Test-Driven Automation: Adopting Test-First Development to Improve Automation Systems Engineering Processes

Dietmar Winkler Stefan Biffl Thomas Östreicher

Vienna University of Technology,
Favoritenstrasse 9/188, 1040 Vienna, Austria
{dietmar.winkler, stefan.biffl, thomas.oestreicher}@qse.ifs.tuwien.ac.at

Abstract

Software provides an increasing part of the added value of modern automation systems and thus becomes more complex. System requirements may change even late in the development process, lead to ad-hoc modifications of the product and require systematic (and automated) testing approaches. However, unit tests for automation software have to consider the interaction with hardware components, are often not systematically automated, and thus make defects during integration testing harder to find. Costly software integration makes the introduction of more flexible software processes that support the late change of requirements more risky. In this paper we introduce the concept of "Test-Driven Automation" (TDA), which adopts the successful idea of test-first development from business software development to the automation systems domain: develop test cases before the implementation and systematically automate unit tests to ensure sufficient testing on unit level to lower the cost and risk of systems integration. As foundation for TDA we present the characteristics of the design of a TDA software component, i.e., interfaces to (a) automation functions, (b) diagnosis functions to allow test observation, and (c) test functions for setting the component to defined states, e.g., to test behavior in error situations. We demonstrate in an industrial sorting application prototype how the TDA approach can make testing more efficient and provide diagnosis information for process analysis and improvement.

Keywords

Test-driven automation, automation systems, process support, testing, diagnosis.

1 Introduction

Software-intensive automation systems, like industrial manufacturing plants, need to become more flexible and robust to respond to changing business processes and business requirements. Functionality is increasingly realized in software components, which leads to an increased complexity of software components embedded within the hardware solution [17]. Engineers in the automation systems domain often have a non-software-engineering background and only limited knowledge on developing complex software-intensive systems. Late changing requirements and costly software and systems integration makes the introduction of more flexible software processes more risky [5][7]. Thus, effective and efficient software development methods and processes are necessary to support systems development and quality assurance across disciplines [8], i.e., software and automation systems.

Diagnosis and test are challenges in the automation systems domain to monitor and control current systems behavior during operation and maintenance [16] and respond to the current system status even before a failure occurs. In typical automation systems solutions we can observe software solutions that focus on functional requirements [18] and contain limited and often unsystematic diagnosis and test capabilities. Test and diagnosis aspects are scattered unsystematically in the code and hinders efficient re-validation and diagnosis. This ad-hoc approach to testing and diagnosis makes enhancements, refactoring and maintenance tasks more risky and expensive to validate [16]. Nevertheless, frequent and automated tests e.g., unit tests, can support engineers in finding defects early [2][9]. In contrast to business software development, unit tests in the automation domain have to consider interaction with the hardware [17]. These unit tests are often not systematically automated, and thus hinder efficient defect detection. A systematic separation of software aspects into functional behavior, testing and diagnosis aspects with well-defined communication and data exchange can increase product quality and support integration testing and maintenance for software-intensive automation systems. These characteristic aspects represent the foundation for "Test-Driven Automation" (TDA): interfaces to automation functions, diagnosis functions to allow test observation, and test functions for setting the component to defined states, e.g., to test systems behavior in error situations.

The concept of TDA adopts the successful idea of test-first development from business software development [2][9], e.g., administrative systems with databases, to help automation systems developers improve their development process. Test-first-development (TFD), an established method in business software development [2], focuses on quality assurance as an integral part of the development process and refers to the concept of early testing. This concept represents the foundation for automated unit-testing to lower the cost and risk of systems integration for iterative systems development [11]. Nevertheless, software processes are necessary to guide the engineers during the project. Software processes define sequences of steps along the product lifecycle and focus on specific needs of the project application context, like application domain and project attributes (e.g., size, complexity, and stability of requirements). In modern business software development a wide range of traditional (sequential) software processes, e.g., W-Model [1] and V-Modell, and flexible software processes, e.g., eXtreme Programming [2] and Scrum [3], support software engineers in developing high-quality products. The V-Modell XT[1], published in 2005, enables flexibility due to a modular process unit structure and provides a range of project execution strategies including agile approaches [6]. In the automation systems development industry the choice of systematic processes, e.g., GAMP [13], seems in practice limited to sequential processes, like waterfall models. To benefit from more flexible iterative approaches, automation systems engineers can learn from process approaches in business software development to handle the increasing complexity of software components in systems development.

The remainder of this paper is structured as follows. Section 2 introduces the concepts of TFD, Section 3 discusses requirements and challenges of the automation systems domain and identifies research issues for TDA. In Section 4 we introduce the TDA concept in context of a common engineering process. To illustrate how the novel TDA approach can increase testing efficiency and can provide diagnosis information for process analysis and improvement, we discuss an industrial sorting application prototype in Section 5. Finally, Section 6 summarizes lessons learned from the TDA prototype application and points out further research work.

