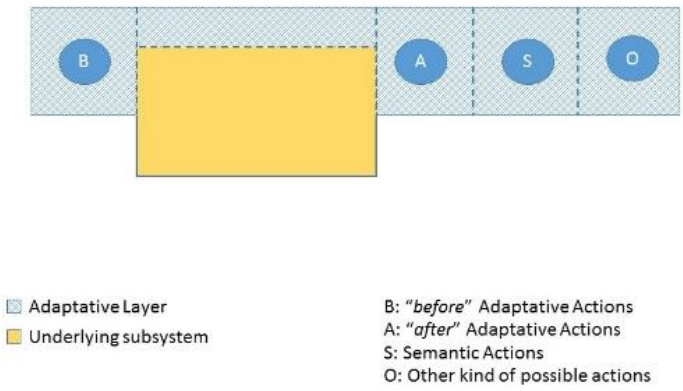

Figure 2. Basic formulation of an adaptive device at LTA

IV. TESTING ON ADAPTIVE SYSTEMS

As we have seen before, the complexity and heterogeneity in nowadays systems is requiring new technologies to be develop and to measure the quality of such systems. In this context, one major challenge in self-adaptive systems is to assure the required quality properties [7]. This is the kind of task in which the work developed in Testing (Validation and verification) can assist. However, before applying the algorithms ad strategies from testing it is necessary to satisfy its requirements and avoid understand its constraints. As an emerging partnership testing adaptive software needs to face some intrinsic challenges, some of them are:

- Handle its natural complexity, uncertainty and incomplete information
- The expansion of decision space
- Handling error propagation

We need a formal model to support the specification of such systems [12], we need to understand how the adaptive features are encoded in such models, we need to choose the proper testing criteria to check those properties [12], and we need to use some kind of formalism to process the model and generate the test cases. Adaptive features are, in formal terms, behavioral properties [12] in the system.

The necessity of testing adaptive systems has been recognized not only by the academic community researching in adaptive behavior, or the software development arena, it has being pointed out as a necessity in the testing community [24].Because this systems are increasingly entering the industry venue and there is need for certification from costumers [17]. It is necessary to provide validation and verification methods to motivate the adoption of adaptive technology by industry and unlock the potential of adaptivity beyond academic arena [25]. Notwithstanding, as we have stated before the task of

testing is by itself very extenuating even in the case of software with classical behavior, so in the case of adaptive, mostly changing, partly undefined behavior the complexity rises. This way, is not possible to perform the activities required for testing by hand because the amount of information that needs to be consider by testers, for designing the tests, incorporate into the test strategy, monitoring, and evaluate, is huge and make the task a herculean mission for engineers.

To design and develop automatic test case generation it is fundamental to describe the system's behavior in a precise way. This is the main objective of formal specification techniques.

Formal specification differs from other type of specification in its high level of detail, precision and correctness. A formal specification language is compose by a clear syntax, a precise semantics within the domain and a proof theory [26]. Whereas other notation aim for more visual or user-friendly presentations, formal languages usually have a syntax based in mathematical techniques that allows analyzing and applying reasoning over it. The branch of mathematics used is discrete mathematics and the mathematical concepts are drawn from set theory, logic and algebra. The use of formal methods is increasing in the area of critical systems development, where emergent system properties such as safety, reliability and security are very important. In such areas, the possibility of automatic derivation over an specification is one of the main advantages of formal languages over informal ones. Two fundamental approaches to formal specification have been used to write detailed specifications for industrial software systems. These are: algebraic and model-based. A basic description of the most used techniques in each case will be given in the following sections.

B. Languages for formal specification of Adaptive Systems

The principle in this approach is to specify a system as a structured collection of mathematical functions. The can be grouped by defining algebraic structures, logical theories or as a structured collection of processes.

Some of the formal languages used in studies to specify systems with dynamic reconfigurations, such as adaptive systems, are:

- **Process calculi:** is a family of approaches for formally modelling concurrent systems that provide a high level description of interactions, communication strategies and synchronization between a collection of individual processes. The most used are π-calculus [27] [9] and LOTOS.
- **Timed automaton** [7] [28]**:** is a theory for modeling and verification of real time systems. It is basically a finite automaton extended with real-valued variables.

