White paper

how to Make the Most of Model-based Testing

Introduction

Test automation based on tools like HP QuickTest Professional® is well established and successfully used in many software organizations.

But though the introduction of test automation has its benefits in promoting regression testing, we also observed some weaknesses concerning the creation and maintenance of the test cases:

- SAutomated test cases are designed by hand, often without taking specific test case design methods into account
- The implementation of automated test scripts requires advanced programming skills
- The maintenance of automated test cases after changes to the SUT (Software Under Test) is difficult and time-consuming.

Model-based testing (MBT) has the potential to resolve these issues. However, it is often still regarded as too academic and therefore as not suitable for testing in a real-world project where resource, cost and time constraints apply.

The most common way of getting a test model for MBT is to reuse a system model that has been created for development. However, this method has serious disadvantages:

- If both the source code and the test cases are generated from the same model, no deviations will be found. Defects in the system model will remain undetected
- The system model is created with a focus on development, not on testing. Thus, it includes a wealth of information that is irrelevant to testing while missing other data that is required
- Many models in notations like UML lack the level of formality required for automated test case generation, e.g., because information is contained in plain text annotations. UML extensions can mitigate this problem, but add a further level of complexity to the model.

For all these reasons, it is recommended to create a separate model focused on the purposes of testing. It should only contain information relevant to testing and be just detailed enough to generate test cases and test data. If the notation used for MBT is understandable for system testers, they are more likely to use it. Most system testers nowadays are familiar with use cases, activity diagrams and Boolean expressions.

Systematic test design techniques like equivalence partitioning, boundary value analysis, and cause-effect analysis are often not used exhaustively enough by the testers. MBT fosters the application of these techniques and as a consequence improves the capability of the test cases to detect specific categories of defects. Despite the initial effort needed to create a test model and integrate MBT into the test process, the improved effectiveness and maintainability of the test cases will be appreciated by all stakeholders.

This article is structured as follows: In the next section we present the recommended workflow and give practical tips for the efficient usage of MBT. Subsequently, we introduce the MBT tool Atos TEMPPO Designer, and show how it supports the MBT workflow using a practical example. Finally, we give a conclusion and an outlook on our future work.

Contents

Workflow and Practical Tips for Model-based Testing

In this section we describe the steps required to successfully introduce MBT into your project and also provide a number of practical tips. But before reading on, you should be aware of the first pre-requisite for applying test automation in general, and MBT in particular: a well-defined software development process that guarantees the adequate management of project tasks; requirements; test artifacts; and change requests. If the process in place is not at the required maturity level, there is a strong probability that MBT will not work. Fast and reliable communication channels between the testers and the rest of the project team are an essential feature of such processes (see Figure 1):