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

2 Test-First Development in Business Software Development

Traditional (sequential) software development approaches, e.g., waterfall models, place test case definition and test execution late in the development process after code construction. The identification of defects (caused by incomplete, wrong or ambiguous requirements) in late development phases can lead to (a) high effort to locate defects in the large application context, (b) high risk of defects that do not occur often, and (c) high rework effort to fix the problems [5].

Test-first development (TFD) [2] is a strategy to address these issues by shortening the cycles between test case definition and test execution [9]. Test cases are defined prior (or at least in parallel) to the implementation of a component. In general, the concept of TFD consists of 4 steps [2]: (1) Selection of a specific requirement and implementation of test cases to check the requirement for correctness (*Think*); (2) Test case execution. As there is yet no implementation of functionality, this test case must fail (*Red Test Result*). (3) Ongoing implementation of test-case related functionality and test-case execution until the test case is successful (*Green Test Result*). (4) Optimization of the implementation design without changing functionality and execution of test cases (*Refactor*). After finishing step 4 selection of the next batch of requirements. This approach can lead to a comprehensive understanding of the basic requirements and early defect recognition in case of unclear and incorrect requirements during test case generation. Additionally, TFD enables immediate feedback during component implementation and test because of frequent test runs in short iterations and thus represents the foundation for a continuous integration strategy [11].

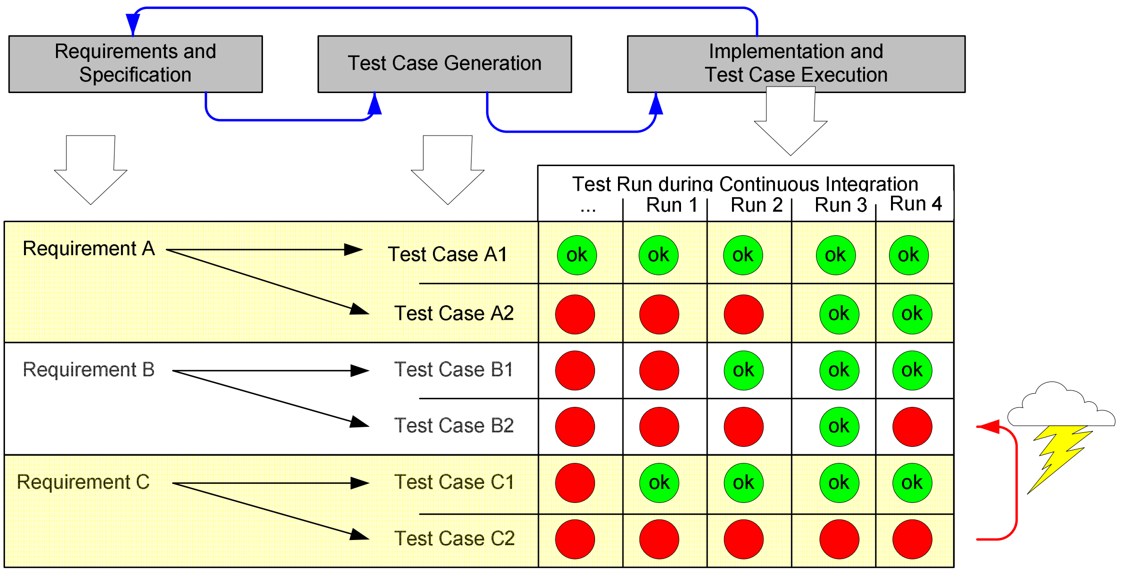


Figure 1: Concept of Test-First Development with several Test Runs.

Figure 1 illustrates the application of TFD in a typical project context in business software development. Test cases on business level (e.g., system and acceptance testing) can be derived from requirements (e.g., Requirement A maps to test cases A1 and A2). Frequent test runs during implementation provide immediate feedback on the current implementation task. Thus, implemented requirements and test case execution results lead to (a) successful test cases (marked green, requirement was already implemented correctly) and (b) unsuccessful test cases (marked red, implementation not finished or corrupted). TFD and continuous integration also helps identifying possible negative side-effects on other system parts (i.e., regression testing) which are not in the focus of the current implementation task (status switched from green to red without working actively on the affected code). For instance, the implementation of functions tested with test case C2 has a negative impact on the test case B2 result.

Frequent test runs in short iterations enable the observation of real project progress and deliver immediate feedback on the overall project status. Thus, automated testing is a pre-condition for continuous integration [11] including early and effective testing approaches.

3 Challenges and Research Issues in Automation Systems

In the automation systems domain we can observe a wide range of application types, e.g., production automation systems with focus on logistics and routing [19], embedded systems with limited resources, and real-time systems with time critical requirements [15]. In this paper we focus on applications of industry automation systems, e.g., assembly workshops to combine smaller parts into more complex products, to identify challenges for developing and testing software for automation systems [4]. Figure 2 illustrates the typical structure of an industrial automation system [12] on three layers: (a) business processes with dispatchers who turn customer contracts into work orders, (b) workshop operators for configuration and coordination of transport system and machines, and (c) layer of individual machines in the workshop with control systems from systems engineers and machine vendors. Plans and status reports link the layers: the current machine status can be used to reconfigure the current workshop (on workshop layer) and propagate status information to the business layer to reschedule work orders, if incidents on lower levels limit the effective workshop capacity [12].

