Agile Testing Concepts Based on Keyword-driven Testing for Industrial Automation Systems

Reinhard Hametner
Automation and Control Institute
Vienna University of Technology
Gusshausstrasse 27-29/E376
A-1040 Vienna, Austria
hametner@acin.tuwien.ac.at

Dietmar Winkler
Institute of Software Technology
and Interactive Systems
Vienna University of Technology
Taubstummengasse 11/E188,
A-1040, Vienna, Austria
dietmar.winkler@qse.ifs.tuwien.ac.at

Alois Zoitl
Automation and Control Institute
Vienna University of Technology
Gusshausstrasse 27-29/E376
A-1040 Vienna, Austria
zoitl@acin.tuwien.ac.at

***Abstract*—In the field of industrial automation systems software becomes an important factor because engineers tend to move the realization of functional requirements from hardware to software components. The main reason for this is that software components allow increasing product flexibility. As a consequence software complexity increases rapidly and requires systematic, automation-supported and agile testing approaches. Thus, systematic and agile testing are key challenges in industrial control software development to ensure and improve systems quality. Further different implementation standards, i.e., IEC 61131-3 and IEC 61499, arise additional challenges in constructing and testing industrial automation systems software. This paper presents an agile and keyword-driven test approach with focus on testing implementations based on both important industrial standards and illustrates the applicability of the purposed approach in a sample implementation, i.e., a High Speed Pick and Place unit. Main results show the applicability of keyword-driven testing based on a defined subset of keywords (common for IEC 61131-3 and IEC 61499) and thus enable agile and automation-supported testing more effective and efficient.**

Keywords: Industrial Control Applications, IEC 61131-3, IEC 61499, Test Automation, Keyword-driven Test.

I. Introduction

Development projects in the industrial control domain tend to move the realization of functional behavior from hardware to software components to increase flexibility of the planned solution based on standardized (hardware) components. For example, adding or changing functional behavior without changing the underlying hardware components are typically implemented as added value of the system under development [1]. Nevertheless, shifting implementations of functional behavior from hardware to software components lead to more complex pieces of software and rise new challenges in software and systems testing. New and additional skills of electrical engineers (typically involved in hardware systems development projects) are required to face these more complex software artifacts.

Practical approaches from software engineering development [2] showed that test-first (test-driven) software development can help addressing tests on various levels, i.e., components, sub-systems, and systems [3], more effective and more efficient. The question is how systems and acceptance tests can help in efficiently testing system components.

Decoupled from traditional software testing approaches, i.e., implementing unit tests on code level, the keyword-driven test approach [4] aims at supporting electrical engineers in defining test cases more efficient and effective from user perspective without having highly sophisticated skills in software testing. Keyword-driven testing [4] focuses on a high abstraction level of individual test cases, i.e., based on customer specifications. Applying a (small) set of keywords can enable non-programmers in reading and writing test cases without knowing details on the implementation behavior of the System under Test (SuT). Additional benefits come from applying high-level tests (e.g., on systems or acceptance test level), which focus on customer and systems requirement without considering concrete implementation approaches. Automation support of test case execution can enable regression tests, i.e., test runs after changes, that become applicable during systems maintenance and evolution and can be adjusted to changing requirements.

Our observations in industry projects showed two important standards, i.e., IEC 61131-3 [5] and IEC 61499 [6], with respect to a cyclic execution and an event-based execution of the applications. Thus, an important question is how to design a test-framework for keyword-driven testing that supports both standards. The test-framework should be independent of the application based on common components; the technical implementation should be exchanged easily [7] [8].

In this paper we address two main research challenges with focus on (a) keywords definitions to support keyword-driven testing with respect to two common systems development standards and (b) system test framework to enable automation supported execution of test cases on systems and acceptance test level. In addition we demonstrate the feasibility of the purposed approach in an example implementation, i.e., a High Speed Pick and Place (HSPP) unit.

The remainder of this paper is structured as follows: Section II presents a test process overview based on Test-First Development (TFD) as strategy for early testing approaches in systems development processes and presents related work of the keyword-driven method. Section III defines the research

questions. Section IV presents the keyword-driven testing concept for industrial automation systems. Section V presents the implementation of the testing framework concept applying on a HSPP unit. Finally, Section VI summarizes, concludes, and identifies future work.

