Selecting UML Models for Test-Driven Development along the Automation Systems Engineering Process

Reinhard Hametner1  Dietmar Winkler2  Thomas Östreicher2  Natascha Surnic2  Stefan Biffl2

1Automation and Control Institute, Vienna University of Technology
Gußhausstr. 27-29/E376, AT-1040 Vienna, Austria
hametner@acin.tuwien.ac.at

2Institute of Software Technology, Vienna University of Technology
Favoritenstr. 9-11/188, AT 1040 Vienna, Austria
{dietmar.winkler, thomas.oestreicher, natascha.surnic, stefan.biffl}@qse.ifs.tuwien.ac.at

Abstract

Test-driven development (TDD) – an established approach in business IT software development – enables test case generation based on models early in the development process. Applying TDD and models in automation systems engineering (ASE) can increase testing effectiveness and efficiency. A key question is which models are suitable for ASE application. UML models support software and systems engineering development in (a) systematically capturing requirements, (b) describing the static system architecture, and (c) specifying dynamic systems behavior. In this paper we discuss selection criteria for UML model selection in ASE and evaluate strengths and limitations of selected models.

Keywords: Automation Systems Development, UML, Test-Driven Development, Test Case Generation.

1. Introduction

Test-driven development (TDD) is a common approach in business IT software development [6][8]. Test cases are defined prior or at least in parallel to product design and implementation. Thus, TDD can (a) facilitate test case generation, (b) improve interaction between various stakeholders from different disciplines (e.g., communication based on test cases and scenarios), and (c) decrease the number of defects during design and development. An important issue is the identification of most valuable test cases to focus on key components of the system under development.

Models can support engineers during design, construction, and verification and validation of systems by systematically capturing static system structures (e.g., components and interaction between components) and systems dynamics (e.g., workflows and behavior) [5]. In addition, models can support (semi-automated) code and test case generation [5]. Because of a graphical representation of static systems structures and systems dynamics, models aim at supporting engineers from various disciplines in conducting quality assurance (QA) activities [4]. For instance, automation systems engineering (ASE) includes mechanical, electric, and software components. Engineers from different disciplines have to interact and collaborate during the development process [4]. However, various disciplines apply different sets of models (within individual scopes of disciplines) that may hard to understand for engineers coming from another discipline.

Unified models, e.g., based on UML [1] and SysML [7] promise to support communication and collaboration between various stakeholders in the ASE process [1]. Common models can support QA, e.g., reviews and testing, to increase systems quality. In addition, models are the foundation for TDD [8][17], because test cases can be systematically derived from models, in many cases automatically [2][5]. As there is a wide range of models we focus on the UML diagram family because it is a well-known and well-investigated approach in business IT software development and we believe that application of UML in ASE is a promising approach to increase development processes and product quality. Lüder et al. [13] investigated the suitability of a wide range of models for the distributed control systems process. Köhler et al. [11] used UML diagrams as a vehicle for visualizing design and implementation aspects. Nevertheless, it remains open which standard models are sufficient for TDD application in ASE projects. In this paper we discuss criteria for selecting appropriate models for TDD application in the ASE domain, e.g., automation and manufacturing systems.

The remainder of this paper is structured as follows: Section 2 presents an overview on the ASE domain and introduces the application of engineering models for TDD. Section 3 shows criteria for model selection. Section 4 presents the applicability of UML diagrams in the ASE domain. Finally, Section 5 concludes and identifies future work.
2. Related Work

This section presents related work on TDD, ASE and introduces to system development processes and the UML diagram family.

2.1. Test-Driven Development

Test-Driven Development (TDD) [6] is a common agile practice in business IT software development with focus on small and manageable iterations. Main idea is to generate and execute test cases based on defined aspects in the system specification phase prior and/or in parallel to design and implementation [3]. Because of early defined test cases, TDD (a) supports engineers in better understanding the artifact under construction (e.g., based customer requirements) and (b) enables the definition of well-defined test cases to check deliverables for correctness. Note that early test case generation enables the detection of unclear and missing requirements early in the development process. While TDD has been extensively investigated in business IT software development [3], there is little work on the adaptation of TDD in ASE contexts.

Note that early test case execution can require mockup-objects to simulate non-existing systems components [10]. Nevertheless, mockup-objects require additional effort but can be used for (a) early simulation of system parts, (b) automated and frequent test runs, and (c) can enable testing of conditions that are difficult to setup in a testing environment handling real hardware components (e.g., defect of mechanical components).

2.2. Automation Systems Architecture and Models

In the ASE domain we observed a wide range of application types, e.g., production automation systems with focus on logistics and routing [15], embedded systems with limited resources, and real-time systems with time-critical requirements [12]. Increasing complexity of automation systems and the need for flexibility (e.g., based on changing customer requirements and the need to flexible adjust automation systems) tend to implement functional behavior in software components rather than in hardware. New and modified architecture and models are required to address these complexity levels with focus on product development and testing.

