Test Agents: The Next Generation of Test Cases

Eduard Enoiu, Mirgita Frasheri
Email: eduard.enoiu@mdh.se, mirgita.frasheri@mdh.se
Mälardalen University, Västerås, Sweden.

***Abstract*—Growth of software size, lack of resources to perform regression testing, and failure to detect bugs faster have seen increased reliance on continuous integration and test automation. Even with greater hardware and software resources dedicated to test automation, software testing is faced with enormous challenges, resulting in increased dependence on centralized and complex mechanisms for automated test case selection as part of continuous integration. These mechanisms are currently using static entities called test cases that are concretely realized as executable scripts. Our key vision is to provide test cases with more reasoning, adaptive behavior and learning capabilities by using the concepts of software agents. We refer to such test cases as test agents. The model that underlie a test agent is capable of flexible and autonomous actions in order to meet overall testing objectives. Our goal is to increase the decentralization of regression testing by letting test agents to know for themselves when they should be executing, how they should update their purpose, and when they should interact with each other. In this paper, we envision test agents that display such adaptive autonomous behavior. Existing and emerging developments and challenges regarding the use of test agents are explored—in particular, new research that seeks to use adaptive autonomous agents in software testing.**

I. Introduction

Even if software testing is widely used in industry for verification and validation, in many cases due to the increased use of continuous integration and the sheer amount of test cases created, automation becomes a bottleneck in software development and is expensive to perform in a cost-efficient manner. Several such challenges have been identified in the automated regression testing of complex software systems [1], [2]: costly scheduling of test cases, badly prioritized test suite, and forgotten test cases. Automated testing is the process of designing, continuously executing and maintaining the confidence in the system dependability in a cost-effective and automated manner. In this context, test cases are created by human testers satisfying different test requirements and domain needs, are scripted and executed automatically and repeatedly. These test cases contain some mechanism for test evaluation that is embedded in a test script. Traditional regression test selection mechanisms are not designed to exhibit capabilities of responsiveness, flexibility, robustness and re-reconfigurability, since they are built upon centralized systems that strive to achieve overall test suite optimization, but have a weak and rigid response to complexity and changes at runtime. Such centralized regression testing mechanisms normally lead to situations where test cases are not adapting, resulting in inefficient and costly test scheduling mechanisms. In these circumstances, the current challenge is to develop collaborative and reconfigurable test cases that support characteristics of adaptation and autonomy.

In this paper we outline our vision *to decentralize the control over regression testing by developing tests cases that are capable of autonomous and adaptive actions*. Such test cases are named *test agents*. We envision the use of a test agent as a self-contained and self-aware test case capable of interactions with other test agents. These test agents represent another way for an engineer to design test cases that will effectively test software. The use of test agents tackles these test automation challenges by enabling test engineers to create autonomous and adaptive test cases which can take decisions about their action-execution mechanism and scope of interactions at runtime.

Many possible definitions of agents exist in the literature [3]. Here we explicitly consider that agents are software systems that operate in an execution environment which they can perceive and respond to, take initiatives and select own goals, and interact with others when deemed fit [4]. Over the past few years, this paradigm has been applied in different application domains, with varying levels of product maturity [5]. These solutions are being deployed in the telecommunication, logistics, e-commerce, and robotics domain. Having learned from the successes and drawbacks of using agents in other domains, our vision is to explore the test agent paradigm. Practically, our vision goals are to: (i) create test cases capable of adaptive and autonomous actions using test agents with a specific purpose in terms of test effectiveness and efficiency, a set of interaction and execution mechanisms, and the ability to perceive the test evaluation results after each run and (ii) investigate how test agents and their interactions evolving in time could be represented.

