Coverage-guided Intelligent Test Loop
A Concept for Applying Instrumented Testing to Self-organising Systems

Jan Kantert[1], Sven Tomforde[2], Susanne Weber[1] and Christian Müller-Schloer[1]
[1]*Institute of Systems Engineering, Leibniz University Hanover, Appelstr. 4, 30167 Hanover, Germany*
[2]*Intelligent Embedded Systems Group, University of Kassel, Wilhelmshöher Allee 73, 34121 Kassel, Germany*

Keywords: Multi-Agent-Systems, Testing, Self-organisation, Organic Computing.

Abstract: Multi-agent systems typically consist of a large set of agents that act on behalf of different users. Due to inherent dynamics in the interaction patterns of these agents, the system structure is typically self-organising and appears at runtime. Testing self-organising systems is a severe challenge that has not received the necessary attention within the last decade. Obviously, traditional testing methods reach their limitations and are hardly applicable due to the runtime characteristics and dynamics of self-organisation. In this paper, we argue that we run into a paradoxon if we try to utilise self-organising testing systems. In order to circumvent parts of the underlying limitations, we propose to combine such an approach with instrumented testing.

1 INTRODUCTION

Technical systems are characterised by a dramatically increasing complexity (Tomforde et al., 2014b) One manifestation of this observation is the law of Glass: Each 25% increase in functionality entails 100% increase in complexity for software solutions (Glass, 2002). Examples for such systems are traffic control systems (Prothmann et al., 2011), data communication networks (Tomforde et al., 2010; Tomforde et al., 2009), volunteer-based resource sharing systems (Kantert et al., 2015), or smart grids (Tomforde et al., 2014a).

As a key concept to counter these challenges arising from complexity issues, self-organisation is applied – meaning that scalable and intelligent control concepts are needed (Tomforde and Müller-Schloer, 2014). Consequently, large parts of former design-time based decisions are transferred to runtime and into the responsibility of the systems themselves (Müller-Schloer et al., 2011). On the one hand, this allows us to master the upcoming issues, e.g. by learning and self-optimisation techniques (Tomforde et al., 2011). On the other hand, this opens novel challenges in terms of reliability and accurate functioning: How to test such a system that will develop parts of its behaviour and important aspects of its structure at runtime?

One possible solution to this problem is to make use of a self-organised testing system – meaning to let the test system develop test cases and situations autonomously and observing if the tested system reacts within certain pre-defined boundaries. In addition, the tested system can be confronted with disturbed situations – so that it has to restore its behaviour. Albeit the obvious advantages, this approach has a severe disadvantage: Who tests the test system?

In this paper, we highlight the resulting paradoxon and propose to combine such an approach with a concept that is known as instrumented testing (Huang, 1978). Although this does not solve the problem of testing the tester completely, it allows for a more standardised test schedule and higher repeatability. Following such a process, the problem of testing self-organised systems, and especially multi-agent systems, may become less complex (Wooldridge, 1998).

The remainder of this paper is organised ad follows: Section 2 briefly summarises the state-of-the-art related to testing self-organised systems. Afterwards, Section 3 formulates a vision to cope with the aforementioned challenges. Section 4 presents a concept to develop an instrumented testing solution for self-organised systems and discusses issues concerning complexity reduction in this context. Finally, Section 5 summarises the paper and gives a brief outlook to future work.

Kantert, J., Tomforde, S., Weber, S. and Müller-Schloer, C.
Coverage-guided Intelligent Test Loop - A Concept for Applying Instrumented Testing to Self-organising Systems.
DOI: 10.5220/0005992702210226
In *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016) - Volume 1*, pages 221-226
ISBN: 978-989-758-198-4

2 TESTING SELF-ORGANISING ADAPTIVE SYSTEMS

Self-adaptive and self-organising systems from the multi-agent system domain (Wooldridge, 1998) are designed to cope with changing internal and external conditions (see, e.g., (Chang et al., 2009; McKinley et al., 2004)). Standard test approaches are situated at design-time and aim at checking the system towards the specification, they try to detect failures, and validate the behaviour. Traditional concepts to achieve solutions for these challenges are based on test cases and consequently static by nature – the underlying dynamics of runtime self-organisation (that is common for multi-agent systems, see (Wooldridge, 1998)) are hardly testable with such an approach (Salehie and Tahvildari, 2009; Welsh and Sawyer, 2010). The applicability of design-time test cases is especially limited due to self-configuration in response to changing requirements and environmental conditions (see (Fredericks et al., 2013)).