Sünd er et al. [14] presented a common architecture approach based on software components, which was extended by Winkler et al. [16] and Hametner et al. [9] into automation component architectures for efficiently developing industrial automation control systems. This approach can be used for designing hierarchical structures based on encapsulated and reusable components. In addition, the increasing complexity of control systems drives the need for systematically applying models across the systems development process. Köhler et al. [11] focus on the design and implementation phase by using UML diagrams as a visual programming language. Lüder et al. report on the strengths and limitations of various models for developing distributed control systems [13] with focus on hardware-related aspects. However, there is very little work on the application of UML models for TDD approaches in the context of ASE.

2.3. UML in Business IT Software Development

The Unified Modeling Language (UML) [1] - embedded within a software engineering process – is widely used in business IT software development and helps (a) identifying and defining various aspects of the systems, (b) enhancing communication and collaboration between various stakeholders, and (c) can enable automated code and testcase generation [5]. Because of the benefits in business IT software development UML diagram are promising approaches for application in ASE [11][13].

UML 2.0 consists of an overall number of 13 diagrams, organized in structure diagrams and behavior diagrams. Behavior diagrams include subset of diagrams, i.e., interaction diagrams, focusing on interaction and collaboration of system components [1]. UML diagrams describe the system under construction from various perspectives with focus on three main diagram types: structure, behavior, and interaction diagrams.

Structure Diagrams describe the static structure of a (distributed) system providing 6 diagrams:

1. **Deployment diagrams** present the run-time configuration of a (distributed) system including hardware and software components.

2. **Component diagrams** help identifying the underlying architecture of systems with focus on interfaces on various levels, i.e., system, subsystem, and individual components.

3. **Class diagrams** describe data structures of systems on a detailed level close to the implementation.

4. **Object diagrams** are used to present instances and the static relationships of instances at run time.

5. **Composite structure diagrams** define the individual architectural components and their interactions.

6. **Package diagrams** enable structuring the models and related artifacts of the UML diagram family.

Behavior diagrams describe the dynamic behavior of the system and include three diagram types:

1. **Use Case Diagrams** describe the functionality of a system from the user perspective including the functional behavior without considering order and sequence of actions.

2. **State Diagrams** describe individual states and transitions of systems and are applicable if system states change depending on system inputs (e.g., emergency stop).

3. **Activity Diagrams** are based on business modeling and provide workflows (sequences of steps).

Interaction diagrams (4 diagram types) are a subset of behavior diagrams with focus on collaboration and communication between systems components.

1. **Sequence diagrams** model the temporal behavior of workflows and related objects (call and response) including timing information.
2. **Timing diagrams** describe the individual system states over time. Timing diagrams are widely used in electronics to specify digital circuit design.

3. **Interaction overview diagrams** model sequences and interactions.

4. **Communication diagrams** are based on messages and illustrate the communication workflow between components.

The construction of models requires additional effort. Thus it is necessary to focus on the most valuable model diagrams and identify a subset of models which are sufficient to support TDD in the ASE context. For ASE projects a key research question is which set of models, in respect to (a) development phases and (b) various test scenarios, is sufficient for ASE application.

### 3. Criteria for Model Selection

Lüder et al. [13] investigated the applicability of various models with respect to distributed control systems requirements close to hardware implementation. In this paper we focus on business case related requirements from the perspective of software development. Lüder et al. [13] identified 6 information sets for model selection (architecture, process of control decisions, signals and states, communication, control-relevant data structures, and interaction and communication processes), which can be summarized in *architecture and structural aspects* [SA] and *temporal behavior* [BE]. Nevertheless, *requirements* [RE] and *risks* [RI] are additional success-critical aspects from business perspective. Thus, based on Lüder et al. [13] and discussions with our project partners, we identified a set of four criteria for model selection and application:

- **[RE] Systematic definition of requirements** to derive test scenarios from user perspective. Test scenarios can be related to acceptance tests, systems tests, and factory tests.
- **[SA] Architecture and structural aspects**, i.e., distribution of components and classes, describing the basic system structure to focus on individual parts of the system (for construction and testing purposes).
- **[BE] Definition of functional and temporal behavior** (i.e., states, activities, and timing aspects). Note that workflows represent sequences of steps of operations and can be used for test case generation, i.e., test scenarios.
- **[RI] Complexity and Risk.** Depending on the complexity and risk of components, specifically focused models become reasonable to capture individual systems characteristics (e.g., by using refined sequence charts) that help understanding and mitigating risks.

### 4. Applicability of UML Diagrams in ASE

This section summarizes the results of the findings regarding the applicability of UML diagrams in ASE with respect to (a) development phases and (b) various test levels. We implemented a simple application (a bottle sorting application, described in [9]) to show the benefits of individual models in TDD and to demonstrate the application within the ASE process approach.