II. Regression Testing

Software testing is the primary method used in industrial practice to evaluate software and can be divided [6] in three distinct tasks: test design, test execution and test evaluation. A test engineer designs tests by creating test requirements which are then written into actual scripts that are ready for execution. These scripts are executed against the software and the results are evaluated. Test automation is using software to control these activities with the aim to reduce the cost of testing. One integral part of test automation is regression testing, the process of continuously testing software that has been modified. A regression test system (shown in Figure 1) is often incorporated into a continuous integration development and determines which test cases to include in a regression suite

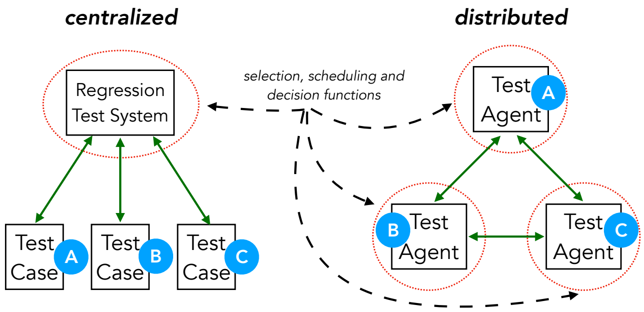

Fig. 1: Centralized and distributed approaches to regression test automation.

by identifying suitable cases based on different information sources (i.e., fault history, execution time, test coverage, failing tests) obtained after the execution of the system. In the current practice of software testing, test cases are entities composed of several discrete parts (i.e., test case input values and expected results needed for evaluation). These components are concretely realized in a script that can be automatically executed and knows exactly what values to expect. As a result, the existing process of software testing is build upon static test cases, thus entailing the use of a highly-complex and centralized test scheduling technique for regression testing. To change this centralized process, we envision a new class of autonomous and adaptive test cases that we refer to as test agents. As a result, test agents could enable testing of goals beyond their original scope and can decide what interactions are needed with other test agents and adapt when their test goal or the software updates. The change from the traditional centralized approach to regression testing to the new distributed and adaptive approach is illustrated in Figure 1.

III. Adaptive Autonomy

The notion of adaptive autonomy refers to the ability of software agents to change their levels of autonomy based on their circumstances. Agent autonomy in itself can be described through two dimensions, self-sufficiency, i.e., ability to fulfill a task without outside help, and self-directness, i.e., ability to decide upon one's own goals [7]. Castelfranchi [8] uses dependence theory to define autonomy as follows: An agent A that lacks means for performing a specific task T and depends on an agent B to acquire such means, is said to be non-autonomous from B with respect to T. It might happen that A is able to perform T by itself at a point in time t_1, but not at t_2 due to circumstantial changes, e.g. A is low on resource-consumption levels. Consequently, A (and B as well) needs to continuously evaluate whether it needs assistance, or whether it is willing to give assistance to other agents that might ask. As a result, based on their circumstances, agents decide by themselves when to adapt their autonomy. Alongside adaptive autonomy, there are other similar notions such as adjustable autonomy [9] [7], mixed-initiative interaction [9], collaborative control [10] and sliding autonomy [11].

IV. Agents in Software Testing

Agents have already been used to automate different aspects of testing. One such approach is the adaptive test management system (ATMS) [12], which aims at selecting an appropriate set of test cases to be executed in every test cycle using test unit agents and fuzzy logic. Researchers have also used a multi-agent approach for intra-class testing of object-oriented software. Dhavachelvan [13], [14] presented three types of agents: distributor agent, testing technique agent, and clones. Distributor agents take assignments and map them to the available testing agents. Other contributions have relied on the agent-based paradigm to specifically target service-oriented systems [15]. The Belief Desire Intention (BDI) agent architecture is used by Rao *et al.* [16] to distinguish between two types of agents: coordinators and runners. Coordinators create testing plans and runners conduct the testing activities and send their results back to the respective coordinator. Hong Zhu [17] is using the agent-paradigm in a framework that targets both software development and management. Zhang *et al.* [18] extended the LoadRunner testing platform for web services using IBM Aglet agents. LoadRunner enables the simulation of users by executing tests on the remote server hosting the service by using Aglets agents. A different approach is presented in the work of Tang *et al.* [19]. Their study aims at automating the whole testing life cycle by using four types of agents: requirement agent, construct agent, execution and report agent.

