Towards Run-Time Testing of Dynamic Adaptive Systems

Erik M. Fredericks, Andres J. Ramirez, and Betty H. C. Cheng
Michigan State University,
East Lansing, MI, 48823, USA
{freder99, ramir105, chengb}@cse.msu.edu

***Abstract*—It is challenging to design, develop, and validate a dynamically adaptive system (DAS) that satisfies requirements, particularly when requirements can change at run time. Testing at design time can help verify and validate that a DAS satisfies its specified requirements and constraints. While offline tests may demonstrate that a DAS is capable of satisfying its requirements before deployment, a DAS may encounter unanticipated system and environmental conditions that can prevent it from achieving its objectives. In working towards a requirements-aware DAS, this paper proposes run-time monitoring and adaptation of tests as another technique for evaluating whether a DAS satisfies, or is even capable of satisfying, its requirements given its current execution context. To this end, this paper motivates the need and identifies challenges for adaptively testing a DAS at run time, as well as suggests possible methods for leveraging offline testing techniques for verifying run-time behavior.**

I. Introduction

A dynamically adaptive system (DAS) must cope with changing system and environmental conditions [1]–[4]. Testing during design and implementation phases serves to verify and validate that a DAS satisfies its specification within a given set of expected operational contexts. Unfortunately, design-time test cases are often static in nature and may contain uncertainty due to partially informed decisions about the DAS's requirements and its expected execution environment [5], [6]. Moreover, the applicability of design-time test cases may become limited as a DAS self-reconfigures in response to changing requirements and environmental conditions. This paper presents a vision for addressing the assurance of a DAS with a run-time requirements-aware testing feedback loop that can maintain consistency between design-time test cases and the DAS as it evolves. In addition, the testing feedback loop includes the refinement of the initial tests as the DAS monitors itself and its execution environment.

It is practically impossible to identify at design-time every possible operational context that a DAS may encounter at run time [1], [4], [7], [8]. To address this concern, researchers have developed design-time techniques for testing adaptive systems [9]–[17], as well as numerous DAS-specific assurance technologies that leverage run-time monitoring information [11], [18]–[21]. While these approaches provide better coverage for testing the possible space of system configurations at design-time, they are typically restricted to evaluating whether a DAS satisfies requirements within specific operational contexts that may differ from actual run-time conditions. Moreover, test cases generated and evaluated by these approaches are likely to gradually lose relevance over time as the DAS adapts and the execution environment changes.

This exploratory paper posits that a DAS should be requirements-aware [4] and treat tests as first-class entities that can evolve as requirements change and/or self-reconfigurations are applied. Within the proposed vision, test evolution is a multidimensional objective that must adapt and safely execute test cases based on the current context of the DAS. Maintaining tests in a consistent state with an evolving DAS should enable the DAS to reuse tests not only to verify that it satisfies its specification at run time, but to also detect possible conditions that warrant adaptation. In addition, modeling tests as first-class and evolvable entities, coupled with a flexible and requirements-aware framework, should also enable a DAS to access and reuse a larger breadth of tools and techniques for coping with system and environmental uncertainty at run time.

This work introduces MAPE-T, a feedback loop for supplementing testing strategies with run-time capabilities based on the monitoring, analysis, planning, and execution (MAPE-K [3]) architecture for adaptive systems. In particular, this paper defines the objectives of each key process as they apply to the testing domain. We also identify challenges to be addressed when adapting and executing tests at run time, as well as possible enabling technologies for realizing this framework. As with MAPE-K, the MAPE-T monitoring process gathers information about the system and its execution environment. The analysis and planning processes use this information to select DAS components to test and how to adapt test case inputs and expected outputs, respectively. The execution process safely executes and evaluates the outcomes of tests to determine if adaptation is necessary. The MAPE-T loop is illustrated with a running example of a smart home system [8].

The remainder of this paper is organized as follows. Section 2 motivates the need for adapting and executing tests at run time. Next, Section 3 identifies key objectives and research challenges for each key process in the testing feedback loop. Finally, Section 4 summarizes the proposed adaptive and run-time testing framework.