#### 4.1. UML diagrams in ASE development phases

We analyzed UML diagrams based on their definition and the selection criteria with focus on various phases of the ASE process (see [17] for details): (a) requirements definition phase (req), (b) system architecture definition (arch), (c) component architecture definition and detailed design (comp), and Implementation (impl). Table 1 presents the summary and highlights the most promising UML models based on the study, including UML diagram types, selection criteria, and individual development phases.

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Crit.</th>
<th>ASE development phases</th>
<th>Diagram</th>
<th>Crit.</th>
<th>ASE development phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case</td>
<td>[RE]</td>
<td>X</td>
<td>Activity</td>
<td>[RE]</td>
<td>X X X</td>
</tr>
<tr>
<td>Deployment</td>
<td>[SA]</td>
<td>X</td>
<td>Component</td>
<td>[SA]</td>
<td>X</td>
</tr>
<tr>
<td>Timing</td>
<td>[BE]</td>
<td>X X</td>
<td>Class</td>
<td>[SA]</td>
<td>X X</td>
</tr>
<tr>
<td>Composite</td>
<td>[SA]</td>
<td>X</td>
<td>State Machine</td>
<td>[BE]</td>
<td>X X</td>
</tr>
<tr>
<td>Structure</td>
<td>[SA]</td>
<td>X</td>
<td>Sequence</td>
<td>[BE]</td>
<td>X</td>
</tr>
<tr>
<td>Interaction</td>
<td>[BE]</td>
<td>X X</td>
<td>Communication</td>
<td>[BE]</td>
<td>X</td>
</tr>
<tr>
<td>Interaction</td>
<td>[BE]</td>
<td>X X</td>
<td>Object</td>
<td>[SA]</td>
<td>X</td>
</tr>
<tr>
<td>Package</td>
<td>[SA]</td>
<td></td>
<td>Package</td>
<td>[SA]</td>
<td></td>
</tr>
</tbody>
</table>

Note that the highlighted diagram types are used for the prototype implementation. Marked diagrams were identified as useful during the individual phases of systems development. Note that risks [RI] can be addressed by systematically refining models, either focusing on [RE], [BE] and/or [SA]. Package diagrams organize the models and are not connected to the system structure or behavior.

#### 4.2. UML Diagrams on Test Levels

Investigating the application of the models in context of testing, we applied four test levels [17]: (a) systems and acceptance test levels, (b) integration test level, (c) component test level, and (d) developer self-tests during implementation. Table 2 presents an overview of development phases, selection criteria and diagram types, and test levels. Risks [RI] is embedded within all diagram types. Depending on the risk a more detailed view on parts of the system is a candidate for more details models and an in-depth testing.

Experiences gained during our implementation phase identified (a) requirements modeling using use cases and high level activity diagrams and (b) structure dia-
grams e.g., deployment and component diagrams, as most reasonable to capture systems requirements and determine the static structure of the system. Regarding the systems dynamics we found state charts and sequence charts most reasonable. State charts illustrate the relationship between individual system states (including possible parallel states, e.g., emergency stops) which are common scenarios in industry settings. Additionally, sequence charts enable to focus on a specific (most critical) part of the system to identify individual sequences of steps of the system and/or critical and more complex subsystems.

**Table 2. Criteria of UML Diagrams for Automation Systems Application.**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Criterium / Diagram</th>
<th>Test Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Definition (req)</td>
<td>[BE][RE] Use Cases</td>
<td>System / Acceptance Testing</td>
</tr>
<tr>
<td>Functional and Tech. Design (arch)</td>
<td>[BE][RE] Activity Diagram</td>
<td></td>
</tr>
<tr>
<td>Component Specification (impl)</td>
<td>[BE] State Chart</td>
<td>Component</td>
</tr>
<tr>
<td>Implementation (impl)</td>
<td>[BE] Sequence Charts</td>
<td>Testing</td>
</tr>
<tr>
<td></td>
<td>[BE] Timing Diagram</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[BE] State Charts</td>
<td>Developer Testing</td>
</tr>
<tr>
<td></td>
<td>[BE] Sequence Charts</td>
<td></td>
</tr>
</tbody>
</table>

In contrast to traditional, late, and isolated testing approaches, the individual models enable early test case generation (on defined levels of detail) based on models [8]. Major benefits are (a) an increased understanding of the system under development and (b) early test cases applicable in various phases of development within heterogeneous engineering environments.

5. Conclusion and Future Work

The selection of appropriate methods depends on the application context and certain criteria. Based on previous reports on model application ([11] and [13]) we identified four major basic criteria for model application from business and testing perspective: [RE] Systematic definition of requirements; [SA] Structural aspects to capture architecture requirements; [BE] Temporal behavior according to systems requirements; and [RI] Risks as foundation to determine the required scope and level of detail for model application.

*Future work* will include (a) the investigation of modeling needs for TDD in the context of complex project in ASE practice, (b) considering additional models for ASE development (e.g., SysML), and (c) the investigation of the capability of automated test case generation, execution, and reporting.

**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**