In general, anticipating each possibly occurring runtime situation a system will face (in combination with the corresponding status) is considered to be impossible (see, e.g., (Chang et al., 2009; Sawyer et al., 2010; Cheng et al., 2009)). Consequently, the focus of researchers shifted towards assurance technologies and runtime monitoring for testing purposes (see, e.g., (Fredericks et al., 2013; Zhang et al., 2011)) within the last decade. One particular example has been presented by Fredericks et al. (Fredericks et al., 2013) which addresses the assurance of self-adaptive systems with a runtime feedback-loop concept that takes the initial requirements into account. It tries to maintain consistency between design-time test cases and the dynamics at runtime. The problem here is that only design-time based test cases are considered. In general, this and similar concepts try to improve the test , mostly through runtime requirements and specification monitoring. The problem here is that these approaches are typically limited in terms of comparing just the satisfaction of requirements rather than full testing.

In addition, test systems have been proposed that are increasingly characterised by self-organisation themselves. One particular example is the test frameworks as outlined by Eberhardinger et al. (Eberhardinger et al., 2014). Based on introducing corridors of desired behaviour that have to be maintained by the self-organising system, the test framework tries to find situations where these corridors are violated. It thereby makes use of self-organisation and self-adaptation itself. Although the framework's current version seems to be situated at design-time, it might be applied to runtime testing as well. The concept entails another important problem: If the test system itself is self-organised and self-adaptive, how is this test system tested?

Trying to force the system under test into situations where it validates the corridors and then observing if and how fast it recovers is a first important aspect. But it does not necessarily incorporate a test coverage that considers issues and challenges the system will face at runtime. For instance, cooperation with novel systems at runtime that have even not been anticipated during design or the rise of novel communication and coupling technology may lead to influences that have not been anticipated by test engineers.

Partly, this problem has been considered in traditional testing under the term "test oracle" (McCaffrey, 2009). Such an oracle is a mechanism that estimates whether a test has passed or failed. It therefore works on the basis of comparing the observed system output with for a given test-related input, to the desired output produces by the oracle following a pre-defined model. This concept goes back to Howden who initially defined the term and highlighted the applicability (Howden, 1978) (this concept has been adapted towards the utilisation of various oracles in (Weyuker, 1980)). The automatic generation of test cases as performed by the oracle covers the general idea of testing the self-organised system under test as outlined before. The major drawback of this solution is that the oracle needs a model that defines how to generate test cases – in case of multi-agent systems, such an appropriate model is hardly available. More precisely, if a sophisticated model exists that allows for a suitable test case generation, the problem itself would follow a pre-definable model – as a consequence, the solution system does not need to make use of self-adaptation and self-organisation techniques. Finally, *multi*-agent systems require communication and cooperation. Testing communication relations is not an issue and can be handled by standard network approaches. In contrast, aspects of the *multi* property within the term of multi-agent systems requires taking group composition, negotiation results, and condition chains following dynamics into account, for instance. This again leads to the questions regarding the runtime behaviour as outlined before when considering self-organisation capabilities.

Consequently, even the self-organised test system will not be able to check how the system under test will react – still there is a need of anticipation by engineers. Some of these issues might be addressed by combining the concept with techniques such as instrumented testing, which is outlined in the following section.

3 COMBINATION WITH INSTRUMENTED TESTING

Self-organising systems (such as multi-agent systems) consisting of multiple distributed entities which can adapt based on their environment are too complex to be tested using a traditional test harness, see (Kantert et al., 2014) for such a complex example. To create novel scenarios and test the adaptation of such systems, we require an adaptive self-organising test system.

3.1 Challenge

The goal of testing is to i) find errors (i.e., crashes) and ii) ensure correct behaviour and convince the user/customer that the system solves the right problem. When finding errors (i) it is important to cover as much of the functionality as possible. In contrast, ensuring correct behaviour (ii) typically only covers all important features of a system.

However, when using a self-organising test system, we cannot ensure correct behaviour of the test system (except by using another self-organised test system on top). We assume that a function $Q(t,s)$ exists, which gives us the quality of the solution s returned by the system for a given test point t. Thereby, a self-organised test system can instantiate different system configurations, autonomously test the solution for different test points, and rate their quality.