978-1-4673-4401-2/13 © 2013 IEEE

SEAMS 2013, San Francisco, CA, USA

II. Motivation

This section motivates the need for adapting and executing test cases at run time as a means to provide run-time assurance in a DAS. Specifically, we introduce the smart home system and use it to illustrate challenges in maintaining test specifications consistent with an evolving adaptive system.

A. Smart Home

A smart home [8] provides assistance to a patient by continuously monitoring and diagnosing the patient's condition, including prescribed diet, body temperature, and target blood pressure and glucose levels. A network of servers and sensors can provide the necessary infrastructure of a smart home, and may be modeled as a DAS.

B. Run-Time Testing Challenges

Traditional testing methods provide assurance for software systems based on anticipated executing conditions. A DAS, however, often operates in changing and possibly adverse contexts that may invalidate its initial set of requirements and test specifications. To provide this assurance to a DAS, existing assurance techniques may need to be extended to handle run-time information at run time. The following points highlight key differences between traditional offline testing approaches versus what is required to test an adaptive system at run time, motivated by the need to provide a consistent smart home system.

Test Case Generation. Test cases explore how different ranges, boundaries, and combinations of input values affect the outputs of software modules with regards to their requirements. In general, traditional testing techniques treat inputs and expected outputs as fixed, static values throughout the testing process. However, the DAS, its requirements specification, and its environment can change and thereby cause input and expected output values to no longer be representative test cases. Currently, developers have to manually identify and reconcile divergences between the system, its requirements, execution environment, and test specification. Unfortunately, for an adaptive system this manual identification and reconciliation process may be open-ended. For example, with respect to the smart home, dynamic test case generation and refinement can occur when a new sensor is added into the system, changes to the patient's prescribed diet occur, or new regulations are enacted.

When to Test. Testing of non-adaptive systems tends to occur during the development cycle. While testing efforts may decrease once the software is released, testing can and often does continue to be performed whenever the system is modified or extended to ensure changes do not break existing functionality. Moreover, traditional post-deployment testing is usually conducted on an isolated instance of the system to prevent introducing errors into a production environment. In contrast, post-deployment testing of an adaptive system must be conducted at run time upon the live system, often in response to changes in its operational context. This implication requires developers to identify when tests can be executed upon a live system without adversely affecting its behavior. Within the smart home example, tests can be executed periodically, directly after test cases or components in the smart home are adapted, or when available resources support run-time testing. It may also be possible to isolate certain test cases to prevent failures from impacting live-system execution.

How to Test. Numerous established methods exist for testing software systems at design time [22], each verifying and validating the system through different means. For instance, at design time, oracles may determine the outcome of test cases, and various strategies such as black-box, white-box, equivalence classes, and boundary testing may be used to ensure software is sufficiently mature for release. Existing run-time techniques include regression testing [23], run-time verification [24], and search-based methods [25]. Each of these testing strategies impose different requirements upon the system under test, from the granularity of information available about the system to the resources required to execute the test strategy. Adaptive systems should leverage these testing strategies at run time. However, applying these techniques at run time upon a live DAS requires assurance that tests will not interfere with the expected functionality and behavior of the system. For example, in a smart home, run-time testing must preserve system resources required for basic functionality while also ensuring data, such as sensor data for critical patient monitoring, is continuously available and in a consistent, secure state.

Impact and Mitigation of Test Results. In traditional systems, failed tests show that the software system violates requirements and typically result in updating the codebase to fix the issue. Common repair strategies require developers to identify what triggered the test failure and then resolve the issue by generating a patch or fixing the bug, each of which may require software to be re-validated to provide a new release. A DAS, however, can leverage both monitoring information to diagnose problems and its ability to self-reconfigure at run time to achieve the same corrective objectives. For example, a smart home could use its monitoring information to evaluate its run-time operational context and determine if a test case failed because of a fault within the smart home or because a clinician or sensor provided an invalid or inaccurate value. Moreover, the smart home can reuse failed tests and monitoring data to identify specific components that either require adaptation or are unsuitable for the current context.