II. Related Work

This section presents related work on software testing based on software engineering development and introduces keyword-driven testing.

A. Test Process Overview

Traditional software and systems projects typically follow a defined development process based on traditional approaches, e.g., W-Model [9] or an agile (software) development approach in defined iterations, e.g., Scrum [10]. In these development processes software testing [2] is a key element, where various test levels are applicable: (a) unit and component-based testing, (b) integration testing with focus on the software and systems architecture, and (c) systems and acceptance testing with emphasis on the overall systems behavior and customer requirements. While unit and component tests focus on self-contained aspects of the system, integration tests focus on the interaction and communication of individual components (via interfaces) within a sub-system on architecture level [11]. Systems and acceptance tests focus on the overall systems behavior, i.e., interaction and communication of individual sub-systems and systems, and customer requirements. Note that acceptance tests typically emphasize on test scenarios, i.e., sequences of individual component and/or integration tests, to test the overall systems behavior. Typically, systems and acceptance tests play a crucial role during the commissioning phase of the system under development. Thus, automating these tests is a key challenge to reduce time and cost of the commissioning phase and increase systems quality from user perspective [12] [9] [13].

Test-Driven Development (TDD), also named Test-First Development (TFD) are common practices in agile software development practices [12] [10]. In a test-first strategy tests are typically defined prior or at least in parallel to the implementation which is based on component or sub-system specification or based on system requirements. Note that non-existing functions have to be mocked appropriately to simulate expected behavior [14]. Nevertheless, the TDD/TFD strategy helps in better understanding the system/component under development because engineers elicit test cases and test scenarios based on requirements and specification items; ambiguities and defects in requirements definitions can be removed early—an important benefit of applying test-first strategies.

In addition, tests have to address various aspects of the systems behavior [2]: (a) regular (positive) cases that refer to typical conditions of the system during regular operation, (b) special cases covering threshold values to test systems limits, and (c) error (negative) cases where the system fails but have to respond correctly, e.g., by changing to a defined error state (e.g., safe-state). These different test case classes have to be addressed by (a) defining test cases and/or sequences of test steps and (b) finding appropriate test values (data) to identify defects in the system. Note that different test cases can be included in the TFD approach early in the development process.

B. Keyword-driven Testing

Keyword-driven testing (KDT), also called table-driven testing or action-word testing, is a software testing method to support manual and automated testing processes in the field of software engineering. This technique is an extended form of the data-driven testing (DDT) method. The main idea is to express each test cases as abstractly as possible while making it precise enough to interpret and execute the test cases by a test tool [15].

This method separates the task of the test case implementation from the software development, i.e., separation of implementation work and test case definition/generation. Note that test cases can be defined early during a TFD approach. Maintainability increases because of this separation of test definition and functionality implementation. As the name KTD implies “keywords”, which control the processing, are in the focus in this approach. These keywords are dedicated in functions and in group of functions which include the executed actions of the test cases. Therefore, one keyword can include one or more test steps [16].

The open source project “Software Automation Framework Support (SAFS)” [7] focuses on implementing tools and frameworks for functional test automation. [7] presented and implemented keyword-driven engines for action-word testing in the field of software engineering, e.g., testing web applications. Tests are developed as data tables using specified keywords to execute the test process which are specific keywords for testing web applications. These keywords and engines are not applicable for testing industrial automation software.

KDT can be seen as a test language [4] and improves the communication between testers, avoid inconsistency in test documents, and to make an infrastructure for test automation available. Furthermore, [4] divided the KDT process in two main layers–Infrastructure Layer (KDT Engine) and the Logical Layer (KDT Test Case).

FitNesse[1] is an open source testing tool based on FIT[2] (Framework for Integration Testing) that enables test case definition on acceptance test level based on keywords without requiring sophisticated skills in test case implementation [17]. Thus, FitNesse is appropriate for communication and collaboration of engineers and non-technical users. A WIKI page holds the test case definition; a test fixture enables the communication with the implementation; results, i.e., whether a test case is executed successful or fails is represented within the WIKI page or within the development environment (e.g., Eclipse[3]).