C. Models for formal specification of Adaptive Systems

Model-based specification exposes the system state and defines the operations in terms of changes to that state. Some of the used models to describe adaptive systems are:

- **Petri nets** [29]: is a state-transition system mainly used to work with distributed systems. It offers an exact mathematical definition of their semantics, a well-developed mathematical theory for process analysis and a graphic representation. Petri nets are well suited for modeling the concurrent behavior of distributed systems. Several extensions of the original formalism created by Carl Adam Petri at 1939 to meet some other features [30].
- **Z**: The Z notation is based upon set theory and mathematical logic. The set theory used includes standard set operators, set comprehensions, Cartesian products, and power sets. The mathematical logic is a first-order predicate calculus. A characteristic feature of Z is the use of types. A variation called uSZ [13] has combine statecharts with Z and temporal logic.

- **Different kinds of EFSM (**Extended Finite State Machine**) and hybrid approaches:** Characterize the required transitions from state to state. The properties of interest are specified by a set of transition functions in the state machine transitions [26]. Statecharts, are one of the most used formalisms to specify dynamic systems behavior because it offers hierarchy, concurrency, broadcasting strategy and history states.

The main observation made at [7] is that regular algebra is one of the most used modeling languages. One of the reasons for that besides the strong community of formal verification in computer science, is that the majority of studies working in formal specification of adaptive systems formulate the properties of interest in the modeling language they use for modeling.

The use of traditional formal modeling languages such as transition systems, automata, state machines and EFSM, Markov models, graphs, and process algebras (*μSZ* [13], λ-calculus, and π-calculus) is about equally distributed and most of the time some type of logic (frst order, deontic, modal and temporal) as property specification language is combined with other modeling languages.

D. Models for testing adaptive systems

Some of the most used models in literature to design testing strategies for systems that possess adaptive features are the following:

- **Linear Temporal Logic**: convenient formalism for specifying and verifying properties of reactive systems [31]. The strategy for testing (by theorem proving or model checking) is to express desired properties using LTL operators and actually check if the model satisfies this property. This model provides a particularly useful set of

operators for constructing LT properties without specifying sets.

- **Computation Tree Logic CTL*** [7]: is a temporal logic where full computation tree logic has no syntactic restrictions on the applications of temporal operators and path quantifiers, allowing explicit quantification over all possible futures starting at some initial state.

- **Communicating Sequential Processes**: [32] formal language for describing patterns of interaction in concurrent systems [29]. CSP has been successfully applied in industry as a tool for specifying and verifying the concurrent aspects of a variety of different systems.

V. RESULTS FROM SYSTEMATIC ANALYSIS OF LITERATURE

From the performed literature review, it is possible to obtain some key insights about the state of the art of testing adaptive software. They are presented grouped by concerns in an intent for clarity.

A. About systematic reviews

There is a clear need for performing literature review, in particular systematic review that allow incoming researchers get organized, clear and synthetized information about specific endeavors to solve current challenges. Following the same logic it is also important to give continuity to work already available to keep the information up to date [7], especially for uses in industry.

B. About Testing criteria for testing adaptive systems

Some well-known classic testing criteria have been applied in adaptive systems, such as: simple, statement coverage, pair-wise, dependency-based and path coverage giving some results about quality. However, the need for exploring and formulate proper criteria to specifically test adaption properties remains a challenge [17] [24].

In the same manner, some requirements for testing adaptive system have been developed in resent research projects [17]. A new approach is the one presented at [24] called Veritas, in which a technique for generating test cases that evolves and customize the set of test cases, and/or parameters, to correspond the current conditions of the adaptive system.

C. About testing models for adaptive systems

Some models have been used to specify adaptive systems, most of them were recommended by literature because they consider properties closely related to adaptivity such as: concurrency, distributive components, communication strategies, and historical records. Most existing formal approaches for self-adaptive systems assume a central point of control to realize adaptations [12].