III. MAPE-T Framework

This section overviews the proposed MAPE-T loop, as shown in Figure 1. This loop is an extension to the standard MAPE-K [3], but revised to support run-time testing of a DAS. Testing with MAPE-T comprises four parts: monitoring, analyzing, planning, and executing. *Monitoring* observes system and environmental changes. *Analyzing* identifies and selects individual test cases to perform at run time. *Planning* adapts test inputs and outputs as needed and schedules run-time testing. *Executing* performs the scheduled test plan and

triggers any necessary adaptations, as well as adjusts any expected outcomes as needed.

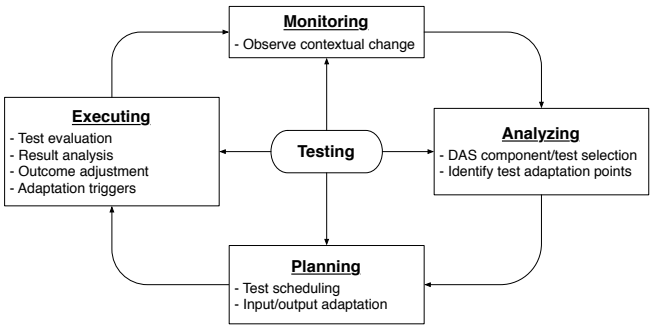

Fig. 1. MAPE-T loop

Next we present the objective, description, challenges, and enabling technologies for each element in the MAPE-T loop.

A. Monitoring

Objective. Observe and identify changes within the system and its environmental context.

Description. Monitoring enables a DAS to be aware of its operational context and how it can affect the DAS's ability to continuously satisfy requirements. Monitoring is both an internal and external process that performs introspection upon components in the DAS as well as measures observable properties about its execution environment, respectively. Monitoring information is often crucial for identifying possible causes for divergences between expected and actual test output.

Key Challenges. The inherent difficulties associated with monitoring within the context of MAPE-T are similar to the challenges encountered when monitoring within the context of MAPE-K [3]. In particular, it is challenging to determine: what properties of the adaptive system and its execution environment must be observed; what sensors, or sensor value aggregations, can measure desired properties; how often should monitoring data be gathered; and how does uncertainty, in the form of imperfect and possibly unreliable sensors, affect gathered values. Ultimately, the monitoring process must carefully balance these concerns such that it provides other processes in the MAPE-T architecture with useful monitoring information. For instance, a smart home needs to collect information about the patient from sensors that observe behavior, such as patient movement and food consumption. Information regarding the tests themselves should also be collected, such as the time of last execution.

Enabling Technologies. We envision that the MAPE-T monitoring process can reuse the monitoring infrastructure already required by the MAPE-K architecture. Both MAPE-T and MAPE-K depend on a monitoring infrastructure that provides information about a DAS's components and its execution environment. In addition to a context-aware monitoring process, however, the MAPE-T architecture must also periodically observe success and failure rates of tests at run time, as well as the consistency between test cases and the system and its environment. As a result, enabling technologies must support traditional MAPE-K requirements and the following: instrumentation of tests such that a DAS is continuously aware of their outcomes; and traceability between requirements and test cases to determine test relevance as the environment and requirements change as the system evolves in response.

B. Analyzing

Objective. Identify individual test cases to adapt and select specific tests to execute at run time.

Description. Test specifications are likely to become irrelevant as a DAS self-reconfigures in response to changes in its requirements and execution environment. The MAPE-T architecture ensures that test cases continue to be consistent and relevant even as the DAS evolves at run time. To achieve this objective, the analysis process must leverage monitoring information to identify divergences between tests and the DAS's context and configuration. Each divergence must be resolved and, consequently, corresponding components must be tested to ensure the system is capable of satisfying its requirements in its new operational context. This analysis process must carefully balance concerns between maximizing test coverage and minimizing testing overhead, as it is unlikely that all tests will require execution at any given point in time. Moreover, test selection must also address the possible explosion of artifacts within the state-space of the DAS that can result in an overwhelming volume of inputs and outputs for a DAS to evaluate and analyze.