Unfortunately, this system can neither provide a solution for (i) nor (ii). Coverage (i) usually cannot be measured in self-organising systems and, therefore, comprehensive error testing is not possible. Additionally, testing self-organisation/adaption in/to unknown situations as an important feature of such systems is hard because we typically do not know unanticipated situations.

The best we can do is fuzzy testing (Sutton et al., 2007) by trying numerous possible configurations and test points. Unfortunately, even in traditional software systems the state space is extremely large and this is not possible in practise.

Therefore, coverage-guided instrumented fuzzy testing is used where the fuzzer (such as American Fuzzy Lop (AFL) (Zalewski, 2015b) or (Drysdale, 2016)) receives feedback about all covered branches in the tested software. Using that data, the fuzzer can "intelligently" modify the test point to cover all functionality. Still, this approach only allows to optimise coverage and cannot verify correctness. Therefore, it has been applied very successfully recently to find simple crashes and security vulnerabilities in complex software such as compression lib or XML parsers (Zalewski, 2015a). Unfortunately, we cannot use traditional instrumented testing in self-organising systems containing multiple agents because agents are considered to be blackboxes (Wooldridge, 1998). Therefore, we cannot know which functionality gets tested.

3.2 Vision

In the following, we present a vision of how instrumented testing can be processed for self-organising systems. We envision to use coverage-guided instrumented testing (Huang, 1978) for self-organising systems containing of multiple distributed agents with configuration C, input I and generating output O. When instantiating a test harness, we propose to relax the blackbox assumption for agents slightly to allow for instrumentation. Our approach does not require access to the source code but to coverage information about the running binary. Additionally, we assume deterministic behaviour when rerunning the same scenario in our test harness (i.e., using the same random seed by including it in C).

In the first step, we test single agents using instrumented testing and a fuzzer. In every execution an agent with configuration C receives a series of messages I. Instrumentation allows us to ensure (nearly) full coverage of all functionality and a creation of a (minimal) test set. Typically, we cannot verify semantic correctness of response O. However, we can find simple crashes and ensure that all messages comply to the predefined protocol.

Unfortunately, this does not test any system dynamics or self-organisation. Additionally, hidden state (e.g., learning algorithms used in the agent) may limit the ability of a fuzzer to gain good coverage.

Therefore, we propose to test the system in integration in a second step. Again, we assume that the test harness including the environment is isolated and can be repeated deterministically. C contains the configuration of all agents while I describes all input from the environment during runtime. Furthermore, we need a function $\Omega(I,O)$ to verify the quality of an output O for a certain (environment) input I.

Using instrumented fuzzing again, we can find many unique situations and simulate the system response. Because of the enormous configuration space C and input size I this can probably not create an exhaustive list. This could be improved by testing only a limited configuration C.

Afterwards, tuples on the list are rated using Ω and the overall quality of the system is calculated. Furthermore, critical situations and inputs can be identified and returned to the designer for further analysis.

4 APPROACH

In the approach, we present a concept to apply our vision from the previous section to test self-organising system with coverage-guided fuzzing in a test harness.

4.1 Testing a Single Agent

As introduced in our vision (see Section 3.2), the first step is to put a single agent S into a test harness (see Figure 1). Input I comprises the configuration C and a sequence of data which is sent to the agent. All data sent by the agent is contained in Output O.

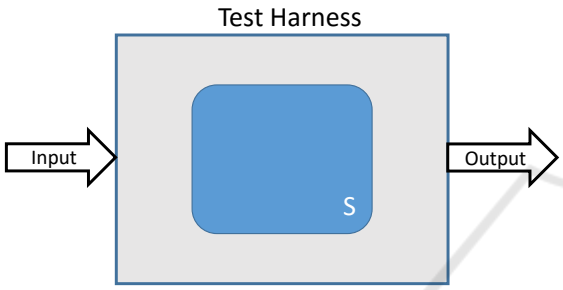

Figure 1: Putting a single agent S into a test harness.

In practice, a test harness for coverage-guided testing of self-organising system is more complex. In Figure 2, we show an overview of the test system. The *Fuzzer* tries to select an input to explore previous untested functionality. This input is passed to the *Executor* which configures and executes a test run of the *Self-organising System* under test. After a run completed, the output O and coverage infos are passed back to the *Fuzzer*. If a configuration with new coverage was found or failures occured, it is reported to the *Test Manager* which stores the configuration in the *Test Database*. The *Test Manager* queries the database to find areas with low coverage and passes those configurations to the *Fuzzer* to explore them.

