Changing Software in a Changing World: How to Test in Presence of Variability, Adaptation and Evolution?

Antonia Bertolino[1] and Paola Inverardi[2]

[1] ISTI–CNR, Pisa, Italy
antonia.bertolino@isti.cnr.it
[2] University of L'Aquila, L'Aquila, Italy
paola.inverardi@univaq.it

Abstract. Modern software-intensive and pervasive systems need to be able to manage different requirements of variability, adaptation and evolution. The latter are surely related properties, all bringing uncertainty, but covering different aspects and requiring different approaches. Testing of such systems introduces many challenges: variability would require the test of too many configurations and variants well beyond feasibility; adaptation should be based on context-aware testing over many predictable or even unpredictable scenarios; evolution would entail testing a system for which the reference model has become out-of-date. It is evident how current testing approaches are not adequate for such types of systems. We make a brief overview of testing challenges for changing software in a changing world, and hint at some promising approaches, arguing how these would need to be part of a holistic validation approach that can handle uncertainty.

Keywords: Adaptation and evolution · Context-aware software · Software variability · Testing changing software

1 Introduction

Nowadays software is ubiquitous and governs our lives interacting with smart objects and other software systems that increasingly pervade our surrounding environment. We got used to require that software -be it working from our portable device or in the public front office we ask or in a newly bought home appliance- reacts promptly to satisfy our requests. We even expect that it is capable to face unforeseen circumstances and events or even more that it can anticipate our future needs.

Under pressure of tackling continuous changes that can potentially occur in many ways, software systems themselves change continuously. For instance, they can be Systems-of-Systems (SoS) emerging from the on-the-fly dynamic composition of services, or they can perform self-repair after a problem, or their components can be substituted at runtime.

Consequently, a traditional view of the software lifecycle as involving three main stages: specification, coding (even if by model-driven transformations), and testing is not adequate anymore.

In the software engineering literature, the problem of handling change has been addressed along different research threads. One thread regards software product line (PL) research. In the past two decades huge progress has been done with methodologies and tools that can model and manage variants of products within one family. By adopting a PL approach, developers can a-priori define points of variations and acceptable alternative solutions for differing instantiations of a broad software architecture. Such notion of change, which is referred to as *variability*, has been a main research focus of Stefania Gnesi for several years: in her work she has shown that variability can be formally described [2], or even extracted a posteriori from the requirements [16]. More recently, the notion of Dynamic Software Product Line (DSPL) has emerged [23], which delays the decision of variations to runtime and extends the scope of variability.

On another thread, researchers have investigated approaches to engineer software systems that can adapt to intervening events and situations. Such approaches generally adopt variations of the MAPE (Monitor, Analyze, Plan, Execute) model [24], i.e., the system needs to sense the context and be able to react accordingly. Self-*adaptation* refers to systems that autonomously can decide how to change themselves so to ensure continuous service.

Similar to adaptation is the notion of *evolution*: whereby adaptation is generally referred to as a reactive change triggered by changes in the external world, evolution is rather conceived as a proactive attitude towards change. An evolving system aims at continuously improving itself and providing enhanced services. It must be able to change its goals and behaviour so to provide a service that can satisfy novel requirements. Indeed, placing change in the center of the software process is recognized as the only way to prevent software aging [33].

Although focusing on different facets of change, the notions of variability, adaptation and evolution share several challenges and requirements. They all make it difficult for a software developer to analyse a system and take decisions on it. By stretching somehow the term, in the context of this work we will refer to this difficulty in understanding or predicting a system behaviour as *uncertainty*. We use this term to imply that we cannot know what to expect from a system, because it can take too many possible configurations (variability), or can adapt to context (adaption), or can change its goals (evolution). Indeed, following [21], uncertainty can be defined as the difference between the amount of information required to perform a task and the amount of information already possessed.

The task we are interested here is validation of systems that *change*. Systems for which at the moment of validation complete information is lacking either because it is unknown or because it is too large. In fact, we started by saying that software systems are pervasive and thus we cannot underestimate the need to ensure a reliable behaviour, notwithstanding changes. However, what does it mean to test a system that exposes variability, adaptation or evolution, and which approaches can be applied are still open research questions.