Key Challenges. We envision that three key challenges must be addressed when designing and implementing the analysis process in MAPE-T. First, the analysis process must reliably identify important changes within the adaptive system and its execution environment. Second, the analysis process must also evaluate precisely how these changes impact the consistency and relevance of individual test cases. Lastly, because of adaptation, the analysis process must also continuously resolve and update traceability links between individual requirements, the DAS's components, and test cases. For example, a smart home may recognize that a temperature gauge requires replacement and will need to adapt requirements and test cases if the units differ between the old and new gauge. Specifically, the older may be in Fahrenheit, and the newer in Celsius.

Enabling Technologies. We envision that the MAPE-T analysis process will leverage techniques from formal methods, run-time requirement adaptation, and non-functional requirement quantification. Falcone *et al.* [9] proposed a formal approach for verifying component-based systems by automatically instrumenting systems via a set of Labeled Transition System (LTS) properties. LTS is a formal method for specifying a set of states and transitions, with each transition being labeled by an action. By formalizing a DAS's state space, it is possible to streamline and monitor the relevant artifacts within a system, and ensure that the tests being performed do not overextend the DAS.

Requirements are typically developed with respect to a target execution context and may become violated over time as that context changes. Sawyer *et al.* [4] proposed treating requirements as objects to be satisfied at run time in order

to counter environmental uncertainty within a DAS. In order to convert requirements to run-time entities, goal- and architecture-based reflection [7] may be applied. Once requirements have become objects, run-time reasoning [20] may be applied in order to consider tradeoffs, as the satisfaction of one requirement may impede the satisfaction of another. A similar approach can be taken for testing by converting test cases into run-time entities, allowing for reasoning to be applied to test entities.

Finally, it is beneficial to consider non-functional requirements when designing a system to be requirements-aware [4]. Non-functional requirements provide important constraints on how the system functions, typically with regards to performance, efficiency, or usability. Filieri [12] and Villegas [26] have introduced methods for deriving measurable values from non-functional requirements, paving the way for them to be incorporated as run-time entities for participation in the requirements-aware framework. Using identified metrics, non-functional requirements can be mapped into adaptive, run-time objects, enabling system functionality to evolve along with system execution.

C. Planning

Objective. Adapt expected inputs and output and schedule run-time testing.

Description. The planning aspect of the MAPE-T architecture defines a test plan for run-time DAS testing with three key considerations. First, test execution is scheduled in such a manner that testing does not adversely affect DAS performance. Second, test case parameters are created and maintained in order to ensure proper test specification coverage. Third, adaptation points are defined within test case parameters to provide plasticity in inputs and expected outputs, thereby allowing test case parameters to evolve in order to better reflect the system and environmental contexts in which the DAS executes.

Key Challenges. Reliably measuring adaptation costs and minimizing side effects is an overarching challenge when scheduling tests at run time. Ensuring that test case execution does not interfere or delay required adaptations is of utmost importance. As a result, a DAS must ensure that resources are not overextended by performing run-time testing. Likewise, methods for automated detection of unused processor cycles must be considered as well. The planning process must also define triggering conditions for the evolution of test inputs and outputs, and decide if the test cases themselves will drive adaptation, or if the results of testing will be simply handed off to a decision engine for further processing. Furthermore, modifying tests to support the inclusion of run-time monitoring information must also be acknowledged. For example, a smart home can schedule intensive self-tests during a patient's sleep cycle as resources will typically be more readily available at that time. Analysis of test results may also be performed in order to determine if adaptations are necessary to test cases.

Enabling Technologies. Continuous testing actively performs tests on a system under development and must consider the setup and maintenance of test parameters, test scheduling, and the decisions on which subset of test cases should be executed. Saff and Ernst studied the effects of continuous testing by using regression tests to aid software developers as they write code [27], [28]. In their approach, spare processor cycles were used to actively run regression test cases and verify that software modifications continue to satisfy requirements. Nguyen *et al.* [11] applied a similar concept to distributed agent-based systems. In their approach, a dedicated and automated testing entity coordinates the testing of other system entities. Specifically, testing is performed by two types of agents: monitoring and testing. The monitoring agent reports faults detected within the system and the testing agent generates test suites to be executed by other entities in the system, thereby providing continuous testing throughout the lifetime of the system.

