Using models in the integration and testing process

Braspenning, N.C.W.M.; van de Mortel - Fronczak, J.M.; Ploeg, van der, D.O.; Rooda, J.E.

Published in:
Tangram: Model-based integration and testing of complex high-tech systems

Published: 01/01/2007

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author’s version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):
Chapter 8

Using models in the integration and testing process

Authors: N.C.W.M. Braspenninig, J.M. van de Mortel-Fronczak, D.O. van der Ploeg, J.E. Rooda

8.1 Introduction

This chapter, which is based on [31], describes how the Model-Based Integration and Testing (MBI&T) method described in Chapter 7 can be used in an industrial integration and test process such as used at ASML. In particular, we focus on a scenario that is common in the integration and test process of many high-tech embedded systems: upgrading a system with new hardware and software to improve the system performance, for example a new sensor with accompanying control software to improve the measurement accuracy. In such a scenario, the goals of integration and testing are to show the functionality and performance of the system upgrade as soon as possible, and to show that the system upgrade does not negatively affect the functionality and performance of the original system. We show how such a scenario is handled in both the current integration and test process and the model-based integration and test process.

Although the use of models in the integration and test process can significantly reduce the integration and test effort, this reduction has to outweigh the additional effort needed to enable model-based integration and testing. For example, the involved components must be modeled, analyzed, and integrated with models or realizations of the other components before the model-based integrated system can be tested. This means that there is a trade-off between the investments and benefits of using models in the integration and test process. In this chapter, we show how such a trade-off analysis can be performed using integration and test sequencing techniques as described in Chapter 6.
The structure of the chapter is as follows. First, Section 8.2 describes the current integration and test process for the system upgrade scenario and introduces the nine test categories that can be distinguished. In Section 8.3, the MBI&T method and techniques are applied to the integration and test process, showing how each of the test categories can be supported by models. Section 8.4 shows how integration and test sequencing techniques can be used to compare the current and model-based integration and test processes, and how this can be used to analyze the trade-off between modeling effort and benefits. Finally, the conclusions are given in Section 8.5.

8.2 Current integration and test process

For the description of the current industrial integration and test process as used within ASML, we consider an existing system that consists of several hardware and software components. The system will be upgraded by implementing some new or improved functionality, which is denoted by a delta sign ($\Delta$). To implement this $\Delta$-functionality, certain components of the original system need to be upgraded, or new components need to be developed and added to the system. Similar to Figure 7.1 in Chapter 7, our view on the development process of this $\Delta$-functionality starts with the global requirements $R_\Delta$ and design $D_\Delta$, as shown on the left hand side of Figure 8.1. After that, the software and hardware components for the $\Delta$-functionality, denoted by $\Delta$SW and $\Delta$HW, respectively, are separately developed. The development process of these components consists of three phases: requirements definition, design, and realization, each resulting in a different representation form of the component, namely the requirements $R_{\Delta SW}$ and $R_{\Delta HW}$, the designs $D_{\Delta SW}$ and $D_{\Delta HW}$, and finally the realizations $Z_{\Delta SW}$ and $Z_{\Delta HW}$.

![Figure 8.1: Current development and integration of a $\Delta$-functionality.](image-url)

The right hand side of Figure 8.1 shows the integration of the $\Delta$ components $Z_{\Delta SW}$ and $Z_{\Delta HW}$ with the other software and hardware components of the (original) system, which we denote as one software component $Z_{{\text{SW}}}$ and one hardware component $Z_{{\text{HW}}}$, respectively. The four components are integrated by means of some infrastructure $I$. Similar to Chapter 7, we abstract from the different forms of infrastructure and only consider the generic infrastructure $I$ (see [30] for more details).
Within the current integration and test process at ASML, nine different test categories can be distinguished, which focus on different aspects of the components or system and require different combinations of realized and integrated components. These nine test categories are listed in Table 8.1, where component integration by means of infrastructure $I$ is denoted by $\{\ldots\}I$.