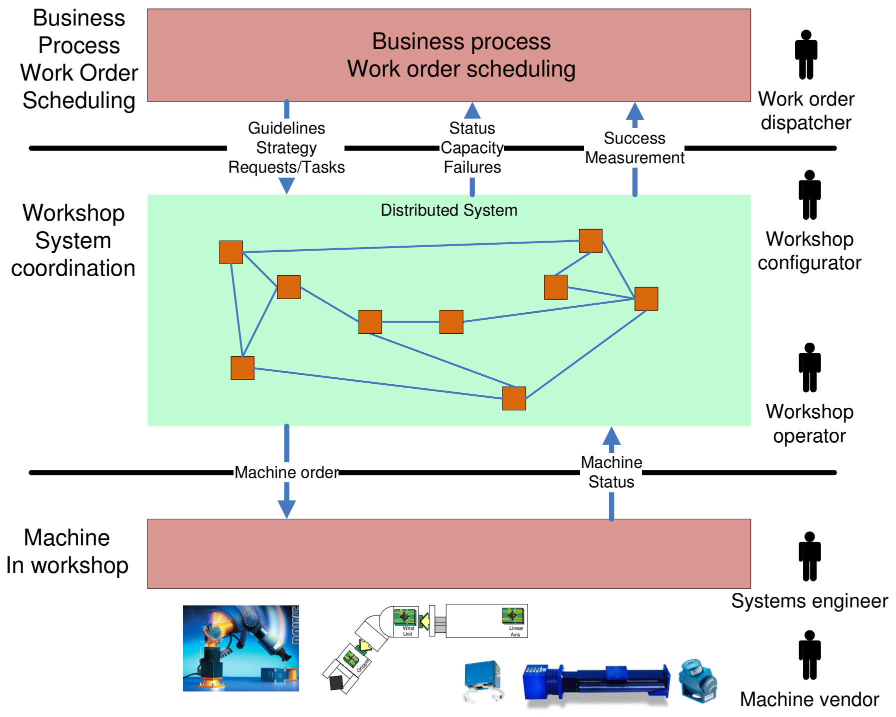


Figure 2: Levels of Automation Systems according to [12].

A typical systems engineering component in automation industry has to address (a) functional aspects to fulfill the required functional requirements [18], (b) diagnosis aspects to monitor and control systems behavior [16] and to predict upcoming maintenance needs, and (c) testing aspects to check systems behavior during systems development including tests cases on error conditions [16], e.g., regression testing after changes, and system tests during operation after completed maintenance tasks [17]. In traditional automation systems engineering the diagnosis and sometimes testing aspects are merged into the functional automation solution which hinders adapting efficient diagnosis and verification to changes during operation and maintenance.

Based on the Artemis[2] roadmap for automation systems research and discussions with industry partners in the Medeia[3] and logi.DIAG[4] research projects we derived the following needs and research issues:

[2] https://www.artemisia-association.org

[3] *Medeia: Model-Driven Embedded Systems Design Environment for the Industrial automation Sector*, http://www.medeia.eu/.

[4] *logi.DIAG: Test-Driven Automation in Systems Environments*, http://www.logidiag.at.

- *Separation of design aspects for automation functionality, testing, and diagnosis aspects.* Automation systems components typically include automation functionality, test-case functionality, and diagnosis functionality without a clear separation of these aspects, which makes the components unnecessarily hard to modify and validate [16][17]. Thus, we see the separation of these aspects as pre-condition for efficient systems development, operation, and maintenance to enable the selective evaluation of required attributes, e.g., test-coverage analysis during development. Test cases need to set the system in a certain state, e.g., stimulate an error state, and test the response of the system by using collected data from diagnosis. Thus, data derived from diagnosis functions can be used (a) for immediate feedback on the systems state (during development, operation, and maintenance), (b) for prediction of required maintenance tasks during operation, and (c) data can be used independently and combined as needed to get information on business level. *Thus, the first research issue is (a) how a TDA component can be designed and (b) how the individual aspects (automation, diagnosis, and test) can interact with each other efficiently.*
- *Efficient validation based on the test-first approach.* Efficient validation and re-validation after changes are key issues to reduce avoidable system downtime. Our observation in the automation systems domain showed a focus on functional requirements with limited and often unsystematic testing capabilities [18]. In the hierarchical design of automation systems we assume that TFD can be applied to components on all levels and in all process steps, e.g., on requirements level, architecture level, and component and module level. Process models, e.g. the V-Modell XT and the W-Model [1], can provide a framework for introducing systematic TDA. *The second research issue is (a) how the test-first approach can be realized in the production automation domain and (b) how the TDA concept can enable early testing on various levels during systems development.*

4 The Concept of Test-Driven Automation (TDA)

This section presents the concept of the TDA component structure and the interaction with its environment and introduces the concept of test-first development based on a suggested process approach for the automation systems domain.