A particular innovative proposal for organizing the testing activity is the one made in [12] and presented at Figure 3 called zones in the state space for different behaviors of adaptive systems. In the *normal behavior zone*, the system is performing its canonical functionality. In the *undesired behavior zone*, the system has entered in a state where adaptation is required. In the *adaptive behavior zone*, the system is adapting itself to manage the undesired behavior, if the adaptation succeed the adjusted system will return to a normal behavior. In case the adaptation fails or generate consequences that can no be properly handled by the system it will enter in the last zone. Finally, the *invalid behavior zone* corresponds to states where the system should never be. One of the most interesting contributions of this mapping model is the possibility to map properties of interest for testing with respect to the system model using transitions between different zones.

Another interesting proposal is one presented at [33]. Their approach uses evolutionary computation to adjust requirements-based test cases at run time so the test suite can handle different system and environmental conditions.

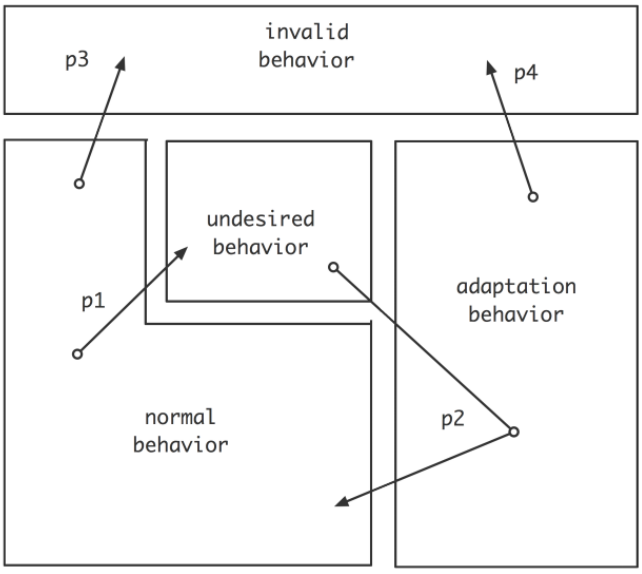

Figure 3. Zones in the space state of an adaptive system.

D. About properties of interest in adaptive systems

From the studies reviewed [7] [25] [24] [33] [27] we can report that the top concerns of interest in adaptation are the following:

- efficiency/performance,
- reliability,
- resilience [25]
- guaranteeing functionality, and
- flexibility [12].

Other properties measure through model testing are: robustness [25] [12], and correctness of adaptations [12].

Traditional properties, including safety, liveness and reachability [12] and deadlock are tested too, but in less frequency.

VI. CONCLUSION

After performing the review we arrive to some conclusions about the challenges that remain open and some of the issues in this research field that require attention from the academic community.

First, we do not found models developed to specify the particularities of adaptive systems behavior. There are formalisms that provide some tools to translate adaptive to some of its properties, but some of them do not allow an entire mapping. Moreover, the very use of formal methods as evidence of system properties remains limited. The development of environment models are an essential condition for model based testing of runtime qualities, which is central to self-adaptation [12]. It allows an engineer to specify in a precise manner the failure scenarios of interest and the conditions under which the failures happens.

Second, no standard tools have been emerged for formal modeling and verification of adaptive systems. [7].

Third, a particular difficult step in the testing activities becomes extremely complex in case of adaptive systems: identify the properties or concerns it is important to test. Several factors need to be consider: the domain of the system, the focus of the test, the strategy of the testing criteria, and the processing of the results. In the literature review we observed that most of the self-* properties (self-healing, self-configuring, and self-optimizing, self-organizing) which are able to form emergent behavior are not properly specified in testing studies. Actually, only self-organizing property has being point out as flexibility in some studies [24].

Four, once we know the properties we need to test selecting the right testing criteria according to them becomes a cumbersome task. The test engineer must take into account controlling the combinatory explosion product of either state space, decision space, path or propagation at executing the strategy. Few information about this concern was found in the literature review, which raises the question about how to deal with these dangers in an efficient way.

Five, managing the testing activity requires not only good technology, but also well prepared managers. This way, Software testing engineers need to be properly train in the use of formal methods to be able to design suites of tests, evaluate the adequacy of testing criteria and analyze the results with mathematical methods. Most of the time testing theory is not included in the curricula of several courses of either software engineering or Computer Science.