V. Defining a Test Agent

Our overall vision for the use of *test agents* is *to shift the bulk of continuous test selection, prioritisation and scheduling from a centralized regression test automation framework to a lower level of abstraction where test agents can decide by themselves how and what to execute*. An adaptive autonomous agent proposed by Frasheri et al. [20] has been modified to fit the purposes of a testing agent. Such an agent can be described from two perspectives, its overall internal operation, and the interaction mechanisms which enable adaptive autonomous behaviour.

A. A Test Agent Model

The test agent is composed of the following five states mirrored in Figure 2: *Idle*, *Interact*, *Execute*, *Regenerate*, and *Out of Order*. The test agent is not committed to anything in the *Idle* state and will execute its own task when needed. Once the execution task has been generated, or the test agent has gotten a task from another test agent, it will go to the *Execute* state and decide if it needs assistance from another test agent before and after its execution. After the execution is completed, the agent will go back to the *Idle* state. When the test agent receives a request from another agent, it will switch to the *Interact* state and will decide whether to accept the request and give assistance or discard it. When the test agent decides it cannot serve its initial purpose it will switch to the *Out of Order* state. Other triggers for switching to *Out of Order* could be devised if necessary. In addition, the agent can switch to the *Regenerate* state and a test case redesign takes place with the help of a test engineer. In the end, the test agent can return to the *Idle* state. For example, let us assume a test agent A has not been able to fulfill its original goal (e.g., achieving 100% branch coverage for a certain function) due to a code change. The agent will ask for assistance at runtime in the *Execute* state from another test agent. Test agent C receives the request, but decides to discard the request since its initial goal was to check the fulfillment of a certain requirement and its last execution is not affecting the logic that needs to be covered. Test agent B decides to accept the request since this new goal serves its initial purpose and goes to the *Execute* state and fulfills it.

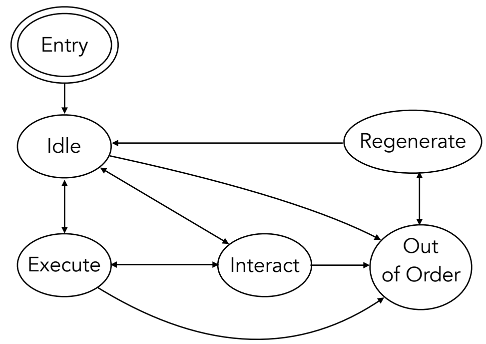

Fig. 2: A high-level test agent model [20].

B. Test Agent Interactions

The adaptive autonomous behavior is determined at those points in which the test agent decides whether to ask or give help, and is modeled through its willingness to interact. This willingness is composed of the disposition to give or to ask for help. Frasheri *et al.* [20] considered the different factors that could influence the willingness of agents to interact, while Van der Vecht *et al.* [21] examined the task urgency and agent dedication to the overall organization as a molder of adaptive behavior. Further studies are crucial for establishing a suitable adaptive autonomous behavior based on the application of test agents in realistic testing contexts.

In this paper, we propose to derive the adaptive behavior of test agents by adapting the following four levels of interactions between agents already identified by Frasheri *et al.* [22] to distributed regression testing: (i) non-committal interactions in which a test agent can broadcast information (e.g, its execution time, fault detection, test coverage) to the other test agents and no response is expected, (ii) one-to-one dialogue in which a test agent *A* asks another test agent *B* for information (e.g., its fault history) and a response is expected, (iii) one-to-one delegation which is used when a test agent *A* delegates a task, or a subtask (e.g., cover certain parts of the code) to a test agent *B* and a response is expected together with some execution evidence and information (e.g., the input parameters used during execution), and (iv) one-to-many dialogue/delegation in which two scenarios are considered: chain interactions and simultaneous interactions (in the former, a test agent *A* makes a request to a test agent *B*, which in turn makes a request to *C*; and whereas in the latter, test agent *A* makes several requests, one to test agent *B*, one to *C*). For example, the one-to-many interactions can be used to achieve a trade-off between multiple test agents and their objectives with regard to some test criteria and cost (e.g., maximize test coverage, minimize the execution time).