An initial suite of test parameters can provide a suitable starting point for run-time testing, however as a DAS adapts, so too must its test cases. To address this concern, the planning process can supplement the initial suite of parameters with run-time monitored values [29] in order to accurately test the conditions that the DAS is experiencing during execution. Search-based techniques may also be leveraged to explore optimal test suite parameter configurations as well.

The use of search-based techniques to provide adaptive test case generation can help to mitigate this challenge. For instance, machine learning techniques using probabilistic methods [15], [30]–[32] or evolutionary computation-based techniques for exploring points of failure [16] could be considered as well.

Another dimension of run-time requirements satisfaction can be the use of models at run time. Baresi *et al.* [33] discussed methods for composition validation at run time via an assertion language that can describe both functional and non-functional properties. Morin *et al.* [34] proposed an approach that considers a DAS to be a dynamic software product line, with adaptations represented as differing configurations of software product lines (SPL). The goal of SPL design is to share a common codebase with varying software products. Tradeoffs in balancing object satisfaction are considered at run time through model reasoning, providing a DAS with the capability to continue satisfaction of its goals in the face of adversity. Methods for reasoning over models at run time have been proposed by Epifani *et al.* [35] as well.

Scheduling test cases transparently requires carefully balancing two competing concerns: maximizing the utilization of system resources and minimizing adverse side effects, such as interference with system behavior. As with continuous testing methods [27], [28], it may be possible for a run-time management framework to address the first concern by executing test cases whenever spare computing cycles become available in a DAS. Likewise, although resource contention tends to be more of a problem in resource constrained systems, a DAS can also leverage selective testing strategies to filter extraneous tests that will consume cycles and energy without focusing on areas that might cause the system to adapt. For

example, these two concerns might restrict a DAS operating in a low-power mode to avoid executing test cases that may violate its current power optimization scheme.

Deciding on the subset of tests to execute at run time is essentially a multidimensional optimization problem, incorporating considerations for performance, scheduling, and adaptation penalties that may occur. Assuming that these objectives have been quantified, it may be possible to use algorithms specifically designed for multiobjective optimization, such as NSGA-II [36].

D. Executing

Objective. Perform the test plan and analyze results, adjust expected outcomes, and trigger adaptations within the DAS, its requirements, or the testing framework.

Description. The execution phase of MAPE-T is concerned with the execution of and response to the scheduled testing. As testing is performed, the results must be analyzed and any necessary adjustments added to run-time entities, specifically their expected inputs and outputs. If necessary, adaptations may be triggered during this phase as well. In addition, MAPE-T provides for reconfiguring the requirements and testing frameworks.

Key Challenges. In order to properly execute and evaluate tests at run time, both requirements and tests must be converted into first-class entities within the system. This transformation is not a trivial task, as it can be quite difficult to distill requirements, especially non-functional requirements, into quantifiable entities. The same issue arises when converting test cases into adaptable entities. Another challenge occurs in deciding how to limit the adaptability of these entities, as providing excessive plasticity can cause adverse effects for DAS execution. Lastly, adapting the acceptance rate for a test suite will be a challenge as well. For instance, execution of self-tests that verify proper function of a motion sensor within a smart home must take into account time of day. If the patient is asleep, then naturally any tests that check a motion sensor will fail, and these results must be filtered from consideration.

Enabling Technologies. As the execution context of a DAS evolves, it is possible that system requirements must change in order to accommodate the new environment. Souza *et al.* proposed eliciting *awareness requirements* to be treated as run-time entities [37] in order to monitor violations over time. Once a predetermined amount of violations occur, parameters that have been previously identified as important to that requirement may be adapted in order to improve its chances for satisfaction. Their approach can be modeled as a set of operations over a goal model, providing the capability to be modeled at run time. Similar approaches also make use of requirements as run-time entities [4], [7], and provide capabilities for requirements reasoning. These *requirements-aware* systems are a step towards handling uncertainty in a systematic manner. With the addition of languages such as RELAX [8] and FLAGS [38], it is possible to make requirements less rigid and brittle through fuzzy logic.

