A framework for automated testing of automation systems

Conference Paper · October 2010

DOI: 10.1109/ETFA.2010.5641264 · Source: IEEE Xplore

CITATIONS
18

READS
4,776

4 authors:

Dietmar Winkler
TU Wien
198 PUBLICATIONS **1,650** CITATIONS
SEE PROFILE

Reinhard Hametner
Thales Group, Austria
26 PUBLICATIONS **307** CITATIONS
SEE PROFILE

Thomas Östreicher
TU Wien
15 PUBLICATIONS **80** CITATIONS
SEE PROFILE

Stefan Biffl
TU Wien
440 PUBLICATIONS **5,801** CITATIONS
SEE PROFILE

A Framework for Automated Testing of Automation Systems

Dietmar Winkler[1] Reinhard Hametner[2] Thomas Östreicher[1] Stefan Biffl[1]

[1]Institute of Software Technology, Vienna University of Technology
Favoritenstr. 9-11/188, AT 1040 Vienna, Austria
{dietmar.winkler, thomas.oestreicher, stefan.biffl}@qse.ifs.tuwien.ac.at

[2]Automation and Control Institute, Vienna University of Technology
Gußhausstr. 27-29/E376, AT-1040 Vienna, Austria
Hametner@acin.tuwien.ac.at

Abstract

Increasing complexity of software components in automation systems require systematic and frequent testing approaches. Test-First Development (TFD) – an established approach in business IT software development – promises to support test automation in automation systems development. Nevertheless, linking test case generation, execution, and reporting requires a sound framework to support testing processes more efficiently. In this paper we present a framework for automating test processes based on UML models and TFD. Applying this framework in prototype applications in industry environment identified the framework as promising candidate to improve automation systems development and product quality.

Keywords. Automation Systems Development, UML, Test-First Development, Testing Process.

1. Introduction

Required flexibility of modern automation systems development, e.g., responding to changing requirements, reusing parts of prior automation systems, and the need for configuring automation systems with respect to individual customer requirements, lead to implementing functional behavior in software components rather than in hardware [10]. We learned from automation systems engineering processes at our project partners, that there is limited experience on systematically developing and testing software components. Nevertheless, software testing is a core aspect and needs to be addressed properly [3]. Based on our observations we identified software testing and automated software testing processes as a promising research area in the automation systems domain.

The main research question is *how we can introduce efficient software testing and test automation approaches in automation systems development*.

Based on our experiences in business IT software development we see four major aspects of software testing embedded within an engineering process [12], which should be addressed in automation systems development: These aspects of testing include (a) test planning, organization, and observation, (b) test case generation, (c) test case implementation and execution, and (d) test reporting on various levels. This paper focuses on technical aspects of software testing rather than on organizational aspects.

Acceptance and systems tests focus on customer requirements and business cases from end user perspective (top level). Integration tests focus on the architectural design of the system, individual components, and the interaction between components, and unit tests (on the lowest, most hardware related level) include testing of individual components [12]. In this paper we present a framework for automated testing of industrial control automation applications and discuss individual aspects of a basic testing process and how these aspects address various systems levels [5]. Parts of the framework were already implemented in a set of small pilot applications at our project partners.

The remainder of this paper is structured as follows: Section 2 presents related work in context of the testing framework, i.e., (a) how to apply test driven development in context with automation systems, (b) how to derive test cases systematically, and (c) how to execute test cases and report results of individual tests and test scenarios. Section 3 discusses the basic research issue and Section 4 presents the proposed testing framework. We briefly summarize the pilot applications in Section 5. Finally, Section 6 concludes and identifies a set of further work.

2. Related Work

This section briefly summarizes background information related to automated testing based on TFD in the automation systems domain.

Test-First development (TFD) is a wide-spread constructive software development practice in agile development processes [4]. Test cases are defined prior to the implementation based on customer requirements and specification documents. Nevertheless, because of a lack of implementation all test cases must fail at the beginning. During implementation frequent test runs enable immediate feedback on the current state of development and the project. Additional advantages of defining test cases early include a more clear understanding of the requirement/specification aspects and the knowledge on how to test individual requirements. Mockup-objects can help simulating non-existing functional behavior [7][11] regarding system parts not implemented at test case definition time.