VI. Challenges

Reassessing the concept of a test case using a test agent representation is not an easy task to accomplish and therefore realizing our vision requires addressing the following challenges.

Test Design

When it comes to creating test cases there are at least two ways [6], [23]: criteria-based and human-based test design. The criteria-based test design is used for creating tests that satisfy some test requirement or coverage criterion. This process requires the creation of explicit test requirements and models. On the other hand, human-based test design is used for creating test cases based on the test engineer's domain-specific knowledge. When engineers create tests, they sometimes attempt to perform positive testing as well as stressing the software using unusual test cases. One challenge to this end is to provide precise guidance to test engineers on how to create a test agent in terms of its purpose, test case values, execution environment, perception capabilities and interaction actions with other test agents. The design of test agents is complicated by the test case heterogeneity given the large space of possible test scenarios and interactions with the software. In addition, a challenge is to define a language for describing the test agent including its perception capabilities, interaction rules for actions, test purpose and test agent hierarchy.

Test Automation

We refer to a test being automated if its execution, evaluation and reporting is controlled by software. As an example, when dealing with test agents, test automation necessarily has to consider a standardized design for test scripts, and should include support for a test execution driver. This driver should be used by each test agent for executing the software, evaluate the results of its execution and report the results back to the test agent. A challenge is to establish a test automation framework that supports (i) the ability to share test data and interaction information among test agents, (ii) the ability for test agents to easily organize and run, and (iii) statistical assertions to evaluate the multi-dimensional information perceived from logs and reports. Clearly, automated support for maintaining test agents is crucial for the success of such an approach.

Regression Testing

Software is subject to frequent modifications. Regression testing is the process of continuously testing modified software. Its purpose is to ensure that software is functionally equivalent to the version before the updates. For example, regression testing can reveal if mistakes in requirements are implemented in the software. The use of regression testing can result in a test suite that is too large to manage and does not finish to execute in a timely manner. For test agents, regression testing is associated with the interaction between agents and their evolution in time. Evolving a test agent is challenging because of more complex dependencies. The adaptive autonomous behavior of test agents is modeled through its willingness to interact with other test agents. This interaction should be based on local built-in preferences that are deciding what to do next and initiate actions during runtime.

VII. Overall Objectives and Conclusion

The goal of this work is to apply the adaptive autonomous agent paradigm to the software testing domain in order to reassess the notion of a test case. The vision proposed in this paper is expected to lead to an operational definition of a test agent. Such agents need to continuously reason and decide on their need for help or when to assist other test agents in different circumstances. In order to validate the proposed vision, the following steps need to be taken: (i) select a platform in which to develop the test agent automation system (i.e., using several agent-based technologies are available such as JADE, NetLogo, SeSAm [24]), (ii) analyze and simulate (e.g., using ROS (Robot Operating System) [25]) how test agent interactions are shaped by a test engineer's preferences and define how interactions between different agents are represented, and (iii) investigate different learning techniques that can help the test agent refine its decision-making process and evolution in time. An advanced capability that can be added to test agents is learning such that they retain useful information from their interactions as training data and utilize various machine learning techniques to adapt to new execution scenarios and improve their performance.

Acknowledgment

This work is partially funded from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No. 737494, The Swedish Innovation Agency, Vinnova (MegaM@Rt2 and XIVT), the DPAC research profile funded by KKS (20150022) as well as the UNICORN project, VINNOVA FFI.

References

[1] P. E. Strandberg, W. Afzal, T. J. Ostrand, E. J. Weyuker, and D. Sundmark, "Automated system-level regression test prioritization in a nutshell," *IEEE Software*, vol. 34, no. 4, pp. 30–37, 2017.

[2] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming google-scale continuous testing," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*. IEEE Press, 2017, pp. 233–242.

[3] S. Franklin and A. Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1996, pp. 21–35.

