A Survey for Testing Self-organizing, Adaptive Systems in Industry 4.0

Caterina Rotondo
Chair of Intelligent Systems
University of Passau
Passau, Germany
caterina.rotondo@uni-passau.de

***Abstract*—Complexity in technical development increases rapidly. Regular system are no longer able to fulfill all the requirements. Organic computing systems are inspired by how complexity is mastered in nature. This leads to a fundamental change in software engineering for complex systems. Based on machine learning techniques, a system develops self*-properties which allows it to make decisions at runtime and to operate with nearly no human interaction. Testing is a part of the software engineering process to ensure the functionality and the quality of a system. But when using self-organizing, adaptive systems traditional testing approaches reach their limits. Therefore, new methods for testing such systems have to be developed. There exist already a lot of different testing approaches. Most of them developed within a research group. Nevertheless, there is still a need for further discussion and action on this topic. In this paper the challenges for testing self-organizing, adaptive systems are specified. Three different testing approaches are reviewed in detail. Due to the ongoing fourth industrial revolution it is discussed which of these approaches would fit best for testing industrial manufacturing robots.**

***Index Terms*—Organic Computing, Self-organizing systems, Adaptive systems, Software Engineering, Software Testing, Industry 4.0**

I. Introduction

Over time an increasing complexity in different aspects of technical development could be observed. This can be explained by Glass's Law. Glass states that if the functionality and the requirements increase by 25% the complexity is increased by 100% [1]. To master this complexity nature can be taken into account. Therefore, Organic Computing (OC) was proposed in [2].

OC systems consist of autonomous sub-systems and a variety of control mechanisms (CMs). A sub-system is also called system under observation and control (SuOC) and it undertakes the productive task of the system. SuOCs possess sensors and actuators which enable them to interact with each other and their environment. CMs are split into an observer and a controller which influence the SuOCs. The architecture of such an autonomous sub-system with control mechanism can be seen in Fig. 1. For a deeper insight into the observer/controller design paradigm for OC systems see [3].

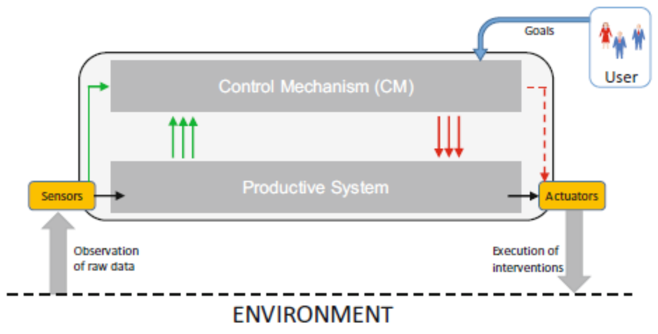

Fig. 1. A part of a OC system which displays the system architecture with SuOC and CM.
Source: [4, p. 174]

Based on local knowledge[1], OC systems have the ability to modify their structure autonomously in response to certain events during runtime, e.g. changes in the environment. They are able to organize and adapt[2] themselves autonomously to previously unknown situations. OC systems gain experience over time and learn which behavior fits best in a certain situation. Because of that, OC systems are more flexible than regular systems. This flexibility also leads to a more robust system against disturbances which change their system state. ”These systems are able to survive in a world of change.” [4, p. 6] Such a behavior of systems is possible because OC systems often build upon machine learning models. Tom Mitchell defined ”learning” for computer systems as follows:

> ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” [5]

The attributes wich can be evolved within an OC system are typically referred to as so-called self*-properties. The * is a placeholder for any property that evolves in the system itself. Such porperties can be self-organizing, self-adaption, self-

[1]External control influences are limited to a minimum. There is nearly no human interaction at runtime.

[2]It is called self-adaption if a system reconfigures itself without (or with only minimal) human interaction. [4]

healing, self-protection or self-optimisation. More attributes are listed and explained in detail in [6].