4.1 Component Aspects in Test-Driven Automation

Bundling functional, diagnosis, and testing aspects in test-driven automation (TDA) components provide a strict separation of individual automation aspects. Interfaces enable an efficient communication in a hierarchical systems design. In common hierarchical automation systems, functional components and test cases are spread and mixed over the design of the systems. Diagnosis functions are typically add-ons without systematic integration within the systems design [16]. A strict separation of these components including defined interaction mechanisms are pre-conditions for efficient systems development, operation, and maintenance. Therefore, a test-driven automation component (TDA component) consists of these three aspects, (a) *automation aspect*, (b) *diagnosis aspect*, and (c) *testing aspect*. Figure 3 illustrates these aspects and Figure 5 provides the example of the prototype study.

The *automation aspect* implements the functional and logical behavior of the component and interacts via interfaces with other system components within a hierarchical systems design. The *diagnosis aspect* is separated from functionality but interacts with the automation aspect to get access to hardware signals. Instructions from automation functions are passed to sub-components via the diagnosis interface; measurements and results are received from sub-components and are (a) interpreted (diagnosis functionality) and (b) passed to the automation interface to respond to system overall results and signals. Additionally, diagnosis includes an interface to report measurement results to higher TDA components (e.g., for aggregation and reporting purposes). *Testing aspects* provide test functions for the TDA component, e.g., mocked results for the setup of test scenarios [14]. An important task of the test aspect is to set up error conditions (e.g., malfunctions of machines) for the unit test of diagnosis and logical functions without disturbing machine hardware components. Thus, these test functions can be applied for unit testing during development and during operation in case of sub-component exchange.

4.2 Test-First Development Approach with TDA Components

Based on the definition of the TDA component, including the three automation aspects, i.e., automation, diagnosis, and testing aspects, test-first development (TFD) can be applied in context of a software engineering process model. Our observations in automation systems industry reveals that typical engineering processes are similar to the waterfall approach. Thus, the V-Modell XT seems to be a promising basic model for systems automation development because of its flexibility and adaptability on various application domains [6]. Figure 3 presents the technical part of our suggested iterative model based on the V-Modell XT with respect to automation systems characteristics. Test-first development (TFD) is applicable on three levels (requirements, architecture & integration, and component) and enables early test case generation on every level.

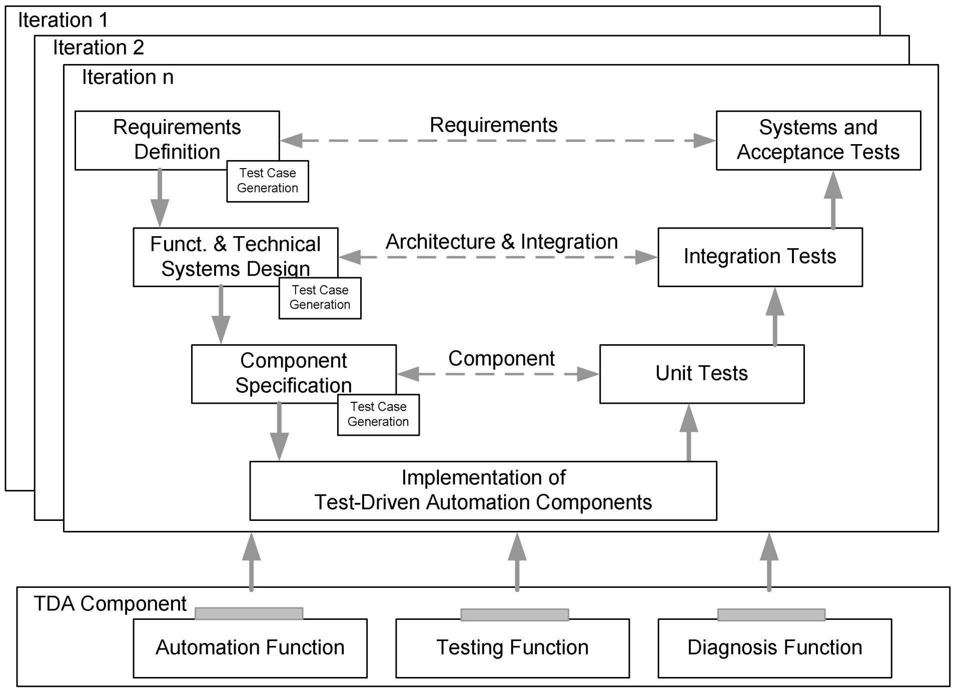


Figure 3: Concept of a TDA process based on the V-Model XT.