Different views on the system. Various test levels support different views on the system [5][9][12]. Acceptance and system tests focus on customer requirements, typically represented by workflows and scenarios. Test scenarios, i.e., a sequence of individual test cases, can help addressing system level tests. A more detailed view on the system is supported by integration tests which cover architecture aspects, components, and the interaction of components (within a subsystem). Again, scenarios can help focusing on (parts) of the architecture. On the most detailed level, unit test help addressing individual components.

Systematic test case generation. Nevertheless, one question is how to derive test cases and test scenarios on various levels systematically. Models, e.g., the Unified Modeling Language (UML), can help addressing individual aspects of the system [2][3][8]. UML provides two basic classes of diagrams (a) 6 structural diagrams to capture static system structures, e.g., deployment and component diagrams and (b) 7 behavior diagrams to capture system dynamics, e.g., use cases, state charts, and activities. In addition behavior diagrams include 4 interaction diagrams, e.g., sequence charts and timing diagrams [1]. An important finding of our previous research [5] was that test cases can be derived directly (and automatically) from behavior and interaction diagrams. Thus, models are most promising candidates to (a) present the static structure of the system and (b) capture test cases for systematically [5].

Test case execution and reporting. The next step after defining test cases and test scenarios focuses on test execution and the report of test results. A test runner provides the required infrastructure and defines the preconditions for test case execution, e.g., sending individual tests cases to the target system sequentially and capturing test responses from the target [11]. Preconditions are required to enable repeatability of test runs under similar conditions, i.e., repeatable results of tests. These results enable immediate feedback (if conducted automatically) on the individual test cases, test scenarios, and requirements.

Our previous research focused on providing an automation component for "Test-Driven Automation", including testing and diagnosis aspects [6][11] and the application of TFD in context of test case generation [5][12]. Nevertheless, a more detailed view on the test framework remains open.

3. Research Issue

Enabling automated testing during automation systems development requires a sound framework that enables early testing approaches (TFD), automation supported systematic test case generation, execution, and reporting on various levels of detail and from different perspectives. Based on initial findings [12], a more detailed testing framework is required to enable automated test processes. An important precondition is how individual test steps, i.e., definition, execution and reporting can be connected to each other. The research issue is how to operationalize this framework.

4. Solution approach

Following the reported testing process approach [5], the testing framework (see Figure 1) consists of four important interlinked components, i.e., (a) *test suite* (set of test cases and test scenarios), (b) *test runner* (monitoring and controlling test case execution, (c) *software under test & test fixture* on the target system, and (d) *test reports* to summarize test responses.

Test Suite. Based on structured and prioritized customer requirements and specification aspects test cases are defined based on UML models. Static diagrams, e.g., deployment and component diagrams, help identifying the static structure of the system and enables systematic scoping of test cases. Behavior and interaction diagrams model system dynamics and enable test case generation [5] on various levels directly; test scenarios include a set of test cases to capture and test systems behavior by means of expected workflows. Models can enable automated test case generation [3] [5]. Individual tests and scenarios are transferred to the test runner in sequence.

The *test runner* manages test case execution and reporting. We applied a spreadsheet solution as a prototype application to organize test scenarios (and individual test cases). Providing a test fixture, i.e., a test environment [6] on the target system, the test runner uses parameters for test case initialization in order to execute individual test cases (based on defined test scenarios). Note that capturing time stamps on the target systems enables measuring the temporal behavior of test case execution. After test case execution results are captured (on individual test case level) and aggregated to test

scenarios for reporting purposes. Note that reporting can include coverage analysis (i.e., share of requirements covered by test cases and scenarios). In addition test diagnosis data, derived from the target system are available in order to analyzing systems behavior in more detail (based on log-files), e.g., in error cases.

The *software under test* (SuT) and *test fixture* includes three aspects based on the Test-Driven-Automation (TDA) Architecture [6][11]: (a) automation aspects, representing the functional behavior of the system under construction; (b) testing aspects to set the system in a predefined state of operation; and (c) diagnosis aspects to capture and measure systems behavior.

We used an automation system development platform for the prototype implementation of TDA. Test and diagnosis results are captured and transferred to the test runner for further evaluation.

Test reports (organized by the test runner) enable the analysis of individual tests (e.g., on unit level) and test scenarios (e.g., architecture and system level). Linking test case and test scenario results and customer / system requirements can enable project monitoring and control based on TFD [12]. Figure 1d presents a sample test report for a set of frequent test runs (e.g., during regression testing) for a set of requirements and derived individual test runs.