In [4] it is stated that OC causes a paradigm shift for complex systems. ”[OC] means to move traditional design-time decisions into runtime, and from the system engineer to the systems themselves” [4, p. 6,7]. Because of that, testing self-organizing, adaptive systems (SOAS) has also to be moved from design-time into runtime.

Many initiatives deal with similar objectives and approaches like OC to achieve the goal of mastering the increasing complexity. Examples are Autonomic Computing (AC) [7], which started evolving parallel to OC, or ProActive Computing [8], which can be seen as a precursor for both AC and OC.

The remainder of this paper is structured as follows: In Section II related work is discussed. Section III lists the main-challenges for testing self-organizing systems. Current approaches for testing such systems are provided in Section IV. Section V adresses the case ’Smart Factory’ and the application of the introduced testing approaches is discussed. The paper is concluded in Section VI.

II. RELATED WORK

SOAS are applied in a lot of fields, and the usage of those systems is increasing fast nowadays. These application areas include more and more security-critical and safety-critical fields like the organic control of traffic lights [9] or the usage of so-called smart grits. Thus, the necessity of testing such systems draws more and more attention. OC started to establish itself in 2003 and a lot of people are conducting research on testing SOAS since then.

In 2005 Schumann and Visser discussed the risks of using autonomous systems in their paper with the title ”Autonomy Software: V& V Challenges and Characteristics” [10]. The paper is based on the result of a survey at the National Aeronautics and Space Administration (NASA). Based on that, they discussed different techniques for validating and verifying SOAS and how to apply them.

Nguyen et al. provided a paper with the title ”Testing in Multi-Agent Systems” [11]. Different approaches were reviewed. They were not only listed but also classified in two different dimensions. The first dimension refers to one of the following testing levels: unit testing, agent testing, integration testing, system testing, acceptance testing. These testing levels were already introduced by Nguyen in his doctoral thesis [12]. The second dimension refers to the testing techniques (active or passive testing). For each class, some state-of-the-art approaches were reviewed. Additionally, they were tagged in term of their maturity level (usable, in progress, concept).

In 2016 Helle, Schamai and Strobel wrote the paper ”Testing of autonomous systems - challenges and current state-of-the-art” [13]. They have made an effort to summarize the challenges in testing SOAS, and listed some of the different testing approaches developed until then.

III. CHALLENGES FOR TESTING SELF-ORGANIZING SYSTEMS

In this section it is discussed why it is important to test SOAS and what the challenges are in testing those systems compared to traditional systems.

Verification and validation (V&V) of software is important to ensure that the software fulfills its intended purpose and to ensure a high quality of the software. This is usually done by testing the software. In traditional software development this is realized by generating test cases. ”A test case is a triplet [I,S,O], where ’I’ is data input to the system, ’S’ is the state of the system at which the data is entered and ’O’ is the expected output of the system.” [14, p.220] This means that for every given input ’I’ we have a fixed output ’O’, and one can ensure that the developed software delivers the according output.

Such a traditional V&V approach can not be used for SOAS. Due to their self*-properties one can not predict at design-time how the systems agents will react to a change in the environment at runtime. For that reason, it is important to also transfer the testing of SOAS from design-time to runtime.

Furthermore, not only the overall SOAS should be tested, but all its sub-systems as well. The SOAS may remain in an acceptable state, even if one or more of its sub-systems are faulty.

Based on a survey at NASA, for which the results are presented in [10], three main issues have emerged which reason the greater complexity in testing SOAS. Those issues have been taken up in [13] as the terms ”complex environment”, ”complex software” and ”non-deterministic behaviour”:

- **Complex environment:** ”[T]he larger size and higher complexity of the valid input space [...].” [10] This makes traditional testing impossible. Because of the changing environment, the input might be partially or completely unknown. There is no fixed input for a possible test case as mentioned above.
- **Complex software:** ”[T]he complexity of the program logic [...] required to derive the answer in the autonomous system [...].” [10] SOAS are often based on machine learning algorithms which give the agents their self*-properties.
- **Non-deterministic behavior:** ”[T]he size and complexity of the domain model[3] and description of the environment.” [10] Due to the self*-properties, the knowledge of SOAS may change over time. Given the same input, it might react differently. There is no fixed output for traditional test case generation as mentioned above.