TFD includes the test case definition in the specification phases, i.e., requirements definition, functional & technical systems design, and component specification, of the process model and the execution of the test cases during and after implementation. Table 1 gives an overview on test levels of individual phases, deliverables generated during the entire phase, and relevant stakeholders.

Table 1: Test Levels and Deliverables for TDA according to the V-Model XT.

Phase	Deliverables	Test Level	Stakeholders
Requirements Definition	Use Cases	System / Acceptance Testing	Customer, Factory Setting
Functional and Technical Systems Design	Component diagrams, State-Charts	Architecture / Integration Testing	Engineering Team
Component Specification	State-Charts	Component Testing	Individual Engineer
Implementation of TDA Components	Function Blocks	Developer Testing	Individual Engineer

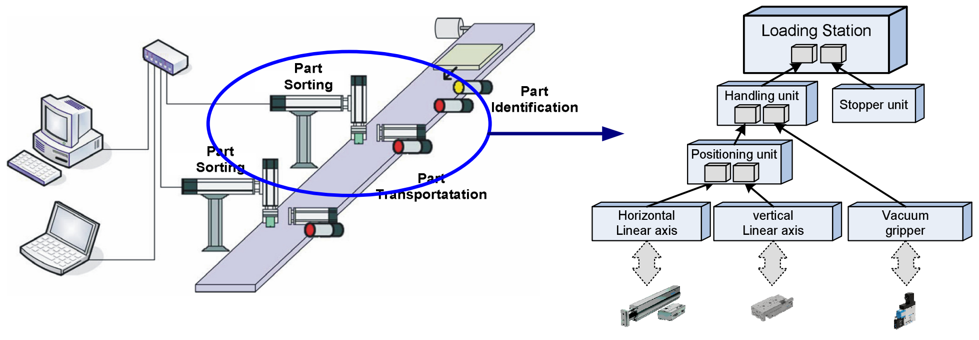
Test cases on requirements level address customer requirements and factory settings according to business goals and risks from business perspective. Note that test case definition on this layer provides a test framework for later development phases. Thus, underlying functionality must be mocked and simulated to enable successful test case execution [14]. This functionality can be provided by the testing aspect (i.e., setting the system in a certain state) and the diagnosis aspect (measuring results of the system) of the TDA component. On architecture and integration level test cases (and mocked individual components) address the engineering team and focus on components, interfaces and the interaction between components. Component diagrams and state charts are common models to describe components and interaction of these components. Component testing [9], based on a detailed state charts, addresses individual engineers and focuses on detailed functionality of the automation system. Function blocks and structured text are used for implementation purposes on the lowest and most detailed level of the process approach. This part must be considered during constructing individual components. In the context of this paper this phase is out of scope. Zhang *et al*. provide an approach for the specification and verification of applications based on function blocks. [20].

5 Prototype Study: Sorting Application

In this section we demonstrate the TDA approach with components in an industrial sorting application prototype, i.e., a typical machine in an assembly workshop as described in Section 3. The goal is to show how the TDA approach can make testing more efficient and provide diagnosis information for process analysis and improvement.

5.1 Sorting Application Description

The sorting application represents a typical setting in an assembly workshop within a production automation system. The main task of the sorting application is to recognize the type of an incoming clamp according to defined sorting criteria (e.g., derived from business goals) and sort it into the appropriate output box. The sorting application should manage to sort 6 clamps per minute. Figure 4a provides a schematic overview of the system.



(a) Workshop Layout

(b) Component Model of the Sorting Application.

Figure 4: Sorting Application: (a) Workshop Layout and (b) Component Model based on Sünder *et al*. [18].

Figure 4b shows the component model of sorting application, a strictly hierarchical system as typical in industry automation system applications. To illustrate the TDA component and the interaction with its environment and to show the process of constructing models with respect to Model-Driven Testing [1] and generating test cases based on the test-first approach, we focus on a subset of components, i.e., the Handling Unit. The main task of the Handling Unit is to control two axes, i.e., the horizontal and vertical axis and the vacuum gripper to pick up the part for sorting purposes.

5.2 TDA Component of the Handling Unit

The TDA component encapsulates functional behavior, testing, and diagnosis functionality including interaction within the TDA component and providing/requesting interaction activities to other parts of the system using defined interfaces for automation, diagnosis, and testing. The component model of the Sorting Application (see Figure 4b) illustrates the interrelationship of the Handling Unit with other parts of the system: (a) *Loading Station* and (b) *Positioning Unit* and *Vacuum Gripper*. The Loading Station monitors and controls the Handling Unit via the functional interface, requests diagnosis data and initiate test cases (e.g., stimulates a defined state or a possible unreachable system state) via appropriate interfaces for diagnosis and testing. Note that the Loading Station represents another (higher-level) TDA component in the hierarchical systems structure. The Positioning Unit and the Vacuum Gripper (also – lower-level – TDA components) are controlled by the Handling Unit via functional interfaces (e.g., move to a certain position or grab a work product) and receives information whether the planned position is reached or the work product has been picked up via the appropriate diagnosis interfaces. Testing interfaces enable the Handling Unit to set the Positioning Unit and the Vacuum gripper in defined states, e.g., an error state.