<table>
<thead>
<tr>
<th>Nr</th>
<th>Test category</th>
<th>Required components</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software qualification</td>
<td>${Z_{SW}, Z_{HW}}<em>I$ and later ${Z</em>{\Delta SW}, Z_{SW}, Z_{HW}}_I$</td>
<td>Periodic qualification of the so-called ‘qualified baseline’ or QBL [59], a common software repository that supports all machine types, by testing it on a set of representative hardware systems $Z_{HW}$.</td>
</tr>
<tr>
<td>2</td>
<td>Software component $Z_{\Delta SW}$</td>
<td></td>
<td>Testing the new software component in isolation.</td>
</tr>
<tr>
<td>3</td>
<td>Software integration ${Z_{\Delta SW}, Z_{SW}}_I$</td>
<td></td>
<td>Testing the new software component in combination with the original software system $Z_{SW}$.</td>
</tr>
<tr>
<td>4</td>
<td>Software regression ${Z_{\Delta SW}, Z_{SW}, Z_{HW}}_I$</td>
<td></td>
<td>Testing whether any of the original system functions are negatively affected by the new software component, performed on the original hardware system $Z_{HW}$.</td>
</tr>
<tr>
<td>5</td>
<td>Hardware component $Z_{\Delta HW}$</td>
<td></td>
<td>Testing the new hardware component in isolation.</td>
</tr>
<tr>
<td>6</td>
<td>Hardware integration ${Z_{\Delta HW}, Z_{HW}}_I$</td>
<td></td>
<td>Testing the new hardware component in combination with the original hardware system $Z_{HW}$.</td>
</tr>
<tr>
<td>7</td>
<td>$\Delta$-functionality test bench ${Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}}_I$</td>
<td></td>
<td>Testing the new $\Delta$-functionality (also called progression testing) on a ‘test bench’, i.e., a partial hardware system including the new hardware component $Z_{\Delta HW}$, which is used for development tests.</td>
</tr>
<tr>
<td>8</td>
<td>$\Delta$-functionality system ${Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}}_I$</td>
<td></td>
<td>Testing the new $\Delta$-functionality on a complete system, i.e., $Z_{HW}$ upgraded with $Z_{\Delta HW}$.</td>
</tr>
<tr>
<td>9</td>
<td>System ${Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}}_I$</td>
<td></td>
<td>Testing the functionality and performance of the complete system with multiple $\Delta$-functionalities before shipment.</td>
</tr>
</tbody>
</table>

Table 8.1: Test categories in the current integration and test process.

Figure 8.2 shows a typical integration and test process for the system upgrade scenario. From left to right, the sequence of test activities, denoted by vertical lines, is shown. The numbers correspond to the nine test categories mentioned above, and the dots indicate which components are integrated and tested. The horizontal lines depict the lifetime of a component: a dashed line means that the component is being developed; a flag symbol followed by a solid line means that the component realization is available. Note that the figure only shows a sequence, and does not contain information on the possible start times and durations of the activities. The flag symbols and the letters indicate the following milestones: (a) QBL $Z_{SW}$ passes qualification tests; (b) development of $Z_{\Delta SW}$ and $Z_{\Delta HW}$ is started, possibly based on the original system (denoted by dashed upward arrows); (c) $Z_{\Delta SW}$ is available; (d) $Z_{\Delta SW}$ passes software tests and is integrated in the QBL $Z_{SW}$ (denoted by downward arrow); (e) upgraded QBL $\{Z_{\Delta SW}, Z_{SW}\}_I$ passes qualification tests; (f) $Z_{\Delta HW}$ is available; (g) $Z_{\Delta SW}$ and $Z_{\Delta HW}$ pass test bench tests; (h) $Z_{\Delta HW}$ passes hardware integration tests.
and the hardware system $Z_{HW}$ is upgraded to $\{Z_{\Delta HW}, Z_{HW}\}_I$ (denoted by downward arrow); (i) similar to the depicted $\Delta$-functionality, the other $\Delta$-functionalities are integrated and tested; (j) complete system with all $\Delta$-functionalities passes tests and is shipped to customer.

Figure 8.2: Typical integration and test process for system upgrade scenario.

The main disadvantage of the current integration and test process is that the test activities can only be performed when the realizations are available. Especially for testing on system level (test categories 7, 8, and 9) this is problematic, because it means that feedback on the system behavior and performance is obtained late in the process, where fixing the problems is expensive. Several case studies, e.g., the one described in Chapter [7], have shown that early integration and testing is possible by using models in the integration and test process, which is discussed in more detail in the next section.