[3]The self-knowledge of a SOAS about its state, its goal, its environment and much more at the beginning of its lifespan is often called a domain model.

IV. APPROACHES FOR TESTING SELF-ORGANIZING SYSTEMS

There are many research projects which deal with the challenges of testing SOAS and thus a lot of different approaches for testing such systems exist. In this section the following three approaches are reviewed in more detail:

- Remote testing module
- The corridor enforcing infrastructure
- Black box validation

A. *REMOTE TESTING MODULE*

In [15] *self-analysis* is introduced as a new self*-property for self-organizing systems. This property is realized by performing self-tests. Therefore, each agent of a self-organizing system possesses an integrated Remote Testing Module (RTM). ”The task of an RTM is to test other agents in the system.” [15] To fulfill this task, it either has access to the sensors and actuators of the agent it is integrated in, or it can possess its own sensors and actuators if necessary. The following two types of tests can be performed:

- **Passive tests:** The information an agent collects and distributes via its sensors and actuators are evaluated by the RTM.
- **Active tests:** Test events are being generated by the RTM. Then, the reaction of an agent is evaluated.

In both cases feedback is provided by the RTM which enables the tested agent to immediately adapt its behavior.

To exclude the possibility of a malfunctioning RTM, more RTMs can cooperate. If a single RTM identifies a faulty agent one can not be sure that indeed the agent is faulty, or if a failure has occured within the RTM itself. Therefore more RTMs can test the same agent. Only if a predefined number (greater than one) of RTMs confirm a malfunction of the same agent, one can assume that the agent is actually faulty.

For further reading see [15]. There are also two use cases included.

B. *THE CORRIDOR ENFORCING INFRASTRUCTURE*

Another approach for testing of SOAS is shown in [16]. The corridor enforcing infrastructure (CEI) is introduced for enabling tests at runtime. ”The CEI is an architectural pattern for SOAS, based on decentralized feedbackloops used to monitor and control single agents or small groups of agents in order to enforce that all system requirements are fulfilled at all times.” [16] It ensures that the agents of SOAS detect changes in the environment and adapt themselves accordingly to fulfill their goals.

This approach is based on the corridor of correct behavior (CCB). Such a CCB is illustrated in Fig. 2. The invariant (INV) specifies and describes the acceptable states of a system which form the CCB. It seperates the correct from the incorrect behavior. Examples for acceptable states are indicated as S_0 - S_3 in Fig. 2. Once the CCB is violated the system should be able to reorganize itself to go back into a safe state. The red flash in Fig. 2 indicates the violation of the CCB. The successfull reorganization process is marked with a green tick in Fig. 2. If the system is not able to reorganize itself properly, the system will remain in an unacceptable state and an error would occur. This is marked with a red cross in Fig. 2. A CCB is usually generated automatically or semi-automatically from the system requirements. For further readings about the CCB and the so called restore invariant approach (RIA) see [17].

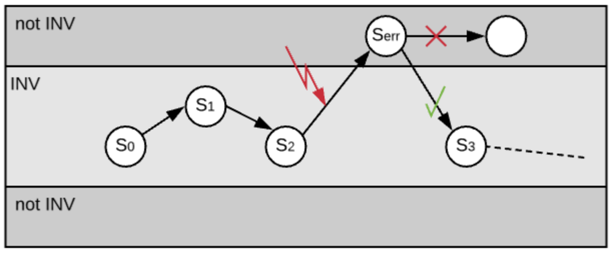

Fig. 2. State-based corridor of correct behavior.
Source: Figure adapted from Fig.1 in [16]

According to [16] it is sufficient to only test the CEI. An error-free CEI guarantees that all sub-systems of SOAS are in an acceptable state. This leads to an enormous reduction of test effort.