While run-time testing has yet to fully gain traction in live systems, it has been successfully applied within hardware-based adaptive systems [39], typically through the use of field-programmable gate arrays. Run-time testing methods within hardware systems are concerned with component and system execution failures, and must reconfigure the system accordingly [40], [41]. These systems have the additional limitation of using embedded CPUs, and must therefore consider power- and memory-optimizations in their design. Through monitoring, redundancy, and a well-defined procedure for recovery, Subhasish *et al.* [39] have provided a framework for run-time testing and reconfiguration of a hardware DAS. This approach allows a system to continuously function without external intervention, however it can be limited by the severity and recurrence of faults. Techniques from the hardware domain may be reused in software, such as using redundant controllers for monitoring and correcting faults as proposed by Nguyen *et al.* [11].

IV. Conclusion

This exploratory paper proposed a vision for MAPE-T, a requirements- and test-aware feedback loop architecture focused on executing tests at run time to verify that an adaptive system satisfies its requirements. MAPE-T is based on the monitoring, analysis, planning, and execution processes as defined in MAPE-K, but tailored for testing purposes. Several key challenges have been identified in realizing a MAPE-T architecture, including assurance that run-time testing does not strain or overextend a DAS, provisions for reliable traceability between test cases and requirements, safe adaptation triggering in an evolving environment, and specification of adaptation constraints. In order to assist in the mitigation of uncertainties within system and environmental contexts, we have proposed adopting and extending traditional testing strategies for run-time application within a DAS.

Acknowledgment

This work has been supported in part by NSF grants CCF-0854931, CCF-0750787, CCF-0820220, DBI-0939454, Army Research Office grant W911NF-08-1-0495, and Ford Motor Company. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Army, or other research sponsors.

References

[1] B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, and et al., "Software engineering for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems*, B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

[2] P. McKinley, S. Sadjadi, E. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56 – 64, july 2004.

[3] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41 – 50, jan 2003.

[4] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 27 2010-oct. 1 2010, pp. 95 –103.

[5] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009.

[6] K. Welsh and P. Sawyer, "Understanding the scope of uncertainty in dynamically adaptive systems," in *Proceedings of the Sixteenth International Working Conference on Requirements Engineering: Foundation for Software Quality*, vol. 6182. Springer, 2010, pp. 2–16.

[7] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, vol. 2, may 2010, pp. 199 –202.

[8] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 468–483.

[9] Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem, "Runtime verification of component-based systems," in *Proceedings of the 9th international conference on Software engineering and formal methods*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 204–220.

[10] D. C. Nguyen, A. Perini, and P. Tonella, "A goal-oriented software testing methodology," in *Proceedings of the 8th international conference on Agent-oriented software engineering VIII*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 58–72.

[11] C. D. Nguyen, A. Perini, P. Tonella, and F. B. Kessler, "Automated continuous testing of multiagent systems," in *The Fifth European Workshop on Multi-Agent Systems (EUMAS)*, 2007.

[12] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, vol. 24, pp. 163–186, 2012.

[13] D. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. Iyer, "Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors," in *Computer Performance and Dependability Symposium, 2000.*, 2000, pp. 91 –100.

[14] A. J. Ramirez, E. M. Fredericks, A. C. Jensen, and B. H. C. Cheng, "Automatically relaxing a goal model to cope with uncertainty," in *Search Based Software Engineering*, G. Fraser and J. Teixeira de Souza, Eds. Springer Berlin Heidelberg, 2012, vol. 7515, pp. 198–212.

[15] J. Camara and R. de Lemos, "Evaluation of resilience in self-adaptive systems using probabilistic model-checking," in *Software Engineering for Adaptive and Self-Managing Systems.*, june 2012, pp. 53 –62.

[16] A. Ramirez, A. Jensen, B. H. C. Cheng, and D. Knoester, "Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, nov. 2011, pp. 568 –571.