8.3 Model based integration and test process

Figure 8.3 shows the development and integration of a $\Delta$-functionality with the MBI&T method from Chapter [7] with models $M_{\Delta SW}$, $M_{\Delta HW}$, and $M_{HW}$ of the $\Delta$ software component, the $\Delta$ hardware component, and the original complete hardware system, respectively. The reason for having $M_{HW}$ but not $M_{SW}$ is explained in [31]. The choice of integrating either the model or the realization of a component, or none of them, is depicted by the integration ‘switches’.

The MBI&T activities can be applied to all nine current test categories of Table 8.1. In contrast to the current test activities, in which only realizations are used, the model-based test activities can be performed with models instead of realizations, which has several advantages. First, model-based test activities can be performed earlier since, in general, models are earlier available than realizations. Earlier testing means earlier (and thus cheaper) detection and prevention of problems. Second, testing with models is generally cheaper than testing with realizations. For example, testing with models can be performed on a common computer system using modeling and analysis software.
tools. The test costs in such a desktop environment are much lower than the costs of realization tests, especially when these tests require expensive machine time and clean room facilities in the case of ASML. Finally, the models enable the use of powerful model analysis techniques and tools like simulation and formal verification, which provide a better system overview.

Although the MBI&T techniques can be applied in all nine test categories of the previous section, they can not fully replace testing with realizations, since models are always an abstraction of reality. Sooner or later, when the component and system realizations are available, they will also be tested. However, these tests with realizations can probably be performed faster and cheaper, since the model-based tests already prevented several problems. Furthermore, it may be difficult to perform certain tests with models, such that only realizations can be used for these tests. For example, this may be the case for tests involving components with complex physical interactions (e.g., heat, air flow, vibrations) or tests covering multiple system aspects at once.

The following list identifies all possible test activities for each test category, including both the current test activities from Table 8.1 and the model-based test activities of the MBI&T method. For each of the nine test categories, the test activities with realizations only are marked with a ‘Z’, and the model-based test activities are marked with an ‘M’, possibly followed by a letter in the case of multiple model-based test activities. This chapter only describes the model-based test activities of test categories 1, 4 and 7 in more detail. We refer to [31] for a complete overview.

**Software qualification testing:**
- 1Za: \( \{ Z_{SW}, Z_{HW} \} \)
- 1Ma: \( \{ Z_{SW}, M_{HW} \} \)
- 1Zb: \( \{ Z_{\Delta SW}, Z_{SW}, Z_{HW} \} \)
- 1Mb: \( \{ Z_{\Delta SW}, Z_{SW}, M_{HW} \} \)

**Software component testing:**
- 2Z: \( Z_{\Delta SW} \)
- 2Ma: \( M_{\Delta SW} \)
- 2Mb: \( Z_{\Delta SW} \) vs. \( M_{\Delta SW} \)

---

Figure 8.3: Development and integration of \( \Delta \)-functionality in the MBI&T method.
Software integration testing:

3Z: \{Z_{\Delta SW}, Z_{SW}\}_I

3M: \{M_{\Delta SW}, Z_{SW}\}_I

Software regression testing:

4Z: \{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I

4Ma: \{M_{\Delta SW}, Z_{SW}, M_{HW}\}_I

4Mb: \{M_{\Delta SW}, Z_{SW}, Z_{HW}\}_I

Hardware component testing:

5Z: Z_{\Delta HW}

5Ma: M_{\Delta HW}

5Mb: Z_{\Delta HW} vs. M_{\Delta HW}

Hardware integration testing:

6Z: \{Z_{\Delta HW}, Z_{HW}\}_I

6M: \{M_{\Delta HW}, M_{HW}\}_I

\Delta\text{-functionality test bench testing:}

7Z: \{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}\}_I

7Ma: \{M_{\Delta SW}, M_{\Delta HW}\}_I

7Mb: \{M_{\Delta SW}, Z_{SW}, M_{\Delta HW}\}_I

7Mc: \{Z_{\Delta SW}, Z_{SW}, M_{\Delta HW}\}_I

7Md: \{M_{\Delta SW}, Z_{SW}, Z_{\Delta HW}\}_I

\Delta\text{-functionality system testing:}

8Z: \{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I

8Ma: \{M_{\Delta SW}, Z_{SW}, M_{\Delta HW}, M_{HW}\}_I

8Mb: \{Z_{\Delta SW}, Z_{SW}, M_{\Delta HW}, M_{HW}\}_I

8Mc: \{M_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I

System testing:

9Z: \{Z_{\Delta SW}, Z_{SW}, Z_{\Delta HW}, Z_{HW}\}_I

9M: \{Z_{\Delta SW}, Z_{SW}, M_{\Delta HW}, M_{HW}\}_I

In the current integration and test process of ASML, the software qualification tests (test category 1) consume quite some machine time, approximately one full day of testing each week. Besides that machine time is limited and expensive, experience shows that also setting up the system for testing is time consuming. Moreover, much time may be lost on solving minor machine problems that are unimportant for the tests. Test time and costs may be reduced by using hardware models instead of hardware realizations for certain parts of the qualification tests. For example, the qualification of the system throughput in principle depends on the sequence and durations of all hardware actions. When the durations of these hardware actions are modeled as time delays in a model $M_{HW}$ of the hardware system $Z_{HW}$, and when the software $Z_{SW}$ executes the sequence of actions on the model $M_{HW}$, the system throughput can be qualified without a hardware realization $Z_{HW}$. In this way, the software qualification tests can be performed in a cheap desktop environment with a hardware model $M_{HW}$. Furthermore, models require less test setup time, and they do not suffer from the minor problems that may occur in other hardware components, since the hardware model only contains the behavior important for the tests and abstracts from these problems.

A model $M_{\Delta SW}$ of the \(\Delta\) software component can be used as replacement of $Z_{\Delta SW}$ for software regression testing (test category 4), i.e., \{M_{\Delta SW}, Z_{SW}, Z_{HW}\}_I\) instead of \{Z_{\Delta SW}, Z_{SW}, Z_{HW}\}_I\) (4Z). By real-time simulation of the model in combination with the other software $Z_{SW}$, tests can be performed on the original hardware system $Z_{HW}$ to check whether any of the original system functions are negatively affected by the new software component. Similar to model-based software qualification testing in test activity 1Ma, the hardware realization $Z_{HW}$ could also be replaced by its model $M_{HW}$, i.e., \{M_{\Delta SW}, Z_{SW}, M_{HW}\}_I\) (4Ma).

Testing the complete \(\Delta\)-functionality using a test bench (test category 7) can be supported by four model-based test techniques. First, the \(\Delta\)-functionality can be tested by using the integrated models of the \(\Delta\) components, i.e., \{M_{\Delta SW}, M_{\Delta HW}\}_I\), in which $M_{\Delta HW}$ is a model of test bench $Z_{\Delta HW}$. Since only models are used, powerful
model-based system analysis techniques like model checking can be used to exhaustively analyze all possible scenarios, as shown in step 2b of the case study in Chapter 7. Second, the model $M_{\Delta SW}$ can be integrated with the other software $Z_{SW}$, and tested on the test bench model $M_{\Delta HW}$. Third, the realization of the upgraded software system, i.e., $\{Z_{\Delta SW}, Z_{SW}\}_I$, can be tested on the model of the test bench $M_{\Delta HW}$. Finally, in the case that $Z_{\Delta HW}$ is available before the software realization $Z_{\Delta SW}$, the model $M_{\Delta SW}$ can be tested with $Z_{SW}$ on the test bench realization $Z_{\Delta HW}$, as shown in step 3 of the case study in Chapter 7.

Figure 8.4 shows all test activities of the MBI&T process, in a similar way as Figure 8.2. The flag symbols and the letters indicate different milestones, for example: (d) $M_{\Delta SW}$ and $M_{\Delta HW}$ pass model tests; (g) $Z_{\Delta SW}$ passes software tests and is integrated in the QBL $Z_{SW}$ (denoted by a downward arrow); (j) $Z_{\Delta SW}$ and $Z_{\Delta HW}$ pass test bench tests; (k) $Z_{\Delta HW}$ passes hardware integration tests and both $Z_{HW}$ and $M_{HW}$ are upgraded (denoted by downward arrows); (m) complete system with all $\Delta$-functionalities passes tests and is shipped to customer.