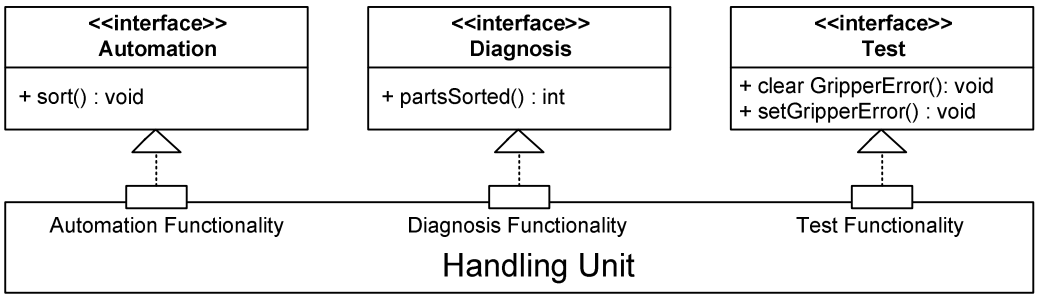


Figure 5: TDA Component Interface Model for a Handling Unit.

Figure 5 illustrates a simple component interface model of the Handling Unit. The component is triggered via the automation (functional) interface by the Loading Station `sort()` and returns the current state via the diagnosis interface `partsSorted()` after task completion. Error states are passed to the Vacuum gripper `setGripperError()` and `clearGripperError()` via testing interfaces to stimulate an error to test appropriate systems behavior of a supposed error of the gripper during development or maintenance. The strict hierarchical systems design of production automation systems enables TDA components to provide functional, diagnosis, and testing aspects to higher levels in the hierarchy. Thus, the TDA concept is, in principle, applicable on all levels of an automation system.

5.3 Test-First Development with Test-Driven Automation

The strict separation of functional, diagnosis, and testing functions in a hierarchical automation systems design based on TDA components is the foundation for successfully applying the test-first approach of the TDA concept. Following the suggested process steps (see Figure 3), test-first is applicable on three levels: (a) requirements definition, (b) functional & technical systems design, and (c) component specification.

Table 2: Test Cases derived from Requirements and Use-Cases.

No.	Desc.	Level	Type*	Pre-condition	Input	Expected Result	Post-condition
1	Sorting a Part	System	NC	Handling Unit in idle Position	Command to sort part	Handling Unit in idle Position and part sorted	Handling Unit in idle position
2	Through-put	System	NC	Handling Unit in idle Position	Command to sort part	Part has been sorted in less or equal than 10sec.	Handling Unit in idle position

Requirements definition. Requirements are success-critical issues in automation systems engineering as customer changes and modifications typically have a major impact on system integration and can cause a high amount of rework effort and cost [5][7]. UML use cases are appropriate approaches in business software development [1] to model requirements from user perspective and are applicable to the automation systems domain. The Handling Unit consists of a few use cases: (a) *sort part* (initiated by the Loading Station), (b) *Pick up part* and *Release part* (controlled by the Handling Unit), and (c) *Gripper Positioning* to get the arm moved to a certain position. Based on the basic requirements and the test-first approach of the TDA concept test cases can be derived directly from the use cases.

Table 2 presents selected regular/normal test cases (NC) on systems level. Note that test-cases should include normal test cases (NC), special test cases (SC) representing systems behavior in the border area of regular systems behavior, and error cases (EC) for error states of the system.

Functional and technical systems design and *component specification.* Based on use cases and the architecture of the hierarchical automation system (see Figure 4b) test cases can be derived on integration test level to test the interfaces and the interaction of the related TDA components prior to the detailed specification and the implementation (i.e., test-first approach). To enable automated testing based on test-first development, state charts can enable modeling the desired behavior of the system. In automation domain state charts are well-established and enable automated verification and validation [10]. Figure 7 illustrates the state chart of the Handling Unit including error states, which can be initiated via the testing interface and monitored via the diagnosis interface of the TDA component.

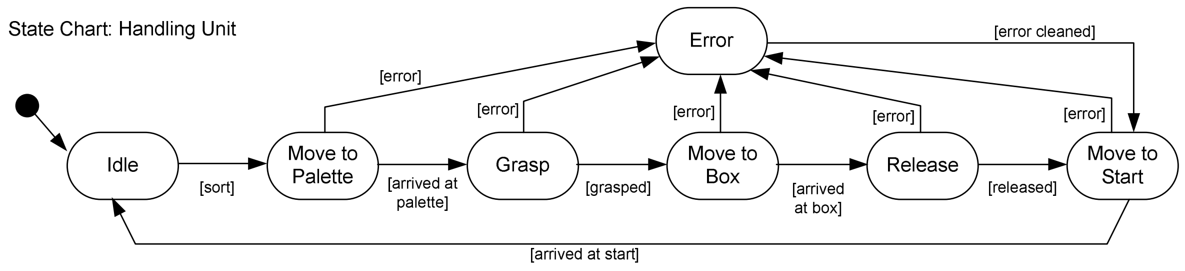


Figure 6: State Chart of the Handling Unit.