To test the CEI, automatically generated environmental variation scenarios (EVS) are used. The EVS is built from sequences which result in a violation of the CCB. The test scenarios orient themselves on three traditional test processes:

- **Unit testing:** ”In unit testing, the units/modules of a system are tested in isolation.” [14]
- **Integration testing:** In integration testing, the interactions between the individual units/modules are tested. Those units may interact with each other using simple message/paramter passing. ”Testing is done to check that there are no errors in the paramter passing.” [14]
- **System testing:** ”System testing is the testing of the whole system based on its specification.” [14]

Thus, this approach covers all stages of traditional software testing. Based on those test processes. the EVS are structured as follows:

- First the EVS are built on single agent situations. The functionality of single agents is validated by testing its components (sensors and actuators). An agent should detect changes in the environment and react accordingly to further reach its goals. This can be done by simulating the input for mock-up agents without setting up initial states.
- The EVS are extended to be able to test the interaction mechanisms between two or more agents. This interaction can no longer be simulated because it is hard to

predict which agents are going to communicate and what exactly they are communicating. The mock-up agents are partially replaced by real, interacting agents.
- For testing a fully integrated system the EVS are further extended. All mock-up agents are replaced by real agents. Several test cases are combined and executed in sequence.

This approach ensures that the CEI detects violations of the CCB and reacts accordingly to lead the system back into an acceptable state within the CCB.

For further reading see [16].

C. BLACK BOX VALIDATION

In [18] an adaptive simulation model is used to validate SOAS. Therefore, black box tests are performed on acceptance level. Acceptance testing is a type of system testing (cf. Section IV-B, [14, p. 242]). The internal structure of a system to be tested via black box testing is not accessible while testing. It is consideres as a ”black box” where one can not see into. To validate the system input data is given to the black box. If the output fulfills the system requirements the system has been validated successfully [14, p. 220].

To deal with the challenges of testing SOAS (cf. Section III) a model is provided which simulates all adaptive-related processes. To ensure correct adaption processes, different aspects have to be taken into account. It has to be simulated when exactly a system has to initialize an adaption process and how exactly this self-adaption has to be executed. Further, it has to be simulated in what state the system should be after adaption. To meet these requirements, the overall simulation model is divided into collaborating models. A graphical representation of the provided simulation model can be seen in Fig. 3.

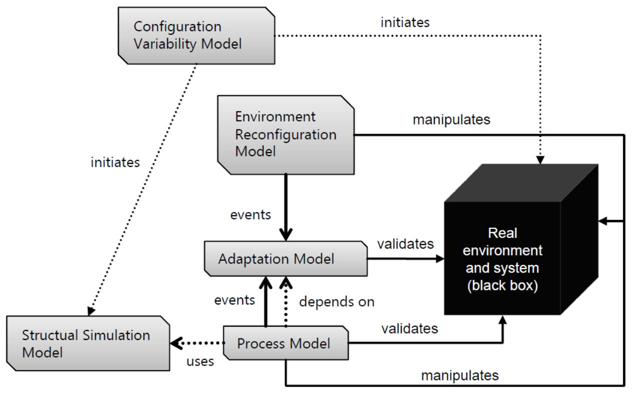
Configuration Variability Model
initiates
initiates
Environment Reconfiguration Model
manipulates
events
Adaptation Model
validates
Real environment and system (black box)
events
depends on
Structual Simulation Model
uses
Process Model
validates
manipulates

Fig. 3. Graphical representation of the simulation model with a black box. Source: [18]

”Parts of [the] model are enriched with assertions on the System Under Tests (SUT) interface in order to define how a simulation state has to be concretely validated.” [18]

The competences of the single components are partitioned as follows:

- **Structural Simulation Model:** This model simulates the current state of the real system. It reflects environmental settings and it manages ”several assumptions on the real system” [18] to be able to ”work with more detailed state-defining information” [18].
- **System Process Model:** Within this model the systems behavior based on its current state is simulated. It is defined how requests and feedback should be processed.
- **Environment Configuration Variability Model:** The Environment Configuration Variability Model contains all environmental states. This way all possible initial states can be simulated and validated. Every valid environment state should be covered by this model. To deal with the enormous amount of possible states, classification can be used [18]. The Cambridge Dictionary[4] defines classification as ”the act or process of dividing things into groups according to their type”. The environment states ”[...] are split into equivalence classes and only representatives are tested.” [18] It is assumed that each member of a class produces the same output. This way it is sufficient to test only representatives.
- **Environment Reconfiguration Model:** This component of the simulation model defines all changes in the environment which trigger the system to reconfigure itself.
- **Adaption Model:** This model simulates in what way the system should adapt itself to remain in an acceptable state. Environmental changes as well as reactions to internal feedback are able to influence this model.

For further reading see [18].

V. CASE STUDY: INDUSTRIAL MANUFACTURING ROBOTS

In this section the term ’Industry 4.0’ is outlined. Furthermore, it is discussed which of the above introduced testing approaches would be best for testing industrial manufacturing robots.

A. Industry 4.0

Paradigm shifts can also be observed in the industry. They are better known as ”industrial revolutions”. Currently we are going through the fourth industrial revolution, which is commonly known under the term ”Industry 4.0”. The evolution and improvement in the manufacturing industry is going to lead to so-called smart factories [19]. Manufacturing robots can be improved by using SOAS.

The economical rise in the 1950s involved a rapid technical development. This technical progress led from automatic machine tools up to the automation of whole production processes. An increasing automation in production was already achieved within the third industrial revolution (1969). Human interaction could be minimized using robots. Industrial robots are self-moving automatons which are programmed to fulfill

[4]See http://dictionary.cambridge.org/ (last access 2019/15/01).

certain workflows. Robots increase productivity and perform dangerous, hard and unpleasant work, so that no human has to do them.

Since then, a lot of progress was made in the field of computing systems. This progress and the progressive digitalization are significant for the ongoing industrial revolution. Equipped with sensors, actuators and self*-properties the robots are able to communicate with each other and to react autonomously to their environment. Human interaction is decreasing towards zero.

There are many factors which justify the usage of SOAS in the field of industrial/manufacturing robots. In [4] three major reasons are listed for the increasing usage of OC systems in industrial robots. These reasons are:

- **Energy efficiency:** The power consumption of factories should be minimized not only to reduce costs, but also to be more environmentally friendly.
- **Reliability:** Physical damage and total system failures can be decreased by increasing the reliability of robots.
- **Utility:** Increasing the skills of a robot, e.g. its accuracy or its agility.

All reasons aim at improving the manufacturing. Industry 4.0 covers multiple needs. Through their improved equipment the robots are able to collect a lot more data than they were able to until now. By analyzing this data the overall factory can be improved. Costs can be decreased and a better, constant quality can be ensured. Not only the costs for energy consumption can be decreased, but also the costs for maintenance for example. One reason for a higher quality is that the influence of the human factor is decreased. Robots are able to perform the same workflow with a high repetition accuracy over a long period of time without any quality loss.

B. Testing industrial robots

Testing industrial robots is of particular significance. Standards for safety and security have also to be applied to such machines. Especially if humans collaborate with manufacturing robots in a shared workplace. To make sure a robot does not cause any damage to its environment, safety is important. Its environment includes humans as well as the product it is working on. Security is needed to ensure that the environment is not disturbing the robot during operation. Loss and theft of data or hostile takeovers from external sources have to be countered. For further reading about the importance of testing robots see [20].

There are a lot of reasons why testing industrial robots is important. But what testing approach is the best? There is no general testing approach which fits all problems. The best approach depends on the overall task, the input and the implementation of a system. In the following, advantages and disadvantages of the above introduced approaches (cf. IV) are discussed. Further, it is discussed which of them would be best for testing industrial robots.