[1] FitNesse: http://fitnesse.org/
[2] FIT: http://fit.c2.com/
[3] Eclipse: http://www.eclipse.org/

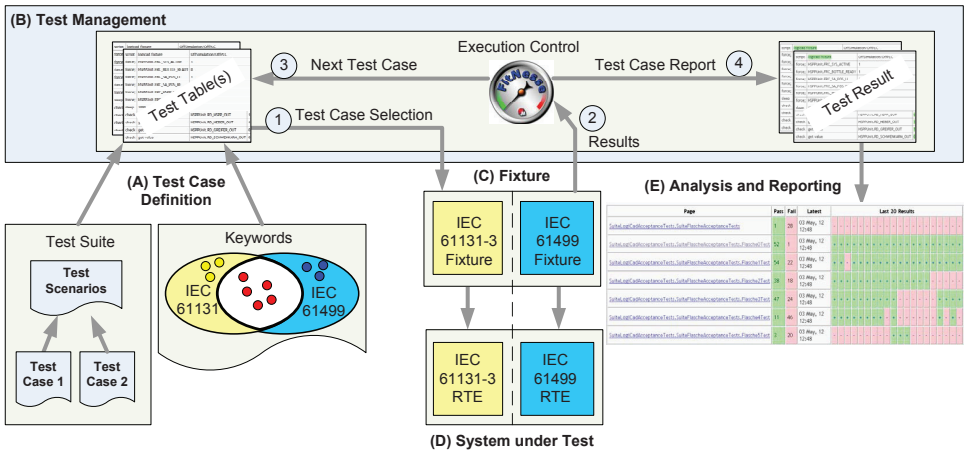

Fig. 1. Overview of the Framework for the Keyword-Driven Testing Process

KDT is used as testing technique in the field of traditional software engineering [18], but it is a novel approach to use this method in the field of industrial control software testing.

III. Research Questions

Based on related work and discussions with our industry partners we identified three important research questions to enable tool-supported test case definition and automated execution of test cases based on the keyword-driven test approach:

RQ1: What test infrastructure, i.e., test framework, is required to support keyword-driven tests?

Automation-supported test case definition, execution, and reporting requires an appropriate test framework [19] including (a) test suites holding all test scenario and test cases, (b) test runners that manages all test runs, (c) the SuT including a test fixture, and (d) mechanisms for test reporting. Nevertheless, this research question focuses on the design of the test framework to handle all individual tasks of the test process.

RQ2: What classes of keywords are required to enable testing of IEC 61499 and IEC 61131-3 applications?

Depending on the implementation standard, appropriate keywords have to be identified to address required actions and states. The main challenge is to identify a minimum common subset of keywords applicable to both standards. Additional - standard specific - features have to be considered and included/excluded if/if-not applicable.

RQ3: How can we demonstrate the feasibility of the keyword-driven approach in a real industrial control application?

The third research question focuses on the evaluation of the proposed approach to demonstrate feasibility and benefits. We applied a sample application, i.e., a High Speed Pick and Place (HSPP) unit to demonstrate applicability and feasibility of our approach in an industrial control application.

IV. Keyword-driven Testing Concepts for Control Systems

This section describes (a) the test-automation framework to enable semi-automated test case definition, automation-supported execution, and reporting and (b) the definition of keywords applicable for IEC 61499 and IEC 61131-3 applications.

A. Overall Test Framework for Keyword-Driven Testing

The test framework presented in Fig. 1 shows an overview for keyword-driven test case definition, execution, and reporting which includes four different building blocks:

(A) *Test Case Definition* representing individual test scenarios and test cases and a basic set of keywords according to implementation standards
(B) *Test Management* for coordinating and controlling test case execution
(C) *Fixture* for communication between Test Management System and System under Test (SuT)
(D) *SuT* Control System with respect to testing the SuT
(E) *Analysis and Reporting* is used to analyze the individual test runs including coverage report generation and aggregation of test results

The Test Suite is based on customer specifications. Individual test cases and test scenarios are constructed manually by applying the defined keywords and test data. Each test specification suite consists of defined test cases and test scenarios which are easy to configure and to edit by the Test Management system. Note that models, e.g., stated in [20], specified in specification documents can support automation-supported generation of test suites including test scenarios. Nevertheless, this automated approach is out of scope of this paper and will be done in future work.