Table 3 illustrates selected test cases derived from the state chart, i.e., a regular test case and an error case, to demonstrate test case definition in TDA. Note that the gripper is in *idle* state (pre-condition) and should move to an *entire position*: Test case 1 describes a *regular test case* and test case 2 addresses an *error situation*, e.g., if one axis got stuck.

Table 3: Test Cases of the Handling Unit based on State Charts.

No.	Desc.	Level	Type*	Pre-condition	Input	Expected Result	Post-condition
1	Gripper move to Pos	Comp.	NC	Handling Unit idle	Sort part	Gripper moved to intended position	Gripper is in intended pos.
2	Axis got stuck	Comp	EC	Handling Unit in idle Position	Sort part; error after 3s	Positioning Unit reports an error; Handling Unit idle	Handling Unit in idle position

Testing automation systems also includes testing error states within a continuous integration strategy. In industry practice the definition of hardware error states (e.g., stuck of an axis) is an increasing challenge because (a) error states often must be initiated manually and are not feasible because of the availability of error states of machines and (b) required software components are not available during test time. Thus, these error states have to be simulated using mocking approaches to enable efficient testing and re-validation [14].

5.4 Mocking with State Charts

The application of TFD on various levels can require the simulation of the underlying functionality, if this functionality is not yet fully implemented. Thus, mocking of components is required to successfully apply TFD. A mocking component is a simple simulation of a subcomponent or external system which is required by a component to perform its behavior [14]. It should be as simple as possible, implementing only the behavior that is needed by the component under test. This facilitates test automation by enabling the testing of a component while its subcomponents are not yet fully implemented. Mocking also enables testing without deploying to the target hardware and the possibility of generating error states although an actual error is not present. Figure 7 shows a sample mockup in state chart notation of the Gripper Unit to simulate its basic behavior including error generation. Similar to test case generation mocking components are efficient modeling approaches for verification and validation purposes [14].

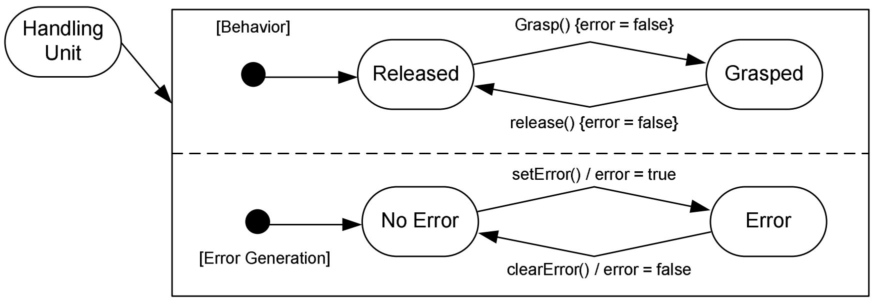


Figure 7: Mocking with Error Generation of the Gripper Unit.

The strict separation of individual software aspects and the interaction of individual TDA components enable a transparent systems design including testing and diagnosis capabilities. Based on the TDA concept TFD enables more efficient and systematic tests in comparison to the traditional automation systems approaches.

6 Summary and Further Work

The increasing need for flexibility of automation systems and the trend to shift functionality from hardware to software solutions leads to new challenges in the software development and to increased complexity of software. Up to now, code & fix approaches have been common practices in industry automation systems development. The increasing complexity requires more systematic software development approaches (methods and processes) for construction, refactoring, and verification and validation. In this paper we introduced the concept of “Test-Driven Automation” (TDA), which adopts the successful idea of test-first development from business software development to the automation systems domain and presented a novel TDA component including a strict separation of functional, testing, and diagnosis aspects to support more efficient testing on various levels of development.

Applying the presented concepts on a pilot sorting application we derived a set of lessons learned: *Packaging automation, testing and diagnosis aspects* in Test-Driven Automation (TDA) components provide strictly separated functions including well-defined communication over interfaces. Test functionality can put automation components into states for testing purposes that the automation logic would not reach with regular input, e.g., testing correct reaction on failure states. Diagnosis functionality enables measurement on the current system status and provides information for independently analysis of test and machine status and can support engineers during development, operation and maintenance, e.g., monitoring of systems behavior during operation and external data analysis for improvement purposes. *Test-first development* based on models can help to foster early test-case generation and increase the understanding of systems behavior and lead to higher product quality. TFD enables automated and frequent test case execution, support continuous integration and provides a framework for fast and efficient re-validation after changes in hardware and software components. Developing TDA components with the test-first approach lead to *an iterative development process* and provide well-defined and flexible framework for project execution.