With this architecture it is possible to perform coverage-guided tests on self-organising systems in a test harness.

4.2 Levering Complexity Reduction

Most agents in self-organising systems use an intelligent control mechanism, such as the observer/controller pattern (O/C (Tomforde et al., 2011)) or similar architectures (such as MAPE (IBM, 2005)). All of those architectures have in common that they reduce the complexity of configuring the internal system to the outside. In Figure 3, we show a system with an intelligent control mechanism (CM - realised

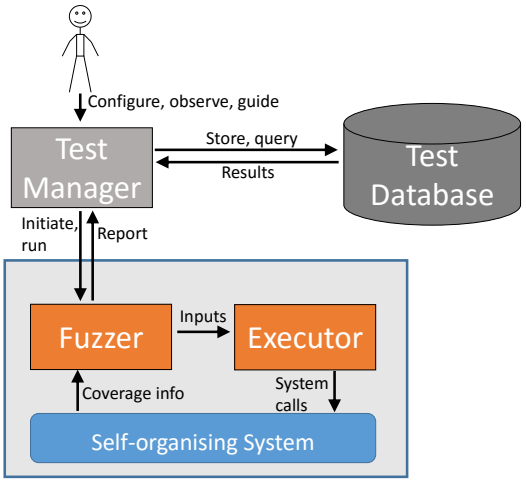

Figure 2: Coverage-guided test harness for self-organising systems.

as O/C) on top. I is passed directly to the production engine below but all external configuration $C_e = C$ is going through the O/C which usually passes more configuration bits C_i to the production engine.

The complexity reduction can be measured using the variability $v(C_e, C_i)$ (see (Schmeck et al., 2010)). First, we determine the size of both configuration parameter sets C_i and C_e in V_i and V_e (see Equations (1) and (2)). We use those to calculate the complexity reduction R (Equation (3)) and the variability v (Equation (4)).

$$V_i := \log_2 C_i \tag{1}$$

$$V_e := \log_2 C_e \tag{2}$$

$$R := V_i - V_e \tag{3}$$

$$v(C_i, C_e) := \frac{R}{V_i} = \frac{V_i + V_e}{V_i} \tag{4}$$

In general, a complexity reduction is considered as good. However, some internal configurations become unavailable to external configuration. When trying to perform coverage-guided testing, this should be considered when looking at the reached coverage. The higher R gets, the lower the resulting coverage will be.

Therefore, in a test harness, a agent should be considered to look similar as sketched in the illustration of Figure 4. We are only able to reach explore the full external configuration C_e of the controller and some parts of the SuOC may be unavailable to external configuration. However, since those parts will also be unavailable during runtime of the agent, there is also no value in covering those internal configurations.

4.3 Testing Self-organising Systems

Finally, in step two, we test self-organising systems in

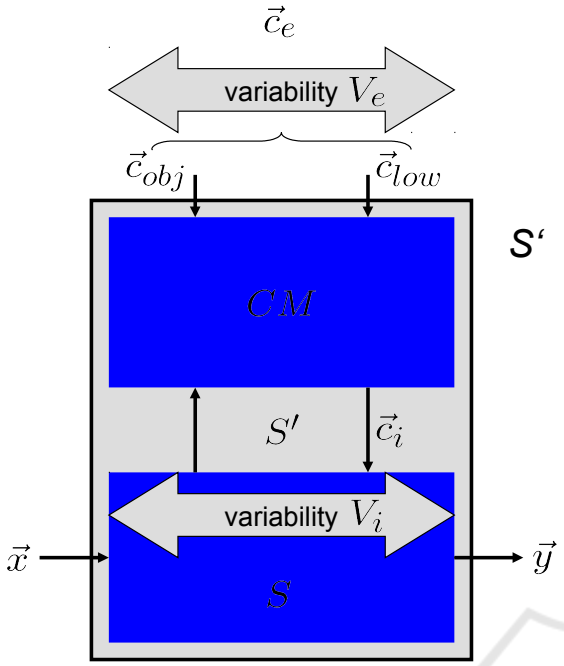

Figure 3: Complexity reduction as proposed by (Schmeck et al., 2010).