In this opinion paper, we first overview current views of variability, adaptation and evolution, including their shared definitions and most common approaches (Section 2). Then, we discuss the challenges descending for testing such type of systems and hint at promising approaches (Section 3). Conclusions and possible research directions conclude the paper (Section 4).

2 Many dimensions of change

As discussed in the introduction modern systems are subject to a number of potential changes during their life time. Those changes cannot always be anticipated or it might not be convenient to anticipate all of them. This introduces levels of uncertainty in the predictable behavior of the system. In the following we analyze the three dimensions of changes we have earlier introduced to understand what are the potential sources of uncertainty.

2.1 Changing software

Software needs to be able to change. *Variability* is the dimension that characterizes the software that shall encompass the possibility of designing alternatives in the systems, that will be solved only before execution, either statically via a configuration step, or dynamically by providing the necessary information. No matter how variability is resolved, it introduces in the validation step of the development process the need to deal with the system's strong degree of non-deterministic behaviors. When explicitly introduced in the software life cycle [25, 3], variability can help reducing the uncertainty by constraining the behavioral analysis into well defined boundaries. However such boundaries can still permit an extremely large search space of potential configurations, like it may happen in the Software Product Line context, thus retaining in practice a degree of uncertainty in the final system behavior. In the past years an extensive research thread contributed by Stefania Gnesi and co-authors has proposed different behavioral expressive models able to compactly represent such search spaces [6, 5, 40], however verification of such systems has not yet reached the maturity of being routinely used in a development process.

2.2 Changing world

Software needs to be sensitive to the changes that the world around it encompasses. Both adaptation and evolution respond, in different ways, to this need. *Adaptation* refers to the ability of a software system to react in presence of changes of context that may compromise the system behavior, either qualitatively or quantitatively. It is a change that the system needs to undergo not to compromise the compliance of its behavior with respect to the requirements [26]. It typically appears concerning quantitative properties, e.g., degradation of performance due to unexpected high workload. It is associated with the so called self-* properties and autonomic systems [30] and, as already mentioned, it is often implemented through possibly multiple feedback loops. Adaptation may let the system acquire completely new behaviors not foreseen at design time, which is even more evident nowadays with the increasing adoption of learning techniques. How to accomplish adaptation by maintaining system's correctness is a challenge that has received a large deal of attention in the research community and has also motivated the need to move part of the development artifacts at run time (e.g., models at run time) [31].

Evolution has been traditionally the last step in the software life cycle coupled with maintenance. Traditionally it was considered for long living systems that might need to change in order to meet new emerging requirements from users, operating system platform producers, machine changes. In such context the pace of change allowed to integrate the evolution step in the ordinary software life cycle with relatively little effort. For example, traceability issues all along the development phases were required [15] as well as regression test emerged in the validation step. Modern software systems are instead experiencing a fast twist in pace due to the speed of changes both in terms of user expectation and in terms of technological upgrades. In this respect the difficulties of evolution are exacerbated. One main issue concerns the problem of keeping the consistency among the different models of a system (i.e., co-evolution), notably requirements, architecture and code implementation [32].

It appears evident that for modern software systems, validation in the presence of variability, adaptation and evolution needs to take into account a certain degree of uncertainty as anticipated in [20]. Referring to the introduced notion of uncertainty, this means that at the moment these systems are validated, developers do not possess the (complete) information about the systems that the validation step may require. In the following we will discuss how the change dimensions impact on testing and consider some research challenges we foresee in validating software in presence of uncertainty due to changes.

3 Testing software that changes

In this section we reflect on the implications brought by change on the software testing discipline. We start by sketching a theoretical framework on which the aims and foundations of software testing are laid. Then we analyse the challenges posed by uncertainties deriving from each of the three kinds of change discussed in the previous section. We conclude by pointing at some promising directions emerging from the literature for addressing the challenges.