- **Remote Testing Module:** The RTM can be realized quickly and easily. The specifications of the system have to be known in order to identify malfunctioning agents within the system. In Subsection IV-A it is written that a predefined number of RTMs have to confirm a malfunction of the same agent to make sure that indeed the agent is faulty and not the RTM itself. A problem that can occur is that more RTMs are malfunctioning and the predefined number of RTMs to identify a faulty agent is not reached. This way a malfunctioning agent is not identified and the complete system can behave in a wrong way. Another disadvantage of an RTM is that no effective countermeasures are initiated. This is a feature which is going to be part of future work on RTMs [15].
  Even though this is a cost-effectively approach for testing SOAS, it does not seem to be developed further enough at this point in time. There are still open security and safety issues which have to be tackled.
- **The Corridor Enforcing Infrastructure:** An advantage of the CEI is that it reduces the test effort a lot. It makes an enormous difference if the CEI or the whole system has to be tested. Furthermore, through the usage of EVS all testing levels are covered by this approach. A problem arises if the system requirements and specifications are not completely known or if they are not detailed enough. In this case the CCB, and thus the CEI, is going to be inaccurate. As a result acceptable states may be identified as unacceptable or vice-versa. In the first case it may lead to unnecessary and unwanted adaption processes while in the second case there will be no adaption at all. Both cases are very bad.
- **Black Box Validation:** The simulation model compares the simulated and thus expected output with the actual output of the black box (the real system). The accuracy of the comparison can be used as a quality measurement. Undesirable behavior is detected if the outputs differ. But a serious disadvantage of this validation method is that with black box testing the precise source of error remains unknown. It may be known which environmental state or which process caused the error. But the malfunctioning agent or sub-system can not be detected because it is not possible to have a look into the real system. Again, the specifications of the systems have to be well known to generate accurate test cases. This is the only way that every possible state can be simulated by the model. If not all possible inputs are identified, there will remain untested paths in the real system. Besides, the development of a simulation model is expensive.

It can be seen that all three testing approaches have their advantages and disadvantages. Because of that, it has to be carefully considered which approach to use for testing industrial manufacturing robots.

Overall, it can be said that the CEI would be the best approach for testing industrial manufacturing robots after the above-mentioned points have been taken into account. The

RTM is not yet developed enough to be successfully used in this field. The realization of the simulation model is expensive and the identification of all possible inputs and environmental conditions is time consuming. There may remain untested system components which is not preferred in such an safety and security critical field. Furthermore, this validation method does not change the systems behavior. If an error is detected, the system has to be changed manually to be able to cope with this error in the future. Using a CEI for testing SOAS seems to be the most sophisticated approach for testing industrial manufacturing robots. The most costly thing about this approach is to elaborate all specifications of a certain type of manufacturing robot. It is important to take the time for this step. Only in this way it is possible to ensure that the CCB is indeed correct. Once all this requirements are known and recorded every further step will be done automatically by the CEI approach. This test method is executed during runtime, and is therefore able to change the behavior of the system.

VI. Conclusion and Future Work

With the emerging usage of SOAS the need for testing SOAS has also increased. New challenges for testing such systems have been discovered. Previously used test methods, which work for regular/traditional systems, are no longer applicable for those complex OC systems.

A lot of research is conducted in this field. Various research groups are trying to develop a sufficient testing method for SOAS. The challenges which are encountered in developing testing approaches for SOAS were presented in this paper. Three of those testing approaches were discussed and reviewed within this paper. To evaluate these approaches the use case "Industry 4.0" is used. Therefore, the term "Industry 4.0" is explained and the advantages and disadvantages of applying each approach on industrial manufacturing robots is discussed.

Future work could focus on comparing more testing approaches to get a better result. The case study could be more specialized since there are a lot of different industrial robots in use. Furthermore, the developed approaches could be applied to different use cases. SOAS are spreading rapidly in a lot of areas of application which all have to be tested extensively. One example for another area are autonomous cars where the safety aspect is again really important. A second example would be the diving robot "ABYSS" which is able to move autonomously on the ground of the sea to fulfill a certain purpose.