Six, there is a need for optimizing test strategies to reduce the consuming of resources, high costs, tenuous manual efforts, and processing of failure at test design activities.

Finally, we considered that performing a systematic revision help us to assemble a consistent vision about adaptivity at two levels: an inner level and an outside level. At inner level we could verify that adaptivity consist on a simple and intuitive concept, a feature, that can be applied to solve or empower different areas. At the outside level, we could gather information in a systematic and organized way and verify the different potential areas in which adaptivity can be beneficial.

This way we consider that systematic reviews help to deeply understand the area of interest and the key information about our particular problem. It is our thought that motivating researchers at early stages of academic life is very fruitful to strength and clarify the emerging area of adaptive technology.

VII. REFERENCES

[1] L. Luo, "Software testing techniques - technology maturation and research strategy," Institute for Software Research International, Carnegie Mellon University, Pittsburgh, 2001.

[2] M. Y. Shafique, "A systematic review of state-based test tools," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 1, p. 59–76, February 2015.

[3] M. Utting, A. Pretschner and B. Legeard, "A taxonomy of model-based testing approaches," *SOFTWARE TESTING, VERIFICATION AND RELIABILITY*, vol. 22, p. 297–312, 12 April 2012.

[4] E. F. Miller, "Introduction to Software Testing Technology," in *Tutorial: Software Testing & Validation Techniques*, Second ed., Vols. IEEE Catalog No. EHO 180-0, IEEE Computer Society, 1980, pp. 4-16.

[5] A. C. Dias Neto, R. V. M. Subramanyan and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, Atlanta, Georgia, 2007.

[6] E. A. Lee, "Cyber physical systems: Design challenges," in *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, Orlando, Florida, 2008.

[7] D. a. I. M. U. a. d. l. I. D. G. a. A. T. Weyns, "A Survey of Formal Methods in Self-adaptive Systems," in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering, C3S2E '12*,

Montreal, Quebec, Canada, 2012.

[8] J. J. Neto, "Um levantamento da evolução da adaptatividade e da tecnologia adaptativa," vol. 5, no. 7, p. 496–505, 2007.

[9] W. Jiao, M. Zhou and Q. Wang, "Formal framework for adaptive multi-agent systems," in *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003*, 2003.

[10] K. Compton and S. Hauck, "An introduction to reconfigurable computing," *IEEE Computer*, April 2000.

[11] K. a. T. P. Naik, "Chapter 10: Test Generation from FSM Models," in *Software Testing and Quality Assurance: Theory and Practice*, New Jersey, John Wiley & Sons, Inc, 2008.

[12] M. U. a. D. W. Iftikhar, "A case study on formal verification of self-adaptive behaviors in a decentralized system," in *11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA'12)*, 2012.

[13] K. Bogdanov, "Atomated testing of Harel's statecharts (PhD. Thesis)," University of Sheffield, Department of Computer Science, Sheffield, 2000.

[14] F. Siavashi and D. Truscan, "Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literature review," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, New York, 2015.

[15] T. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Transactions on Software Engineering*, Vols. SE-4, no. 3, pp. 178 - 187, May 1978.

[16] The International Software Testing Qualifications Board, "Model-Based Tester Extension Syllabus," The International Software Testing Qualifications Board, 2015.

[17] G. Püschel, S. Götz and C. a. A. Wilke, "Towards Systematic Model-based Testing of Self-adaptive Software," in *ADAPTIVE 2013, The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications*, n Valencia, Spain, 2013.

[18] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic and R. Desmarais, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," in *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, 2013.

[19] J. J. Neto and A. V. Freitas, "Using Adaptive Automata in a Multi-Paradigm Programming Environment," in *International Conference on Applied Simulation and Modelling, ASM2001 - IASTED*, Marbella, Espanha, 2001.

[20] J. J. Neto, "Um Glosário sobre Adaptatividade , Laboratório de Linguagens e Técnicas Adaptativas," in *Memórias do WTA 2009 - Terceiro Workshop de Tecnologias Adaptativas*, São Paulo, Brasil, 2009.