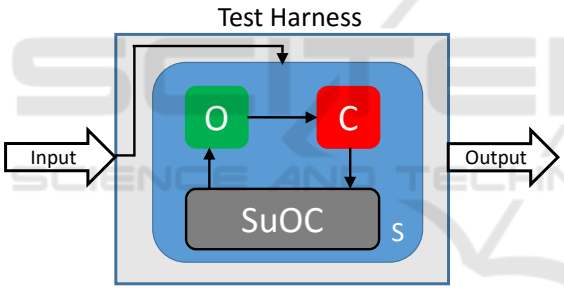

Figure 4: Complexity reduction in test harness with a single agent.

integration. In this scenario, the external configuration of all agents is contained in input *I* and the output of all agents in output *O*. In Figure 5, we show a system consisting of three agents in test harness.

Since the size of the input *I* is larger, naive coverage-guided fuzzing is much slower. However, we can exploit the *Test Database* from the single agent tests to select configurations which enables or disables certain internal parts of the agent. That way, the fuzzing can invest more effort into testing distinct situations which allows testing different communication and cooperation patterns.

5 CONCLUSION

This paper outlines the challenges that we assume as

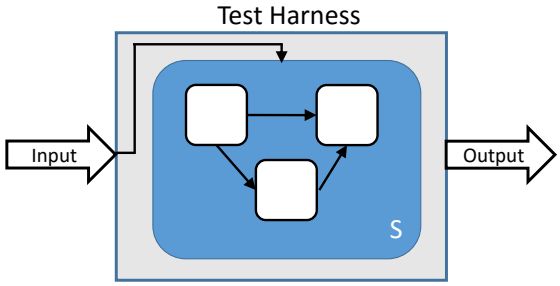

Figure 5: Putting a Self-Organising System into a test harness.

most important in the context of testing multi-agent systems with respect to self-adaptive and self-organised behaviour. We highlighted the need of novel solutions to automated testing, especially since traditional test cases are too static to be applicable.

As a result, we found a pardoxon: Testing self-organised systems with a self-organised test systems poses the question how such a test system is tested itself. Since there seems to be no obvious solution, it will most probably be the best way to follow when considering testing. However, we proposed to combine such a concept with more established techniques such as instrumented testing in order to reduce the possible impact of vast configuration and situation spaces of the system.

ACKNOWLEDGEMENTS

This research is funded by the research unit “OC-Trust” (FOR 1085) of the German Research Foundation (DFG).

REFERENCES

Chang, B., de Leos, R., Inverardi, P., and Magee, J., editors (2009). *Software Engineering for Self-Adaptive Systems*. Springer Verlag, New York, USA. ISBN 978-3-642-02161-9.

Cheng, B. H., Sawyer, P., Bencomo, N., and Whittle, J. (2009). "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*, MODELS '09, pages 468–483, Berlin, Heidelberg. Springer-Verlag.

Drysdale, D. (2016). Coverage-guided kernel fuzzing with syzkaller. https://lwn.net/Articles/677764/.

Eberhardinger, B., Seebach, H., Knapp, A., and Reif, W. (2014). "towards testing self-organizing, adaptive systems". In Merayo, M. and de Oca, E., editors, *Testing Software and Systems*, volume 8763 of *Lecture Notes*

in Computer Science, pages 180–185. Springer Berlin Heidelberg.

Fredericks, E. M., Ramirez, A. J., and Cheng, B. H. C. (2013). "towards run-time testing of dynamic adaptive systems". In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 169–174, Piscataway, NJ, USA. IEEE Press.

Glass, R. L. (2002). *Facts and Fallacies of Software Engineering*. Agile Software Development. Addison Wesley, Boston, US.

Howden, W. E. (1978). Introduction to the theory of testing. In *Software Testing and Validation Techniques*, pages 16 – 19. IEEE Computer Society Press, Long Beach, CA, USA.

Huang, J. (1978). "program instrumentation and software testing". *Computer*, (4):25–32.

IBM (2005). An architectural blueprint for autonomic computing. Technical report, IBM.

Kantert, J., Edenhofer, S., Tomforde, S., and Müller-Schloer, C. (2015). Representation of Trust and Reputation in Self-Managed Computing Systems. In *IEEE 13th International Conference on Dependable, Autonomic and Secure Computing, DASC 2015*, pages 1827–1834, Liverpool, UK. IEEE.