3.1 Software testing foundations in light of change

As defined by Bertolino in [7], software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the specified expected behavior.

This definition highlights the main concerns in software testing, in particular that we need a strategy to select a feasible set of test inputs and that we must be able to compare the test output against an expected behaviour, *a.k.a.* the *oracle* problem [4].

In the early 80's, a framework providing a theoretical foundation of software testing was proposed by Gourlay [22]. The framework established a mathematical relation among sets of specifications S, programs P and tests T, and defined the oracle as an *ok* predicate over a test $t \in T$, a specification $s \in S$, and a program $p \in P$. More formally, Gourlay's framework defined a theoretical predicate $corr(p, s)$ over specifications and programs implying that a program p is correct with respect to a specification s, and postulated that $\forall p \in P, \forall s \in S, \forall t \in T, corr(p, s) \Rightarrow ok(t, p, s)$.

More recently, Staats and coauthors [38] revisited Gourlay's framework, and introduced a set O of test oracles (in place of the unique oracle *ok*), whereby a test oracle o is a predicate over programs and tests; they defined a new $corr_t$ predicate over tests, program and specifications that holds if and only if when running test t, specification s holds for program p.

However, neither Bertolino's definition for software testing, nor Staats and coauthors' revisited version of Gourlay's theoretical framework consider explicitly that a program, and/or its input domain and/or its expected behaviour (i.e., oracle), can change and how the derived uncertainty can impact testing validity and effectiveness.

Indeed, *variability, adaptation and evolution clearly affect the notion of testing*, and we claim that in presence of change the theoretical framework for testing should be revised to cope with the uncertainty they bring.

In presence of variability, not only we need to select a finite set of test cases, but also we need to select a set of configurations among those implied by the variation points.

In presence of adaptation, a test case should include a test input but also the context in which the test is executed, and the program itself becomes a function of the context. As a consequence, also the very concept of correctness of a program with respect to a specification may change depending on context.

In presence of evolution, again the correctness relation between a program and the specification becomes relative, in this case because specification can proactively change.

Therefore, we leave as a challenging task for future work a revision of testing theory as formulated in [22] and in [38] to take into account change and uncertainty.

3.2 Testing challenges ahead

In front of a rich literature addressing the design and management of changing systems, research on how such systems should be tested is still lacking. For example, focusing on adaptation, in 2009 Salehie and Tahvildari [35] affirmed that testing and assurance are probably the least focused phases, and there are only few works addressing this topic. Concerning variability, in 2014 Galster and coauthors observed that it is "*studied in all software engineering phases, but testing is underrepresented*" [19]. Fortunately today this situation seems to be changing, and several works appear addressing efficient approaches to variability testing, such as, e.g., [27, 1].

The testing challenges implied by change in the three forms that we distinguish have been studied in the literature generally along separate threads. It is rarely the case that the three dimensions of change have been considered in holistic way.

Concerning variability, this has been mostly addressed in the domain of software Product Lines. The systematic survey in [12] distinguishes two main research interests, namely the PL features and the PL products. Along the first one, testing aims at verifying all feature interactions by testing all variations across all dimensions. The second one concerns the actual testing of the products members of a family. In both cases, the great challenge is to manage the huge number of potential test cases, which can increase exponentially with the PL features.

The testing challenges stemming from adaptation have been characterised by Siqueira and coauthors [37], who made a systematic survey of literature. They list several general challenges, among which: the exponential growth of the number of configurations to be tested; the difficulty of anticipating environment changes when testing on a large-scale multi-vendor system; the problem to keep traceability between the requirements and the test cases due to the changing characteristics; the arduousness of simulating realistic contexts and workloads due to unpredictability and unclear system boundaries.