[21] J. Neto and C. Pariente, "Adaptive Automata - a reduced complexity," in *Proceedings of the Conference on Implementation and Application of Automata–CIAA*, Tours, France, 2002.

[22] A. H. Tchemra, "Aplicação da Tecnologia Adaptativa em Sistemas de Tomada de Decisão," *IEEE America Latina*, pp. 552-556, 2007.

[23] M. de Sousa, A. Hirakawa and J. Neto, "Adaptive Automata for Mapping Unknown Environments by Mobile Robots," in *Proceedings of the Ibero-American Conference on Artificial Intelligence*, 2004.

[24] B. a. S. H. a. K. A. a. R. W. Eberhardinger, "Towards Testing Self-organizing, Adaptive Systems," in *Proceedings of International Conference on Testing Software and Systems: 26th IFIP WG 6.1 , ICTSS 2014*, Madrid, Spain, 2014.

[25] J. Cámara, R. de Lemos, N. Laranjeiro, R. Ventura and M. Vieira, "Testing the robustness of controllers for self-adaptive systems," *Journal of the Brazilian Computer Society*, vol. 20, no. 1, pp. 1-14, 23 January 2014.

[26] A. v. Lamsweerde, "Formal Specification: A Roadmap," in *Proceedings of the Conference on The Future of Software Engineering, ICSE '00*, Limerick, Ireland, 2000.

[27] A. M. a. A. K. Misra, "Formal Aspects of Specification and Validation of Dynamic Adaptive System by Analyzing Execution Traces," in *2011 Eighth IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, Las Vegas, Nevada, 2011.

[28] Uppsala Universitet; Aalborg University, "Uppaal," [Online]. Available: http://www.uppaal.org/.

[29] J. a. C. B. H. C. Zhang, "Model-based Development of Dynamically Adaptive Software," in *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*, Shanghai, China, 2006.

[30] F. D. a. A. A. D. M. Zhou, "A hybrid methodology for synthesis of Petri net models for manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 350-361, June 1992.

[31] Stanford University, "Temporal Logic," Stanford Encyclopedia of Philosophy, [Online]. Available: http://plato.stanford.edu/entries/logic-temporal/#LinTimTemLogLTL.

[32] C. A. R. Hoare, "Communicating Sequential Processes," *Commun. ACM*, vol. 21, no. 8, pp. 666-677, August 1978.

[33] E. Fredericks, B. DeVries and B. Cheng, "Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2014.

Rosalia Edith Caya Carhuanina was born in Lima, Peru. She received the degree of BsC (Bachelor in Science and Engineering) with major in Computer Engineer, and *the* professional degree of Computer and Systems Engineer from the Pontifical Catholic University of Peru (PUCP, Pontificia Universidad Católica del Perú) at Lima in 2009. Between the years 2007 and 2011, she participated of the activities at the Artificial Intelligence Research Group at the Computer Engineering Department of the same university. She holds a MsC degree in Electric Engineering, with major in the field of Computer Engineering from the University of São Paulo (USP) at São Paulo, Brazil. She is currently a PhD student developing her research in the Laboratory of Adaptive Languages and Techniques (LTA). Her main areas of research are: Artificial Intelligence, Natural Language Processing, Human-Machine Interaction and Adaptive Technology.

João José Neto graduated in Electrical Engineering (1971), Master in Electrical Engineering (1975), PhD in Electrical Engineering (1980) and Professor (1993) at the Polytechnic School of the University of São Paulo. Currently, he is an associate professor at the Polytechnic School of the University of São Paulo and coordinates the LTA - Laboratory of Languages and Adaptive Technology of PCS - Department of Computer Engineering and Digital Systems of EPUSP. He has experience in the area of Computer Science, with emphasis on the Fundamentals of Computer Engineering. He works mainly on the following topics: adaptive devices, adaptive technology, adaptive automata, and in their applications to Computer Engineering, particularly in adaptive decision making systems, analysis and processing of natural languages, construction of compilers, robotics, computer-aided teaching, modeling of intelligent systems, automatic learning processes and inferences based on adaptive technology