As an example, the circles indicate test activities of test category 7, $\Delta$-functionality test bench testing. Their positions in the process clearly illustrate how models enable earlier testing on system level when compared to the current integration and test process, in which only the realization test $7Z$ can be performed late in the process.

---

Although Figure 8.4 shows more activities than Figure 8.2, the length of these figures does not relate to the total duration of the integration and test process since the possible start times and the durations of the activities are not accounted for. For example, in the case that the hardware realization is available earlier than the software (i.e., milestone i. before f.), model-based activities 7Md and 8Mc could be performed before 2Z.
8.4 Process comparison and trade-off analysis

Although all possible integration and test activities have been defined in the previous section, there is still a problem that needs to be addressed before the MBI&T process can be applied to a real integration and test problem. This problem, which also exists in the current integration and test process, is called integration and test sequencing, see also Chapter 6. Integration and test sequencing involves deciding which components to integrate when, and which tests to perform in which order on which components. The goal of integration and test sequencing is to determine a sequence of integration and test activities according to certain optimization criteria like lead time, total test time, test costs, and remaining risk in the system.

For the MBI&T process, there is not only the problem of determining the optimal sequence of the integration and test activities, but there is also a choice of using models for certain integration and test activities or not. This involves a trade-off between the required modeling effort and the potential integration and test effort reduction. In some cases, it might be better to use models, e.g., when the realization of a component is available only late in the process or when testing with realizations is expensive. In other cases, it is wise not to invest in models but to perform the tests with realizations immediately, e.g., when the realization is already available, or when the realization is a mature component, i.e., the probability of finding errors is low and probably not worth the modeling effort.

In this section, we use a basic example to show how the integration and test sequencing method from Chapter 6 can be used for both the integration and test sequencing problem and for the trade-off between the effort and the potential profits of using models. For this example, we defined a fictitious but representative integration and test problem of the system upgrade scenario as used in the previous sections. The problem is instantiated once with realizations only (as in the current integration and test process), and once with the possibility to use models as well (as in the MBI&T process), so that the determined integration and test processes can be compared.

The input for the integration and test sequencing method is an integration model that describes the integration and test problem. Note that this integration model is different from the models $M$ used in the MBI&T method, which describe the behavior of the components to be integrated (process vs. product model; see also Chapter 1). Figure 8.5 and Table 8.2 show the information used for the integration models. Figures 8.5(a) and 8.5(b) depict the components (circles) and interfaces (lines) for the current and for the model-based integration and test process, respectively. The numbers at the top right of each circle denote the development or delivery times of the component. This example uses typical development times for the system upgrade scenario, in which the original hardware and software are available from the start (development time is zero) and the $\Delta$ software component is available after 60 time units, which is 20 time units before the $\Delta$ hardware component. In the MBI&T process, the models of the $\Delta$ components are available after 40 time units.

Table 8.2 shows the available tests in the integration model, including the compo-
nents that need to be available and that must have been integrated for each test, and the test durations. The table contains all 28 test activities listed in the previous section, i.e., including both the current and the model-based test activities. Tests for which only realizations can be used (with a ‘Z’ in the identifier) have to be performed in both the current and the model-based integration and test process. The model-based test activities (with an ‘M’ in the identifier) cannot be executed in the current integration and test process since there are no models available. However, we assume that the aspects covered by a model-based test activity can also be covered by an equivalent test activity using realizations only. This choice of using models or realizations for a certain test activity is denoted by the parentheses in the second column in Table 8.2. For the current integration and test process, only the realization alternatives can be chosen, while for the MBI&T process, both alternatives can be chosen. Taking test category 1 as an example, we see that test activity 1Za requires $Z_{SW}$ and $Z_{HW}$ to be available and to have been integrated. Test activity 1Ma always requires $Z_{SW}$, but gives a choice between using the hardware system model $M_{HW}$ or the realization $Z_{HW}$. For the current integration and test process, only the equivalent realization test with $Z_{HW}$ can be used for 1Ma, while in the MBI&T process, the model $M_{HW}$ can also be used for Ma. In this way, we can use a single set of tests to compare the current and model-based approach. The test durations in the third column of Table 8.2 are fictitious but give a representative distribution of test time over all test activities for an average system upgrade scenario.