Individual test scenarios and test cases are selected, e.g., based on value contribution of corresponding requirements, and executed by the corresponding test fixture, see item (1) at Fig. 1. Every test configuration requires an appropriate test Fixture related to the implementation standard. The Fixtures of IEC 61131-3 and IEC 61499 are different. Results of one test run, see item (2), are collected by the *Execution Control* component of the Test Management system. Note, we apply FitNesse as a concrete example for the *Test Management*

purposes. The Execution Control component included in the Test Management System is responsible for coordinating and controlling individual test runs, see item (3), and further for preparation of test result analysis, see item (4). Based on the results an analysis and reporting functionality is available for more detailed investigations, e.g., test coverage measures, quality metrics, and project observations which can be derived automatically.

B. Specification of the Keywords

The test case description by the KDT method should be defined formal as necessary so that test engineers understand the test specification easier. Furthermore, the test cases should be executed automatically. Therefore a KDT approach supports readable test cases by special defined keywords which are needed to control a test case for their execution, e.g., set value or get value.

An example command can be formulated as:

```
|KEYWORD;|Parameter1|Parameter2|
```

Therefore several keywords need to be defined, where the keyword vocabulary should be framework independent. This means that the keyword definition and specification should not be related or included in the Test Management system. Therefore a separate interface between Test Management system and SuT has to be defined, this interface is called "Fixture". The Fixture consists of all keyword specifications and the related behavior implementations. In our case we need two Fixture implementations for testing applications based on IEC 61131-3 and IEC 61499.

Needed keywords are to set values, read values, and compare results with the expected values. Through these general requirements several additional keywords are needed which are presented in the following.

First, the keywords are defined for testing IEC 61131-3 applications which are used as basis keyword-subset. Then the keyword-subset is extended for testing IEC 61499 implementations. Most of the defined keywords are applicable for both standards, but because of the event-based execution of IEC 61499 additional keywords have to be defined. TABLE I shows an overview of the defined keywords and the applicability for testing IEC 61131-3 and IEC 61499 implementations. The selection and definition of the keywords is done to get a minimum subset of keywords which are useful for testing industrial control applications. In further work a combination of basis keywords can be grouped to new keywords (e.g., `SystemStartUp` includes `startConn`, `create resource`, `create function block`, and `start`). Currently, only a basis set of keywords is defined.

The following new keywords are defined to test industrial automation systems:

- *startConn*: establishes a connection to the target system - SuT (IEC 61131-3 or IEC 61499)
- *force*: sets a new value to the variable with force-flag (force means that the application cannot overwrite the forced-variable-value, the forced value is set by the Test Management system)
- *set*: sets a new value to the variable without force-flag
- *sleep*: pauses the test for a number of milliseconds
- *get value*: reads the value from a specified variable
- *stopConn*: closes the connection to the target system and stops the application in the SuT
- *create resource*: creates a resource with the specified name (only in IEC 61499)
- *create function block*: creates a function block with a specified instance name (only in IEC 61499)
- *start*: starts the specified resource (only in IEC 61499)
- *create watch*: creates a watch property on an input/output variable, on a specified resource (only in IEC 61499)
- *trigger event*: trigger an event of a function block (only in IEC 61499)
- *script*: 'internal keyword function of the test management' to select the correct Fixture and additional target system parametrization for testing IEC 61131-3 and IEC 61499 applications can be done
- *check*: 'internal keyword function of the test management' to compare the resulting value with the stated value and presents a result (true or false)

Keywords	*IEC 61131-3*	*IEC 61499*
startConn	X	X
force	X	X
set	X	X
sleep	X	
get value	X	X
stopConn	X	X
create resource		X
create function block		X
start		X
create watch		X
trigger event		X
script*	X	X
check*	X	X

TABLE I
OVERVIEW OF THE USED KEYWORDS FOR TESTING INDUSTRIAL AUTOMATION SYSTEMS, (*... INTERNAL KEYWORD FUNCTION OF THE TEST MANAGEMENT)

V. EXAMPLE IMPLEMENTATION

In order to validate the presented keyword-driven testing approach for industrial automation systems a High Speed Pick and Place (HSPP) unit is used. The HSPP unit is part of a bottle sorting machine and is used to transport bottles from one conveyor belt to another conveyor belt. A detailed explanation of the used machine is presented in [21].