Evolution is the dimension of change that has been more extensively addressed in the software testing literature, because it corresponds in a sense to the classical problem of regression testing. Strictly speaking, regression testing concerns the re-testing of previously tested software to verify that changes do not cause previously successfully passed test cases to fail. In recent work, the step of "test suite augmentation" within regression testing process is attracting more emphasis: it refers to creating new test cases specifically addressing the changed behaviour of the evolving software [36]. However, proposed test suite augmentation approaches are mostly code-based, and they do not scale up to consider the complexity of modern evolving systems. The challenges in testing evolving software include finding black-box approaches that can consider dependencies among concurrently running processes, as addressed for instance in [41], as well as dependencies from context changes, as described in [34].

Moreover, a challenge that we see as shared by all three types of change concerns the difficulty of setting an oracle, be it automated or even manual. If we accept that the software behaviour may change because of context adaptation, or evolution of requirements, how can we discern whether an observed behaviour that is not as we would have expected at a given moment is a failure, or is rather a correct deviation because of a change?

When a test is executed we need a way to decide whether a test is successful or fails. However, if we consider the testing of a changing software program P, one issue is that since the system has evolved or has assumed very different forms, we cannot have a readily available reference model to act as an oracle. Even assuming that an oracle is available, for example from a specification, we have to take into account that due to evolution the specification $Spec_t$ that was available at time t may become invalid in later time. So, if at time $t' > t$ we observe that P is not compliant with $Spec_t$, what can we deduce? Is it because the system has evolved (in good way) and hence we need to also evolve *Spec*? Or instead it is because there is a failure in P behaviour?

In other words, in presence of changing systems, when an observed behaviour is not compliant with the oracle, how can we decide whether it is for good (hence the *Spec* we referred to is obsolete and, e.g., a new specification should be mined [18]), or for bad (the system has evolved in unacceptable way or some failure has occurred)?

3.3 Promising testing techniques

Based on our overview of how the foundations of testing software that changes differ from those laid down for "traditional" testing, it is clear that we need to find completely novel approaches to testing software in light of the uncertainty brought by dynamic adaptation and evolution, and of the huge number of possible configurations to test due to variability.

In this section we overview some recent techniques that could be adapted to deal with change in all its three forms.

A natural approach to address the lack or obsolescence of models that can be referred as test oracle or even for test generation is that of mining the model from the program, in particular from the traces obtained by test executions. This is the idea outlined in the *anti-model based testing* proposal by Bertolino et al. [10], even though at the time it was aimed at testing applications when a model is not available. Later on, the idea is further developed by Kanstrén et al [28], who used the term *observation-based modeling.*

Another promising research avenue is the one to identify so-called "core relatives" [39], which are defined as *pieces of code exhibiting "similar" behavior*, even though structurally different and producing different outputs. Such techniques could be usefully adapted in testing changes.

A well established approach for software testing within some specific domain where deriving an oracle is extremely difficult is *metamorphic testing*: this approach was introduced in the late 90's [13], and has been applied in several contexts and to solve various problems [14]. Metamorphic testing is based on a set of properties that must hold between different executions of the tested system: these necessary properties are called the metamorphic relations. Therefore, even if we do not know what is the expected correct result for an execution, we can compare the outputs across different executions against the expected relations, and detect possible failures when the properties are not fulfilled.

We see several interesting ways in which metamorphic testing naturally applies to the case of testing changing software. For example, where sensible, we could define a set of necessary relations to be maintained across adaptation or evolution, and perform metamorphic testing based on such relations to verify if the software continues to keep the necessary properties. In presence of variability, metamorphic relations could be used to express common features within a family of products.

Yet another potential direction to explore could be to raise the level of abstraction at which the testing is conducted, and perform the testing of the model and not of the implementation, as proposed by Briand and coauthors [11]. The authors proposed to deal with uncertainty by associating appropriate probability distributions to the model elements. A more detailed and complex approach should be conceived to be able to consider all dimensions of variability, adaptation and evolution.

From the field of deep learning systems, we could also adopt the concept of *surprise adequacy* testing [29]: the authors propose that for testing these systems where we cannot know the exact correct outputs, we could expect that what we observe in operation can be different from what we observed during training, but not too much different: they say that the "surprise" we observe must not be too big. We could apply a similar concept for testing in operation a changing system: we establish some "surprise" distances we can admit in operation, and test accordingly.