Based on an integration model, the integration sequencing algorithm determines all feasible integration and test sequences and determines the best sequence according to the optimization criteria used. In our trade-off analysis, we used the duration of the complete integration and test process, the lead time, as the optimization criterion, since time (in particular time-to-market) is the most important business driver for ASML as explained in Chapter 3. The lead time is different from the total test time, which is
**Table 8.2: Available tests in the integration model.**

The sum of the durations of all separate test activities. By performing the test activities as much as possible in parallel, the total test time remains equal but the lead time is reduced.

Figure 8.6 shows the determined optimal sequences for the current (top) and model-based (bottom) integration and test processes, in the form of an MS Project Gantt Chart. The figure shows all development activities (dashed bars), integrations (diamonds), and test activities (solid bars) over time, and the precedences between the activities (arrows). On the first lines of the integration and test sequences, the long white bar with triangular ends indicates the lead time of the sequence.

Several conclusions can be drawn from these sequences. First, the lead time of the total MBI&T sequence is shorter, 167 time units against 188 time units for the current integration and test sequence, a reduction of 11%. Besides lead time, also the duration of the final system test phase (the long solid bars at the right hand side of Figure 8.6) is important for ASML, since this phase is on the critical path and has a major influence on the time-to-market T. The final system test phase is 78 time units for the MBI&T sequence, 25% less than the 104 time units for the current integration and test sequence.
Figure 8.6: Current (top) and model-based (bottom) integration and test sequences.
Besides lead time, also the total test time and related costs can be used to compare both approaches. Looking at the total time spent on testing with realizations and models in this example, we see that the current integration and test process uses 136 time units of realization testing. The model-based integration and test process uses 92 time units of realization testing and 44 time units of testing with models. At ASML, the costs per time unit for realization testing are orders of magnitude higher than for testing software or models in a desktop environment. In this example, the reduction of test costs clearly outweighs the onetime investments needed in model development, which are 80 time units of modeling in a (cheap) desktop environment.

Finally, the integration and test sequences show that the test activities in the MBI&T process can be performed earlier and more in parallel by using models, see for example the position of the circles in both sequences, indicating when the tests of test category 7 are performed. This means that design and integration problems can be detected and prevented at an earlier and cheaper stage of development. Although this cannot directly be expressed in terms of test time or costs, the advantages of earlier testing can be explained in terms of quality and risk. By incorporating risk into the integration and test model, this can be dealt with analogous to Chapters 5 and 6.

For simplicity, the trade-off analysis example in this chapter only used a basic integration model. However, the integration model can be extended in several ways to perform a more detailed analysis. For example, by including test selection from Chapter 5, the test time and cost differences between testing with models and realizations can be incorporated in the model and in the decision making process, as well as test coverage and the risks of faults in the system. In this way, decisions like longer but cheaper model testing or shorter but more expensive realization testing can automatically be made by the sequencing algorithm. Also ‘what if’ scenarios can be investigated, for example the effects of developing more detailed models, which means higher model development times, but also a higher coverage of the model-based test activities and less test activities that can only be performed with realizations. This provides a systematic and automatic method for improving the current integration and test process by applying models at places where it is possible and profitable.

8.5 Conclusions

This chapter described an integration and test process for a system upgrade scenario that is common in industry, including nine different test categories that cover different system aspects. Since tests can only be performed with realizations, the test costs are rather high and the tests can only be performed late in the process, where fixing problems is expensive.

We applied the MBI&T method of Chapter 7 to the current integration and test process, resulting in additional model-based test activities that allow earlier, cheaper, and more parallel testing. The feasibility and potential of several of these techniques have already been demonstrated in industrial case studies such as the one of Chapter 7.

By using the integration and test sequencing technique from Chapter 6 we showed...
how optimal sequences of integration and test activities can be determined and how the trade-off between the effort and potential benefits of using models for integration and testing can be analyzed. The results of a basic system upgrade example show that the lead time and costs of the current integration and test process can be reduced by performing certain tests earlier with models. Several extensions can be applied in order to perform automatic trade-off analysis for real integration and test problems.