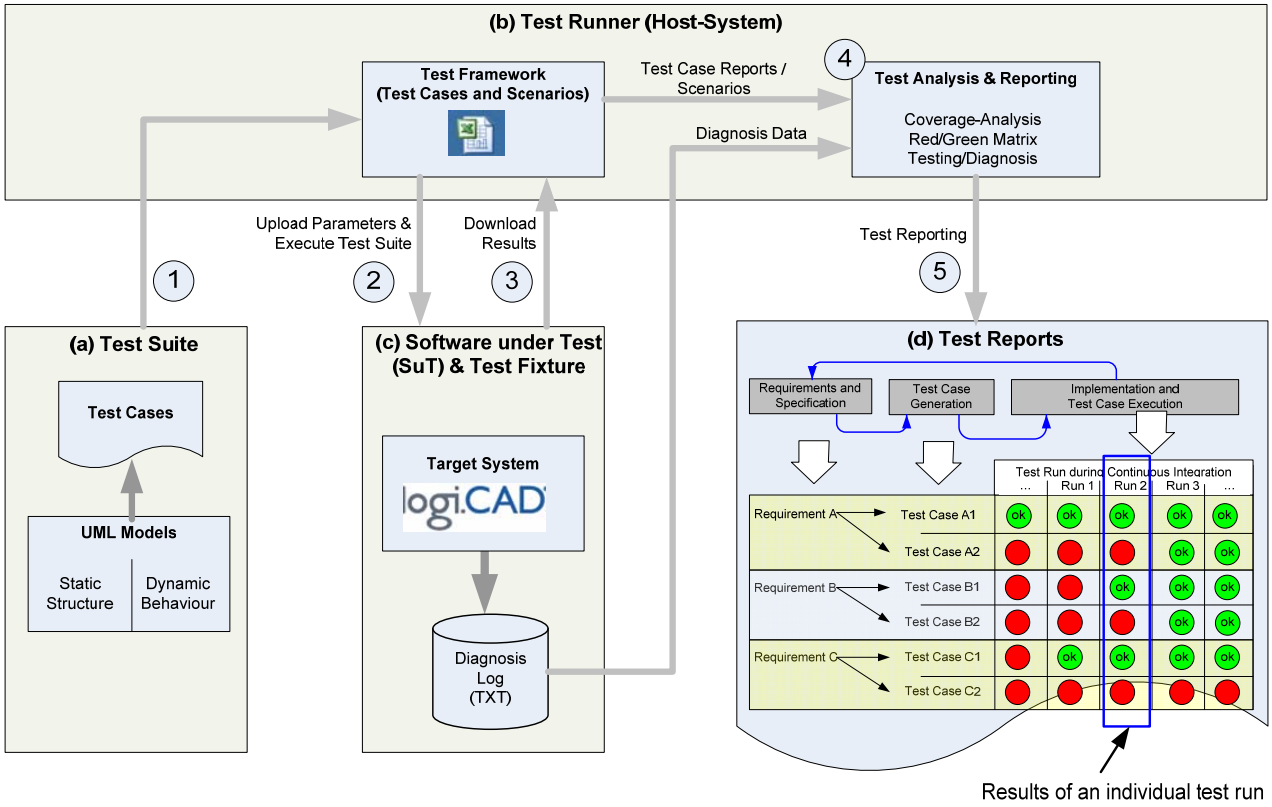

Figure 1: Framework for Automated Testing in Automation Systems Development.

5. Prototype Applications & Limitations

Test-Driven Automation (TDA), i.e., supporting automation systems engineering with automated testing approaches, is a promising area of research and most valuable for industry. Thus we developed the proposed testing framework, embedded within an engineering process approach and evaluated isolated parts of the framework in prototype applications.

- *Process support*. Engineering processes support projects by providing a defined framework for project planning and execution. We used the V-Modell XT[1] as underlying process approach for the pilot application of TFD. To illustrate test case generation of various levels based on UML we used a small sorting application [12].
- *Test-Driven Automation Component*, i.e., encapsulating functional, testing, and diagnosis aspects, enable a separated (but interconnected) configuration of individual aspects [6][11]. We used a simple triangle generator component to show the feasibility of the TDA component architecture using function block diagrams provided by logi.cad[2] [11]. Based on the experiences of TFD and the tri-

[1] V-Modell XT documentation: http://www.v-modell-xt.de

[2] Logi.cad descripton: http://www.logicals.com/products/logi.CAD/

angle generator pilot application we refined the TDA architecture and applied the concept in a bottle sorting application [6]. We learned from this more complex (but still simple) example that the application of TDA in context of automation systems development and the concept of TFD increases product quality.