Inspired by the Proteus framework by Fredericks and Cheng [17], a test platform for changing software should support the adaptive generation of test plans including a core set of test cases that must be satisfied even after change, and an additional set of test cases aiming at testing possible adaptations/evolutions. The former should be based on invariant properties that could be tested applying metamorphic testing between source test cases before change, and follow up test cases after change. The latter would require test suite augmentation: we could perform observation-based testing and assess the mined model against a defined degree of surprise, i.e., distance we can tolerate.

4 Perspectives for research

As we discussed, testing software in presence of change opens a number of challenging research directions. Notwithstanding, testing remains indispensable, as for such dynamic systems we cannot assume the availability of valid reference models or test suites. On the contrary, we have to deal with uncertainty and the only fact is the behaviour we observe. Because of this, we cannot adopt traditional model-based testing techniques, and need to adapt approaches for anti-model based testing or observation-based modeling or model testing.

An appropriate approach for testing in presence of change should handle change in its three identified dimensions, which should be considered in combination, scaling up further the complexity of the task.

We have overviewed some promising research directions for testing changing software in a changing world. As we cannot rely on the availability of an oracle, we have suggested to adapt metamorphic testing principles for testing changing software but still guaranteeing a core set of invariant properties. In combination we also suggested the opportunity to adapt a notion of surprise-based testing for test suite augmentation.

We have only scraped the surface of the tackled problem, though: for example, we did not discuss when and how testing should occur. For sure monitoring software behaviour is essential, but what would be a proper trigger for moving from passive testing, to proactive? We would need to introduce proper test governance policies [9].

Moreover, we did not discuss the challenges behind reproducing the context of a changing world within which the testing should occur. As this could be too costly or even infeasible, several authors have suggested to perform the testing in production (e.g., [8]), but this poses many new challenges.

For sure many other challenges exist and many new research avenues could be identified. The aim of this paper was not that of providing an exhaustive survey of issues and opportunities, but rather that of depicting a preliminary understanding of the problem difficulties and outlining promising directions for tackling them.

Acknowledgements

This work has been partially supported by the GAUSS national research project (MIUR - PRIN 2015, Contract 2015KWREMX).

References

1. Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., Saake, G.: Effective product-line testing using similarity-based product prioritization. Software & Systems Modeling **18**(1), 499–521 (2019)
2. Asirelli, P., Ter Beek, M.H., Gnesi, S., Fantechi, A.: Formal description of variability in product families. In: 2011 15th International Software Product Line Conference. pp. 130–139. IEEE (2011)
3. Autili, M., Benedetto, P.D., Inverardi, P.: A hybrid approach for resource-based comparison of adaptable java applications. Sci. Comput. Program. **78**(8), 987–1009 (2013). https://doi.org/10.1016/j.scico.2012.01.005, https://doi.org/10.1016/j.scico.2012.01.005
4. Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: The oracle problem in software testing: A survey. IEEE transactions on software engineering **41**(5), 507–525 (2014)
5. ter Beek, M.H., Damiani, F., Gnesi, S., Mazzanti, F., Paolini, L.: On the expressiveness of modal transition systems with variability constraints. Sci. Comput. Program. **169**, 1–17 (2019). https://doi.org/10.1016/j.scico.2018.09.006, https://doi.org/10.1016/j.scico.2018.09.006
6. Beohar, H., Varshosaz, M., Mousavi, M.R.: Basic behavioral models for software product lines: Expressiveness and testing pre-orders. Sci. Comput. Program. **123**, 42–60 (2016). https://doi.org/10.1016/j.scico.2015.06.005, https://doi.org/10.1016/j.scico.2015.06.005