The implementation of the control software of the HSPP unit is done in both standards: in logi.CAD[4] based on IEC 61131-3 and in 4DIAC [22] which is based on IEC 61499. Therefore an appropriate comparison of the two different test case descriptions by keyword-driven testing can be done.

We used FitNesse as Test Management system infrastructure. Fitnesse is a standalone-WIKI based on Java which enables non-technical users to write tests for software testing.

[4]http://www.logicals.com/

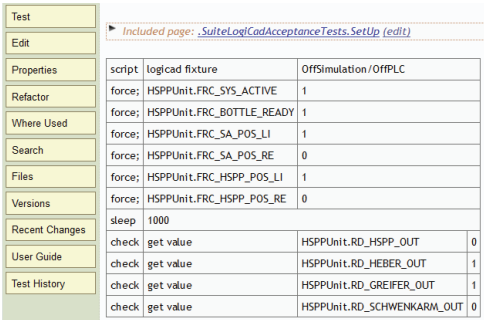

Fig. 2. FitNesse Test Case for IEC 61131-3 Application

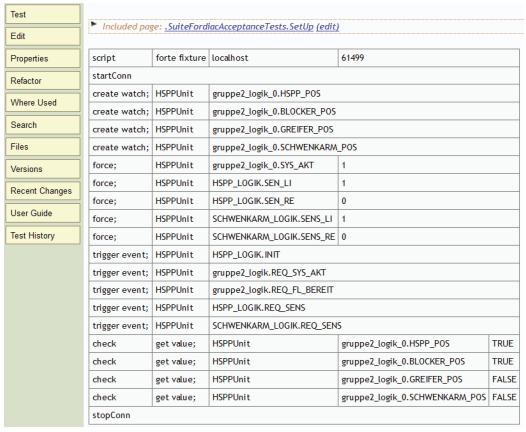

Fig. 3. FitNesse Test Case for IEC 61499 Application

Each site of this WIKI can be a static WIKI page which can be combined to test scenarios through a pre-defined order for executing the tests/pages. The properties of the specified tests can easyly be changed by editing the page through using the included 'Properties' link (define a test scenario by several pages), see Fig. 3. A test case itself can be changed by the 'Edit' link.

The test case specification with keywords is presented for IEC 61131-3 applications in Fig. 2 and for IEC 61499 applications in Fig. 3. These test case definitions communicates through the Test Management system and our developed "Fixture" with the SuT. In our case, the SuT is on the one hand the SoftPLC "logi.RTS" for IEC 61131-3 applications and on the other hand for IEC 61499 applications the runtime environment "4DIAC-RTE".

The communication from the Test Management system to the IEC 61131-3 based logi.RTS Programmable Logic Controller (PLC) is implemented by using COM-libraries on the local workstation (softPLC). This means that currently the testing process is executed not on a hardware PLC. However this is not required as hardware and software PLC behave the same. This allows applying test results for both platforms.

First of all, the SuT has to be running to test the implementation behavior. Fig. 2 shows one test case specification of an IEC 61131-3 function block implementation. The test case definition can be divided into four parts. The first part is the initialization which starts with the keyword `script`. Therefore the 'logicad fixture' is selected which connects to the PLC-Resource named 'OffSimulation/OffPLC'. The second part is to force the variables with defined values, which can be seen through the `force` keyword. Variables can only forced if the IEC 61131-3 application has been extended by so called "force marker". Force markers are a logi.CAD specific feature to write values from external (outside from the PLC system) into the application. The third part is the `sleep` command which stops the execution, in our case, for 1000ms. This command is only necessary in IEC 61131-3 based testing processes, because of the cyclic execution. It guarantees that a state is reached and will not be skipped by the execution machine of the runtime. The fourth and last part is the check of the expected value and the result value by using the `check` and `get value` keywords, e.g., `HSPPUnit.RD_HSPP_OUT=0`.