- *(Automated) test case generation* requires a systematic test process to identify the static structure and the dynamic aspects of automation systems. We developed a test case definition process and evaluated this process in a feasibility study regarding an automated application to control a simple irrigation system *(waterworks)* [5].

The implementation of individual parts of the testing framework using a set of small pilot applications confirmed the feasibility of the proposed framework. Nevertheless, limitations include the scalability of the proposed framework and related aspects, e.g., applying UML models to capture system structure and behavior. Additionally, the spreadsheet solution (as a test runner) will include strong limitations regarding capturing timing information (e.g., real-time constraints).

6. Conclusion and Future Work

The results of the pilot application confirmed our expectations that Test-Driven Automation including process support based on test-first development, the TDA component structure (functional, testing, and diagnosis aspects) and the testing framework, is a promising approach in automation engineering to increase process, project, and product quality.

Nevertheless, a set of research issues remain open for further work:

(a) *Connecting individual TDA aspects*. Previous research focused on isolated pilot applications with strong focus on an individual aspect of TDA. Thus, a next step is to connect these individual findings to a comprehensive view with respect to processes and the testing framework to enable automated and systematic testing.

(b) *Scalability and timing considerations*. Small sample applications confirm our expectations that TDA can support automation systems engineering. Nevertheless, investigating the scalability of TDA and timing considerations with focus on the test runner of the proposed testing framework is part of future work.

(c) A next step of our research is an *empirical evaluation* of the TDA approach with respect to showing the benefits in real world application in an industrial setting in a larger development project together with our industry partner.

Acknowledgements

We want to thank our partners from academia and industry in the logi.DIAG project for their valuable discussions and feedback. Parts of this work were funded by the Austrian Research Funding Agency (FFG) grant logi.DIAG (Bridge7-196929).

References

[1] Ambler S.: Elements of UML 2.0 Style, Cambridge University Press, 2005.

[2] Baker P., Dai Z.R., Grabowski J.: Model-Driven Testing: Using the UML Testing Profile, Springer, 2007.

[3] Broy M., Jonsson B., Katoen J-P., Leucker M.: Model-Based Testing of Reactive Systems: Advanced Lectures, Springer, 2005.

[4] Damm L.-O., Lundberg L.: Quality Impact of Introducing Component-Level Test Automation and Test-Driven Development, Proc. EuroSPI, Springer, 2007.

[5] Hametner R., Winkler D., Östreicher T., Biffl S., Zoitl A.: The Adaptation of Test-Driven Software Processes to Industrial Automation Systems, Proc of the Int. Conf on Industrial Informatics (INDIN), 2010.

[6] Hametner R., Zoitl A., Semo M.: Automation Component Architecture for the Efficient Development of Industrial Automation Systems, (to appear) at 6th IEEE Conference on Automation Science and Engineering, Toronto, Canada, 2010.

[7] Karlesky M., Williams G.: Mocking the Embedded World: Test-Driven Development, Continuous Integration, and Design Patterns, Proc. Emb. Systems Conf, CA, USA, 2007.

[8] Köhler A. J., Nickel U., Niere J., Zündorf A., Integrating UML Diagrams for Production Control Systems, Proceedings of the 22nd Int. Conf. on Software Eng., Limerick, Ireland, p: 241 – 251, 2000, ISBN:1-58113-206-9.

[9] Lüder A., Peschke J., Reinelt D.: Possibilities and Limitations of the Application of Agent Systems in Control, Proc. Conf. On Concurrent Enterprising (ICE), Italy, 2006.

[10] Schäfer W., Wehrheim, H.: The Challenges of Building Advanced Mechatronic Systems. Int. Conf. on Software Engineering. IEEE, pp. 72-84, 2007.

[11] Winkler D., Hametner R., Biffl S.: „Automation Component Aspects for Efficient Unit Testing“, Proc of 14th IEEE Int. Conf. on Emerging Technologies and Factory Automation, Spain, 2009.

[12] Winkler D., Biffl S., Östreicher T.: „Test-Driven Automation – Adopting Test-First Development to Improve Automation Systems Engineering Processes“, Proc. of 16th EuroSPI, Industrial Track, Madrid, 2009.