7. Bertolino, A.: Software testing. In: P. Bourque, R.D. (ed.) SWEBOK Guide to the Software Engineering Body of Knowledge Trial Version, chap. 5, pp. 69–86. IEEE CS, Los Alamitos, CA (2001)
8. Bertolino, A., Angelis, G.D., Kellomaki, S., Polini, A.: Enhancing service federation trustworthiness through online testing. IEEE Computer **45**(1), 66–72 (2012). https://doi.org/10.1109/MC.2011.227, https://doi.org/10.1109/MC.2011.227
9. Bertolino, A., Polini, A.: Soa test governance: Enabling service integration testing across organization and technology borders. In: 2009 International Conference on Software Testing, Verification, and Validation Workshops. pp. 277–286. IEEE (2009)
10. Bertolino, A., Polini, A., Inverardi, P., Muccini, H.: Towards anti-model-based testing. In: In Proc. DSN 2004 (Ext. abstract. pp. 124–125 (2004)
11. Briand, L., Nejati, S., Sabetzadeh, M., Bianculli, D.: Testing the untestable: Model testing of complex software-intensive systems. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 789–792. ICSE '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2889160.2889212, http://doi.acm.org/10.1145/2889160.2889212
12. do Carmo Machado, I., Mcgregor, J.D., Cavalcanti, Y.C., De Almeida, E.S.: On strategies for testing software product lines: A systematic literature review. Information and Software Technology **56**(10), 1183–1199 (2014)
13. Chen, T.Y., Cheung, S.C., Yiu, S.M.: Metamorphic testing: a new approach for generating next test cases. Tech. rep., Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong (1998)
14. Chen, T.Y., Kuo, F.C., Liu, H., Poon, P.L., Towey, D., Tse, T.H., Zhou, Z.Q.: Metamorphic testing: A review of challenges and opportunities. ACM Comput. Surv. **51**(1), 4:1–4:27 (Jan 2018). https://doi.org/10.1145/3143561, http://doi.acm.org/10.1145/3143561
15. Cleland-Huang, J., Gotel, O., Hayes, J.H., Mäder, P., Zisman, A.: Software traceability: trends and future directions. In: Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014. pp. 55–69 (2014). https://doi.org/10.1145/2593882.2593891, https://doi.org/10.1145/2593882.2593891
16. Fantechi, A., Ferrari, A., Gnesi, S., Semini, L.: Requirement engineering of software product lines: Extracting variability using nlp. In: 2018 IEEE 26th International Requirements Engineering Conference (RE). pp. 418–423. IEEE (2018)
17. Fredericks, E.M., Cheng, B.H.: Automated generation of adaptive test plans for self-adaptive systems. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 157–168. IEEE Press (2015)
18. Gabel, M., Su, Z.: Testing mined specifications. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. pp. 4:1–4:11. FSE '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2393596.2393598, http://doi.acm.org/10.1145/2393596.2393598
19. Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P.: Variability in software systemsa systematic literature review. IEEE Transactions on Software Engineering **40**(3), 282–306 (2014)
20. Garlan, D.: Software engineering in an uncertain world. In: Proceedings of the FSE/SDP workshop on Future of software engineering research. pp. 125–128. ACM (2010)