The communication process of the second Fixture implementation for testing IEC 61499 applications utilizes TCP communication, which directly enables remote testing. That means, that the KDT process is applicable for testing on real hardware PLCs. In opposite to the IEC 61131-3 test process, the IEC 61499 Fixture has additional keyword implementations, see Fig. 3. The `create watch` keyword enables to log variables of the IEC 61499 application which can be used for checking the results. That enables reading several variables which are watched at the same time. IEC 61499 has an event-based execution model, so that an additional `trigger event` keyword is implemented to trigger event inputs of function blocks which enables a detailed controlled execution of the application under test. Finally the `check` and `get value` keywords are used to consider the variable, e.g., `gruppe2_logik_0.HSPP_POS=TRUE`, in the defined IEC 61499 resource named 'HSPPUnit'. Therefore the resource name has to be defined, because IEC 61499 enables distributed control over several resources.

Additionally to execute a test case applied on the SuT, our IEC 61499-Fixture implementation enables to create a full IEC 61499 implementation on the 4DIAC-RTE before the test process starts. Several keywords are defined like `create resource`, `create function block`, and `start`. This allows applications modeled in an IEC 61499 development tool to be converted into a WIKI-syntax and then used as part of the KDT process. First the SuT will be created on the PLC and further the test cases will be executed to support a fast test execution flow.

Limitations of the proposed approach and the sample implementation include the consideration of temporal behavior (real-time) because we did not focus on time restrictions and control sequences in this paper.

Note that we applied the same example implementation to enable comparability of our approach with respect to both implementation standards, IEC 61131-3 and IEC 61499.

VI. Conclusion and Future Work

Increasing complexity of industrial control software requires a systematic testing approach to enable efficient and effective testing in case of changes. Keyword-driven testing (KDT) is a

valuable testing method to support such test requirements. This paper presents a test infrastructure which enables a systematic test process by using the KDT method.

RQ1: For the KDT a test framework is presented which supports automatic test case execution.

RQ2: Based on the requirements for testing industrial control software we specified and defined a basis subset of keywords which results that IEC 61131-3 needs less keywords. This has two reasons: (a) IEC 61131-3 has a cyclic execution model and therefore no `trigger event` keyword is necessary. (b) 4DIAC-RTE enables a dynamic creation of a function block network which needs several additional keywords like `create resource` and `create function block`.

RQ3: In order to validate the presented KDT method an industrial control application, a High Speed Pick and Place unit, is used to show the feasibility. Summing up, testing IEC 61499 applications by the KDT method is more applicable because the control applications under test (CAuT) need not be changed for the test process (i.e., IEC 61131-3 need force marker). Furthermore because of the standardized download interface of IEC 61499 the CAuT can be integrated in the Test Management system which enables an easier test flow. This testing method is applicable to test software units like IEC 61499 or IEC 61131-3 function block libraries, but it is also useful for testing complex industrial applications.

Future work will include the application of Model-Based Testing [23] [20] to enable an automation-supported generation of test cases as input for the test suite. Previous work [24] identified state charts and sequence charts to be appropriate approaches for automatically deriving test cases based on models. Several publications presented some combinations of structural testing to improve test case generation [25] and more formal models for modeling systems behavior [26] [27]. Considering the temporal behavior remains open but will be considered as future work, e.g., based on [28]. In addition to including the temporal behavior of industrial control application, scalability of the proposed approach, i.e., the evaluation in larger industry projects, will be part of future work.

ACKNOWLEDGMENT

The authors want to thank Michael Strasser for his support at the implementation part. The research leading to these results has received funding by the Klima+Energie Fonds under grant agreement no. 829823 (E3ICP) managed by the Österreichische Forschungsförderungsgesellschaft (FFG). Dietmar Winkler's work has been partly supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

REFERENCES

[1] W. Schäfer and H. Wehrheim, "The challenges of building advanced mechatronic systems," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 72–84. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.28

[2] A. Spillner, T. Linz, and H. Schaefer, *Software Testing Foundation: A Study Guide for the Certified Tester Exam*, 3rd ed. Rocky Nook, 2011.

[3] K. Beck, *Test Driven Development by Example*. Addison-Wesley, 2002.