[4] N. R. Jennings and M. J. Wooldridge, "Applications of intelligent agents," in *Agent Technology*. Springer, 1998.

[5] J. P. Müller and K. Fischer, "Application impact of multi-agent systems and technologies: A survey," in *Agent-oriented Software Engineering*. Springer, 2014, pp. 27–53.

[6] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2016.

[7] M. Johnson, J. Bradshaw, P. Feltovich, C. Jonker, B. Van Riemsdijk, and M. Sierhuis, "The fundamental principle of coactive design: Interdependence must shape autonomy," in *Coordination, organizations, institutions, and norms in agent systems VI*. Springer, 2011, pp. 172–191.

[8] C. Castelfranchi, "Founding agent's 'autonomy' on dependence theory," in *Proceedings of the 14th European Conference on Artificial Intelligence*. IOS Press, 2000, pp. 353–357.

[9] B. Hardin and M. Goodrich, "On using mixed-initiative control: A perspective for managing large-scale robotic teams," in *Proceedings of the 4th International Conference on Human Robot Interaction*. ACM, 2009, pp. 165–172.

[10] T. Fong, C. Thorpe, and C. Baur, *Collaborative control: A robot-centric model for vehicle teleoperation*. Carnegie Mellon University, The Robotics Institute, 2001, vol. 1.

[11] J. Brookshire, S. Singh, and R. Simmons, "Preliminary results in sliding autonomy for assembly by coordinated teams," in *Proceedings of the International Conference on Intelligent Robots and Systems*, vol. 1. IEEE, 2004, pp. 706–711.

[12] C. Malz and N. Jazdi, "Agent-based test management for software system test," in *International Conference on Automation Quality and Testing Robotics (AQTR)*, vol. 2. IEEE, 2010, pp. 1–6.

[13] P. Dhavachelvan, G. Uma, and V. Venkatachalapathy, "A new approach in development of distributed framework for automated software testing using agents," *Knowledge-Based Systems*, vol. 19, no. 4, pp. 235–247, 2006.

[14] P. Dhavachelvan and G. Uma, "Multi-agent-based integrated framework for intra-class testing of object-oriented software," *Applied Soft Computing*, vol. 5, no. 2, pp. 205–222, 2005.

[15] X. Bai, B. Chen, B. Ma, and Y. Gong, "Design of intelligent agents for collaborative testing of service-based systems," in *Proceedings of the 6th International Workshop on Automation of Software Test*. ACM, 2011, pp. 22–28.

[16] A. Rao, M. Georgeff *et al.*, "Bdi agents: From theory to practice." in *International Conference on Manufacturing Systems*, vol. 95, 1995, pp. 312–319.

[17] H. Zhu, "Cooperative agent approach to quality assurance and testing web software," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, vol. 2. IEEE, 2004, pp. 110–113.

[18] J. Zhang and D. Xu, "A mobile agent-supported web services testing platform," in *International Conference on Embedded and Ubiquitous Computing*, vol. 2. IEEE, 2008, pp. 637–644.

[19] J. Tang, "Towards automation in software test life cycle based on multi-agent," in *International Conference on Computational Intelligence and Software Engineering (CiSE)*. IEEE, 2010, pp. 1–4.

[20] M. Frasheri, B. Çürüklü, and M. Ekström, "Analysis of perceived helpfulness in adaptive autonomous agent populations," *LNCS Transactions on Computational Collective Intelligence*, 2017.

[21] B. van der Vecht, F. Dignum, and J. C. Meyer, "Autonomy and coordination: Controlling external influences on decision making," in *International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, vol. 2. IEEE, 2009, pp. 92–95.

[22] M. Frasheri, B. Çürüklü, and M. Ekström, "Towards collaborative adaptive autonomous agents." in *ICAART (1)*, 2017, pp. 78–87.

[23] S. Eldh, "On test design," Ph.D. dissertation, Mälardalen University, 2011.

[24] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, no. 1, p. 11, 2015.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.