21. Giese, H., Bencomo, N., Pasquale, L., Ramirez, A.J., Inverardi, P., Wätzoldt, S., Clarke, S.: Living with uncertainty in the age of runtime models. In: Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.) Models@run.time: Foundations, Applications, and Roadmaps. pp. 47–100. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-08915-7_3
22. Gourlay, J.S.: A mathematical framework for the investigation of testing. IEEE Transactions on software engineering (6), 686–709 (1983)
23. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. Computer **41**(4), 93–95 (2008)
24. IBM White Paper: An architectural blueprint for autonomic computing (2006)
25. Inverardi, P., Mazzanti, F.: Experimenting with dynamic linking with ada. Softw., Pract. Exper. **23**(1), 1–14 (1993). https://doi.org/10.1002/spe.4380230102, https://doi.org/10.1002/spe.4380230102
26. Inverardi, P., Tivoli, M.: The future of software: Adaptation and dependability. In: Software Engineering, International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures. pp. 1–31 (2008). https://doi.org/10.1007/978-3-540-95888-8_1, https://doi.org/10.1007/978-3-540-95888-8_1
27. Jakubovski Filho, H.L., Ferreira, T.N., Vergilio, S.R.: Preference based multi-objective algorithms applied to the variability testing of software product lines. Journal of Systems and Software **151**, 194–209 (2019)
28. Kanstrén, T., Piel, E., Gross, H.G.: Observation-based modeling for model-based testing. Technical Report Series TUD-SERG-2009-012 (2009)
29. Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy. arXiv preprint arXiv:1808.08444 (2018)
30. Kounev, S., Lewis, P.R., Bellman, K.L., Bencomo, N., Cámara, J., Diaconescu, A., Esterle, L., Geihs, K., Giese, H., Götz, S., Inverardi, P., Kephart, J.O., Zisman, A.: The notion of self-aware computing. In: Self-Aware Computing Systems., pp. 3–16 (2017). https://doi.org/10.1007/978-3-319-47474-8_1, https://doi.org/10.1007/978-3-319-47474-8_1
31. de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., Andersson, J., Litoiu, M., Schmerl, B.R., Weyns, D., Baresi, L., Bencomo, N., Brun, Y., Cámara, J., Calinescu, R., Cohen, M.B., Gorla, A., Grassi, V., Grunske, L., Inverardi, P., Jézéquel, J., Malek, S., Mirandola, R., Mori, M., Müller, H.A., Rouvoy, R., Rubira, C.M.F., Rutten, É., Shaw, M., Tamburrelli, G., Tamura, G., Villegas, N.M., Vogel, T., Zambonelli, F.: Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In: Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers. pp. 3–30 (2013). https://doi.org/10.1007/978-3-319-74183-3_1, https://doi.org/10.1007/978-3-319-74183-3_1
32. Mens, T., Serebrenik, A., Cleve, A. (eds.): Evolving Software Systems. Springer (2014). https://doi.org/10.1007/978-3-642-45398-4, https://doi.org/10.1007/978-3-642-45398-4
33. Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., Jazayeri, M.: Challenges in software evolution. In: Proceedings of the Eighth International Workshop on Principles of Software Evolution. pp. 13–22. IWPSE '05, IEEE Computer Society, Washington, DC, USA (2005). https://doi.org/10.1109/IWPSE.2005.7, https://doi.org/10.1109/IWPSE.2005.7
34. Nanda, A., Mani, S., Sinha, S., Harrold, M.J., Orso, A.: Regression testing in the presence of non-code changes. In: 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. pp. 21–30. IEEE (2011)

35. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM transactions on autonomous and adaptive systems (TAAS) **4**(2), 14 (2009)
36. Santelices, R., Chittimalli, P.K., Apiwattanapong, T., Orso, A., Harrold, M.J.: Test-suite augmentation for evolving software. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. pp. 218–227. IEEE (2008)
37. Siqueira, B.R., Ferrari, F.C., Serikawa, M.A., Menotti, R., de Camargo, V.V.: Characterisation of challenges for testing of adaptive systems. In: Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing, SAST 2016, Maringa, Parana, Brazil, September 19-20, 2016. pp. 11:1–11:10 (2016). https://doi.org/10.1145/2993288.2993294, https://doi.org/10.1145/2993288.2993294
38. Staats, M., Whalen, M.W., Heimdahl, M.P.: Programs, tests, and oracles: the foundations of testing revisited. In: Proceedings of the 33rd international conference on software engineering. pp. 391–400. ACM (2011)
39. Su, F.H., Bell, J., Harvey, K., Sethumadhavan, S., Kaiser, G., Jebara, T.: Code relatives: detecting similarly behaving software. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 702–714. ACM (2016)
40. Varshosaz, M., Beohar, H., Mousavi, M.R.: Basic behavioral models for software product lines: Revisited. Sci. Comput. Program. **168**, 171–185 (2018). https://doi.org/10.1016/j.scico.2018.09.001, https://doi.org/10.1016/j.scico.2018.09.001
41. Yu, T.: Simevo: Testing evolving multi-process software systems. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 204–215. IEEE (2017)