References

[1] R. L. Glass, *Facts and Fallacies of Software Engineering*. Addison Wesley, 2002.

[2] C. Müller-Schloer, C. von der Malsburg, and R. P. Würtz, "Organic computing," *Informatik-Spektrum*, vol. 27, no. 4, pp. 332–336, Aug 2004. [Online]. Available: https://doi.org/10.1007/s00287-004-0409-6

[3] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck, "Observation and control of organic systems," in *Organic ComputingA Paradigm Shift for Complex Systems*. Springer, 2011, pp. 325–338.

[4] C. Müller-Schloer and S. Tomforde, *Organic Computing–Technical Systems for Survival in the Real World*. Springer, 2017.

[5] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997. [Online]. Available: http://www.worldcat.org/oclc/61321007

[6] S. Tomforde, B. Sick, and C. Müller-Schloer, "Organic computing in the spotlight," *arXiv preprint arXiv:1701.08125*, 2017.

[7] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41 – 50, 02 2003.

[8] D. Tennenhouse, "Proactive computing," *Commun. ACM*, vol. 43, no. 5, pp. 43–50, May 2000. [Online]. Available: http://doi.acm.org/10.1145/332833.332837

[9] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, and H. Schmeck, "Organic control of traffic lights," in *Autonomic and Trusted Computing*, C. Rong, M. G. Jaatun, F. E. Sandnes, L. T. Yang, and J. Ma, Eds. Springer Berlin Heidelberg, 2008, pp. 219–233.

[10] J. Schumann and W. Visser, "Autonomy software : V & v challenges and characteristics," 2005.

[11] C. D. Nguyen, A. Perini, C. Bernon, J. Pavón, and J. Thangarajah, "Testing in multi-agent systems," in *Agent-Oriented Software Engineering X*, M.-P. Gleizes and J. J. Gomez-Sanz, Eds. Springer Berlin Heidelberg, 2011, pp. 180–190.

[12] C. Nguyen, "Testing techniques for software agents," Ph.D. dissertation, University of Trento, 12 2008.

[13] P. Helle, W. Schamai, and C. Strobel, "Testing of autonomous systems - challenges and current state-of-the-art," 2016.

[14] J. Mishra and A. Mohanty, *Software Engineering*, ser. Always learning. Dorling Kindersley, 2011. [Online]. Available: https://books.google.ie/books?id=YnGz2ghKF-gC

[15] H. Heck, S. Rudolph, C. Gruhl, A. Wacker, J. Hähner, B. Sick, and S. Tomforde, "Towards autonomous self-tests at runtime," in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Sept 2016, pp. 98–99.

[16] B. Eberhardinger, H. Seebach, A. Knapp, and W. Reif, "Towards testing self-organizing, adaptive systems," in *Testing Software and Systems*, M. G. Merayo and E. M. de Oca, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 180–185.

[17] F. Nafz, H. Seebach, J.-P. Steghöfer, G. Anders, and W. Reif, "Constraining self-organisation through corridors of correct behaviour: The restore invariant approach," in *Organic Computing - A Paradigm Shift for Complex Systems*, 2011, pp. 79–93.

[18] G. Püschel, C. Piechnick, S. Götz, C. Seidl, S. Richly, and U. Assmann, "A black box validation strategy for self-adaptive systems," 2014.

[19] D. Lucke, C. Constantinescu, and E. Westkämper, "Smart factory - a step towards the next generation of manufacturing," in *Manufacturing Systems and Technologies for the New Frontier*, M. Mitsuishi, K. Ueda, and F. Kimura, Eds. Springer London, 2008, pp. 115–118.

[20] L. A. Kirschgens, I. Z. Ugarte, E. Gil-Uriarte, A. M. Rosas, and V. M. Vilches, "Robot hazards: from safety to security," *CoRR*, vol. abs/1806.06681, 2018. [Online]. Available: http://arxiv.org/abs/1806.06681