Future work is (a) to refine the process approach with emphasis on automation systems development based on the V-Modell XT including domain-specific process tailoring and (b) to investigate the scalability of the TDA concept by addressing a larger pilot application with industry partners, with particular emphasis on data collection to compare the effectiveness of traditional testing in automation systems engineering and testing following the TDA concept.

Acknowledgements

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

Literature

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

[2] Beck K., Andres C.: Extreme Programming Explained - Embrace Change, Addison-Wesley, 2004.

[3] Beedle M., Schwaber K.: Agile Software Development with Scrum, Prentice Hall, 2008.

[4] Biffl S., Schatten A., Zoitl A.: Integration of Heterogeneous Engineering Environments for the Automation Systems Lifecycle, accepted for publication in Proc. IEEE Industrial Informatics Conf., 2009.

[5] Biffl S., Aurum A., Boehm B., Erdogmus H., Grünbacher P. (eds.): Value-Based Software Engineering, Springer Verlag, 2005.

[6] Biffl S., Winkler D., Höhn R., Wetzel H.: Software Process Improvement in Europe: Potential of the new V-Modell XT and Research Issues, in Journal Software Process: Improvement and Practice, Volume 11(3), pp.229-238, Wiley, 2006.

[7] Boehm B.: Software Engineering Economics, Prentice Hall, 1981.

[8] Chan K.K., Spedding T.A.: An integrated multi-dimensional process improvement methodology for manufacturing systems, Comput. Ind. Eng. 44(4): 673-693, 2003.

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

[10] Drusinksy D.: Modeling and Verification using UML Statecharts, Newnes, 2006.

[11] Duvall M.P., Matyas S., Glover A.: Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, 2007.

[12] 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.

[13] GAMP 4: Good Automation and Manufacturing Practice – Guide for Validation of Automated Systems, 2001.

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

[15] Larsen K. G., Mikucionis M., Nielsen B., Skou A.: Testing real-time embedded software using UPPAAL-TRON: An industrial Case Study. In Proc. 5th ACM Int. Conf. on Embedded Softw.,. ACM Press, 2005.

[16] Nandi S., Toliyat H.A.: Condition Monitoring and Fault Diagnosis of Electrical Machines, Conf on Industry Applications, Phoenix, US, 1999.

[17] Schäfer W., Wehrheim, H.: The Challenges of Building Advanced Mechatronic Systems. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, pp. 72-84, 2007.

[18] Sünder C., Zoitl A, Dutzler C.: Functional Structure-Based modelling of Automation Systems, Jounal of Manufacturing Research, 1(4), pp405-420, 2007.

[19] Vyatkin V., Christensen J.H., Lastra J.L.M.: OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation, IEEE Trans. on Industrial Informatics, vol. 1, pp. 4-17, 2005.

[20] Zhang W., Diedrich C., Halang W.A.: Specification and Verification of Applications Based on Function Blocks, Computer-Based Software Development for Embedded Systems (LNCS 3778), Springer Verlag Berlin Heidelberg, pp. 8-34, 2005.

Author CVs

Dietmar Winkler

Dietmar Winkler is researcher and lecturer at the Institute for Software Technology and Interactive Systems, Vienna University of Technology, Austria. He received an MS in Computer Science from Vienna University of Technology. In 2007 he worked as a guest researcher at Czech Technical University, Department of Cybernetics in Prague (CZ). In 2008 he received a scholarship by Siemens Austria and the Faculty of Informatics of TU Vienna for a 4 months research stay at the Fraunhofer Institute of Experimental Software Engineering in Kaiserslautern (DE). His research interests include software engineering and software processes, quality management and quality assurance, and empirical software engineering. Additionally, he is software engineering and quality management consultant in the automotive business domain and in public sector IT projects and project member of the research project “Test-Driven Automation with logi.DIAG”.

Stefan Biffl

Stefan Biffl is an associate professor at the Institute for Software Technology and Interactive Systems, Vienna University of Technology, Austria. He received MS degrees in Computer Science and Business Informatics from Vienna University of Technology (VUT) and University of Vienna, respectively, and a PhD degree from VUT. In recent years he worked as guest researcher at the Fraunhofer Institute for experimental Software Engineering (Kaiserslautern, Germany) and at Czech Technical University in Prague, Department of Cybernetics. His research interests are in the areas: Value-Based Software Engineering; Empirical Software Engineering, risk and quality management, and software process improvement. He is a member of the IFIP Technical Committee on Software Engineering (IFIP TC2) and principal researcher of the research project “Test-Driven Automation with logi.DIAG”.

Thomas Östreicher

Thomas Östreicher is researcher and project member of the research project “Test-Driven Automation with logi.DIAG”. His research interests are in the areas: modeling of software intensive systems, programming languages, and software process management.