Kantert, J., Scharf, H., Edenhofer, S., Tomforde, S., Hähner, J., and Müller-Schloer, C. (2014). A Graph Analysis Approach to Detect Attacks in Multi-Agent-Systems at Runtime. In *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*, pages 80–89, London, UK. IEEE.

McCaffrey, J. D. (2009). *Software Testing*. Fundamental Principles and Essential Knowledge. Samir Riad Press. ISBN-13: 978-1439229071.

McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. C. (2004). "composing adaptive software". *Computer*, 37(7):56–64.

Müller-Schloer, C., Schmeck, H., and Ungerer, T., editors (2011). *Organic Computing - A Paradigm Shift for Complex Systems*. Autonomic Systems. Birkhäuser Verlag.

Prothmann, H., Tomforde, S., Branke, J., Hähner, J., Müller-Schloer, C., and Schmeck, H. (2011). Organic Traffic Control. In *Organic Computing – A Paradigm Shift for Complex Systems*, pages 431 – 446. Birkhäuser Verlag.

Salehie, M. and Tahvildari, L. (2009). "self-adaptive software: Landscape and research challenges". *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E., and Finkelstein, A. (2010). "requirements-aware systems: A research agenda for re for self-adaptive systems". In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 95–103, Washington, DC, USA. IEEE Computer Society.

Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M., and Richter, U. (2010). Adaptivity and Self-organisation in Organic Computing Systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 5(3):1–32.

Sutton, M., Greene, A., and Amini, P. (2007). *Fuzzing: brute force vulnerability discovery*. Pearson Education.

Tomforde, S., Hähner, J., Seebach, H., Reif, W., Sick, B., Wacker, A., and Scholtes, I. (2014a). Engineering and Mastering Interwoven Systems. In *ARCS 2014 - 27th International Conference on Architecture of Computing Systems, Workshop Proceedings, February 25-28, 2014, Luebeck, Germany, University of Luebeck, Institute of Computer Engineering*, pages 1–8.

Tomforde, S., Hähner, J., and Sick, B. (2014b). Interwoven Systems. *Informatik-Spektrum*, 37(5):483–487. Aktuelles Schlagwort.

Tomforde, S. and Müller-Schloer, C. (2014). Incremental Design of Adaptive Systems. *Journal of Ambient Intelligence and Smart Environments*, 6:179 – 198.

Tomforde, S., Prothmann, H., Branke, J., Hähner, J., Mnif, M., Müller-Schloer, C., Richter, U., and Schmeck, H. (2011). Observation and Control of Organic Systems. In *Organic Computing - A Paradigm Shift for Complex Systems*, pages 325 – 338. Birkhäuser Verlag.

Tomforde, S., Steffen, M., Hähner, J., and Müller-Schloer, C. (2009). Towards an Organic Network Control System. In *Proc. of the 6th Int. Conf. on Autonomic and Trusted Computing (ATC'09), held in Brisbane, Australia (Jul 7, 2009 - Jul 10, 2009)*, pages 2 – 16. Springer Verlag.

Tomforde, S., Zgeras, I., Hähner, J., and Müller-Schloer, C. (2010). Adaptive control of Wireless Sensor Networks. In *Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC'10), held in Xi'an, China (October 26-29, 2010)*, pages 77 – 91.

Welsh, K. and Sawyer, P. (2010). "understanding the scope of uncertainty in dynamically adaptive systems". In Wieringa, R. and Persson, A., editors, *Requirements Engineering: Foundation for Software Quality*, volume 6182 of *Lecture Notes in Computer Science*, pages 2–16. Springer Berlin Heidelberg.

Weyuker, E. J. (1980). The oracle assumption of program testing. In *Proceedings of the 13th International Conference on System Sciences (ICSS)*, pages 44 – 49, Honolulu, HI, USA. IEEE.

Wooldridge, M. J. (1998). *Agent Technology: Foundations, Applications, and Markets*. Springer Science & Business Media, Berlin, Heidelberg, New York.

Zalewski, M. (2015a). Bugs found by AFL. http://lcamtuf.coredump.cx/afl/.

Zalewski, M. (2015b). Technical "whitepaper" for afl-fuzz. http://lcamtuf.coredump.cx/afl/technical_details.txt.

Zhang, Z., Thangarajah, J., and Padgham, L. (2011). "automated testing for intelligent agent systems". In Gleizes, M.-P. and Gomez-Sanz, J., editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 66–79. Springer Berlin Heidelberg.