[4] A. Zylberman and A. Shotten, "Test language - introduction to keyword-driven testing," *Quality Assurance and Software Testing*, p. 9, 2010.

[5] IEC 61131-3, *IEC 61131-3 Standard - Programmable controllers - Part 3: Programming languages*, 2nd ed. International Electrical Commission, 2003.

[6] IEC 61499-1, *Function blocks – Part 1: Architecture*. Geneva: International Electrical Commission, 2005.

[7] C. J. Nagle, *Software Automation Framwork Support SAFSDEV*, http://safsdev.sourceforge.net/, SAS Institute Inc. Std., 11 2011. [Online]. Available: http://safsdev.sourceforge.net/

[8] K. Thramboulidis, "Model-integrated mechatronics - toward a new paradigm in the development of manufacturing systems," *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 54 – 61, 2 2005.

[9] P. Baker, Z. Dai, and G. J., *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2007.

[10] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Pearson Studium, 2008.

[11] A. Mattsson, B. Lundell, B. Lings, and B. Fitzgerald, "Linking model-driven development and software architecture: A case study," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 83–93, Jan/Feb 2009.

[12] D. Winkler, S. Biffl, and T. Östreicher, "Test-Driven Automation - Adopting Test-First Development to Improve Automation Systems Engineering Processes," in *Proceedings of the 16th EuroSPI Conference*, 2009.

[13] R. Hametner, D. Winkler, T. Östreicher, S. Biffl, and A. Zoitl, "The Adaptation of Test-Driven Software Processes to Industrial Automation Engineering," in *8th IEEE International Conference on Industrial Informatics*, 2010.

[14] M. Karlesky and G. Willams, "Mocking the embedded world: Test - driven development, continuous integration, and design patterns," in *Proceeding of the Embedded Systems Conference*, 2007.

[15] M. Utting and B. Legeard, *Practical Model-based Testing*. Morgan Kaufmann, 2007.

[16] M. Fewster and D. Graham, *Software Test Automation*. Addison-Wesley Professional, 1999.

[17] X. Wang and P. Xu, "Build and auto testing framework based on selenium and fitnesse," in *Proceedings of the Int. Conf on Information Technology and Computer Science*, 2009, pp. 436–439.

[18] Rashmi and N. Bajpai, "A keyword driven framework for testing web applications," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 3, no. 3, pp. pp. 8–14, 2012.

[19] D. Winkler, R. Hametner, T. Östreicher, and S. Biffl, "A Framework for Automated Testing of Automation Systems," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2010.

[20] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-Based Testing of Reactive Systems*. Springer, 2005.

[21] R. Hametner, A. Zoitl, and M. Semo, "Component Architecture for the Efficient Development of Industrial Automation System," in *6th IEEE Conference on Automation Science and Engineering Proceedings*, August 2010.

[22] A. Zoitl, T. Strasser, and A. Valentini, "Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study," in *IEEE International Symposium on Industrial Electronics (ISIE)*, 7 2010, pp. 3817 –3819.

[23] K. Thramboulidis, "Using uml in control and automation: a model driven approach," in *INDIN*, 2004.

[24] R. Hametner, D. Winkler, T. Östreicher, N. Surnic, and S. Biffl, "Selecting UML Models for Test-Driven Development along the Automation Systems Engineering Process," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2010.

[25] S. Mouchawrab, L. Briand, Y. Labiche, and M. Di Penta, "Assessing, comparing, and combining state machine-based testing and structural testing: A series of experiments," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 161–187, Mar/Apr 2011.

[26] C. Seidner and O. Roux, "Formal Methods for Systems Engineering Behavior Models," *Industrial Informatics, IEEE Transactions on*, vol. 4, no. 4, pp. 280 –291, 11 2008.

[27] S. K. Swain, D. P. Mohapatra, and R. Mallc, "Test Case Generation Based on State and Activity Models," *Journal of Object Technology*, vol. 9, no. 5, pp. 1–27, 2010.

[28] J. Kim, I. Kang, J.-Y. Choi, and I. Lee, "Timed and Resource-Oriented Statecharts for Embedded Software," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 568 –578, 11 2010.