[17] A. Ramirez and B. H. C. Cheng, "Verifying and analyzing adaptive logic through uml state models," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, april 2008, pp. 529 –532.

[18] C. Ghezzi, "Adaptive software needs continuous verification," in *Software Engineering and Formal Methods (SEFM), 2010 8th IEEE International Conference on*, sept. 2010, pp. 3 –4.

[19] H. J. Goldsby, B. H. C. Cheng, and J. Zhang, "Models in software engineering," in *Models in Software Engineering*, H. Giese, Ed. Berlin, Heidelberg: Springer-Verlag, 2008, ch. AMOEBA-RT: Run-Time Verification of Adaptive Software, pp. 212–224.

[20] N. Qureshi, S. Liaskos, and A. Perini, "Reasoning about adaptive requirements for self-adaptive systems at runtime," in *Proceedings of the 2011 International Workshop on Requirements at Run Time*, aug. 2011, pp. 16 –22.

[21] O. Wei, A. Gurfinkel, and M. Chechik, "On the consistency, expressiveness, and precision of partial modeling formalisms," *Information and Computation*, vol. 209, no. 1, pp. 20–47, 2011.

[22] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *FOSE*, 2007, pp. 85–103.

[23] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, 2010.

[24] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293 – 303, 2009.

[25] P. McMinn, "Search-based software test data generation: a survey: Research articles," *Softw. Test. Verif. Reliab.*, vol. 14, no. 2, pp. 105–156, Jun. 2004.

[26] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Waikiki, Honolulu, Hawaii, USA: ACM, 2011, pp. 80–89.

[27] D. Saff and M. D. Ernst, "An experimental evaluation of continuous testing during development," in *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*. Boston, Massachusetts, USA: ACM, 2004, pp. 76–85.

[28] ——, "Reducing wasted development time via continuous testing," in *Proceedings of the 14th International Symposium on Software Reliability Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 281–.

[29] K. Mahbub and G. Spanoudakis, "A framework for requirements monitoring of service based systems," in *Proceedings of the 2nd international conference on Service oriented computing*. New York, NY, USA: ACM, 2004, pp. 84–93.

[30] E. Pavese, V. Braberman, and S. Uchitel, "Automated reliability estimation over partial systematic explorations," in *To appear in the 2013 International Conference on Software Engineering*, 2013.

[31] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *Proceedings of the 33rd International Conference on Software Engineering*. Waikiki, Honolulu, Hawaii, USA: ACM, 2011, pp. 341–350.

[32] A. Filieri, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Conquering complexity via seamless integration of design-time and run-time verification," in *Conquering Complexity*, M. Hinchey and L. Coyle, Eds. Springer London, 2012, pp. 253–275.

[33] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of web service compositions," *Software, IET*, vol. 1, no. 6, pp. 219–232, 2007.

[34] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@ run.time to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44 –51, oct. 2009.

[35] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 111–121.

[36] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182 –197, apr 2002.

[37] V. Souza, A. Lapouchnian, and J. Mylopoulos, "(requirement) evolution requirements for adaptive systems," in *Software Engineering for Adaptive and Self-Managing Systems, 2012 ICSE Workshop on*, june 2012, pp. 155 –164.

[38] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 27 2010-oct. 1 2010, pp. 125 –134.

[39] S. Mitra, W.-J. Huang, N. Saxena, S.-Y. Yu, and E. McCluskey, "Reconfigurable architecture for autonomous self-repair," *Design Test of Computers, IEEE*, vol. 21, no. 3, pp. 228 – 240, may-june 2004.

[40] N. R. Saxena, S. Fernandez-Gomez, W.-J. Huang, S. Mitra, S.-Y. Yu, and E. J. McCluskey, "Dependable computing and online testing in adaptive and configurable systems," *IEEE Des. Test*, vol. 17, no. 1, pp. 29–41, Jan. 2000.

[41] R. DeMara and K. Zhang, "Autonomous fpga fault handling through competitive runtime reconfiguration," in *Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on*, june-1 july 2005, pp. 109 – 116.