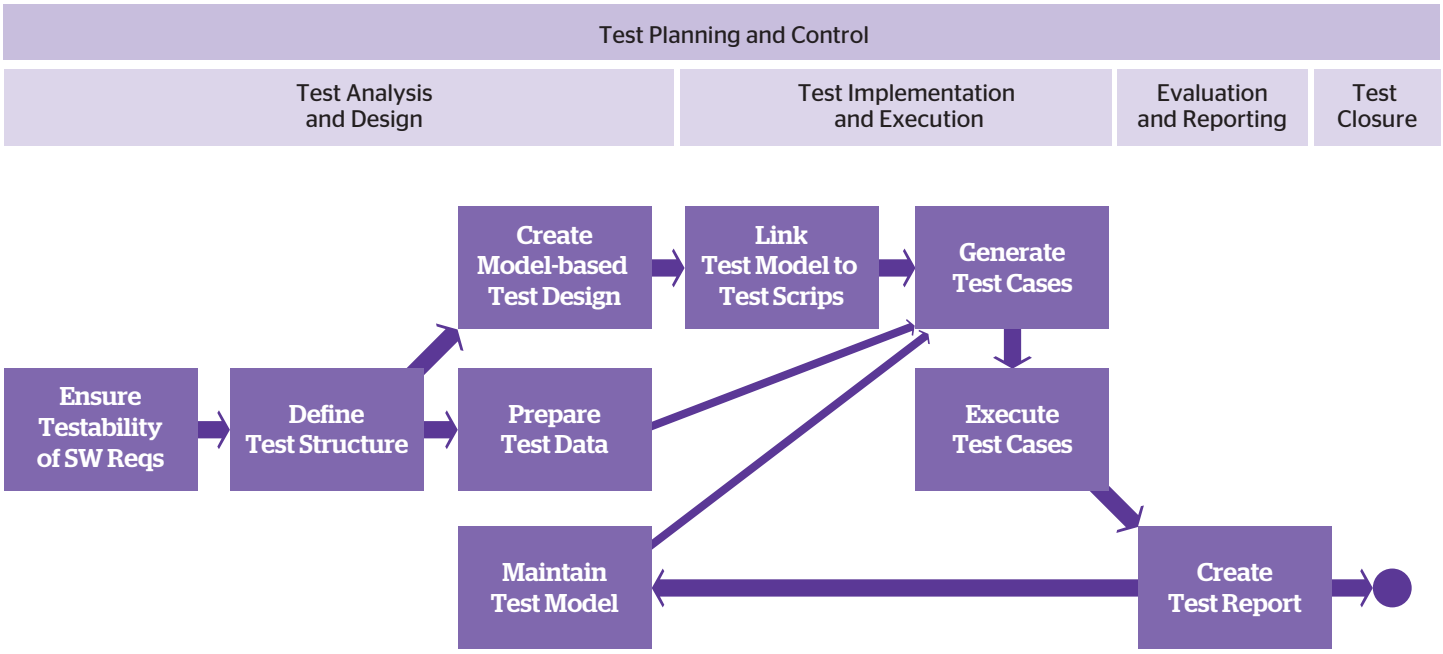

Figure 1: MBT-related activities in relation to the ISTQB® test process

Ensure Testability of Software Requirements

While requirements engineering is usually not considered a part of the test process, it is, nonetheless, crucial for the subsequent testing phases. There are some points that have to be taken into account already at this early stage to ensure the success of the testing activities, especially if MBT is going to be applied.

Testability comprises two aspects:

1. Ensuring that the input documents for the test process have the required quality level
2. Ensuring that the system under test fulfills all technical pre-requisites for test execution.

Roles
Requirements Engineer,
Test Manager,
Test Analyst,
Test Automation Specialist

Input
User Requirements

Output
SW Requirements,
Testability Analysis Report

Practical Tips

- Involve test experts in early reviews of requirement and design specifications in order to avoid testability problems later on
- A pre-requisite for MBT is that the requirements are detailed and clear enough that a formal model can be derived from them
- Try to use activity diagrams, state diagrams, usage scenarios etc., instead of plain text
- Make sure to plan interfaces that can later be used by test automation tools to access the system under test.

Define Test Structure

Based on the requirements, a test structure consisting of test packages and test cases is derived. This should be done with a professional test-management tool that provides convenient interfaces to the other tools involved in the test process (requirements management, model-based test design, test execution, defect management, etc.).

Roles
Test Analyst

Input
SW Requirements

Output
Test Structure

Practical Tips
- Don't only focus on the required functions, also plan tests for the non-functional requirements
- Decide for which test cases model-based design shall be used. Typical candidates are tests for requirements that can easily be formalized as a model. On the other hand, for some requirements MBT might not make sense
- In addition, don't forget to plan some experience-based tests in addition to the systematic tests. A test model is a good thing but it cannot replace human experience and intuition!

Create Model-based Test Design

In contrast to an ordinary test project where test cases are designed e.g., in a tabular format, a model-based test project requires the creation of a test model. This model depicts the system behavior from a testing perspective and contains all scenarios, constraints, and dependencies that are relevant to testing. Modeling can require considerable effort, especially if the input documents do not have the required level of detail and frequent clarifications are required. On the other hand, this is an excellent opportunity to detect defects and inconsistencies at an early stage even before the implementation phase. Therefore, this initial investment helps to avoid serious problems in later stages of the project and greatly enhances the maintainability of the tests.

Nonetheless, it has to be checked beforehand when the ROI will be reached and if MBT makes sense for a certain test item.

Roles
Test Analyst,
Requirements Engineer,
Software Designer

Input
SW Requirements,
Test Structure

Output
Test Model

Practical Tips
- Avoid reusing a model that has also been used for code generation. If the code and the tests are generated from the same model, no deviations will be found! Try to create a specific model from the testing perspective instead
- Creating a test model is a challenging task that requires many different skills (test design, abstraction, understanding of requirements, technical background). Don't expect end users to perform this task but employ specialists instead!
- Start modeling at an abstract level, then add more detail step by step
- Make sure that requirements engineers and software designers are available to clarify any questions from the test designers
- Pay attention to the reusability and maintainability of the model components, e.g., use parametrizable building blocks.

Link Test Model to Executable Test Scripts

If the test execution shall be done automatically, the abstract test model has to be linked to a test script containing concrete instructions for a test execution tool. A possible way of doing that is to implement an adapter or script for each keyword that is used in the model. However, this requires considerable effort and advanced programming skills.

A much better alternative is to use an MBT tool that can import GUI information and generate executable scripts automatically. In this case, each step in the model only has to be assigned to the affected GUI element.

Roles
Test Analyst,
Test Automation Specialist

Input
Test Model

Output
Detailed Test Model / Test Script for each Keyword

Practical Tips
- If using a professional test generator that generates complete scripts (e.g., TEMPPO Designer), record GUI information and assign steps to GUI elements
- If using a generator that only produces abstract keywords, implement an adapter or a script for each keyword.

Prepare Test Data

Depending on the requirements it has to be decided whether test data can be imported from an existing source, has to be created by hand or will be generated automatically. In the latter case, some MBT tools provide the possibility to specify data dependencies as part of the model and to produce data accordingly.

Roles
Test Analyst

Input
SW Requirements,
Test Structure

Output
Test Data

Practical Tips
- Try to cover both test data and test sequence-related aspects in your model (e.g., define equivalence partitions for test data, activity diagrams for sequences). Finally, connect the two aspects by specifying which data is used in which step of the sequence
- Simple test data design methods like equivalence partitioning and boundary value analysis should only be applied for data fields that do not have any dependencies with others. For all othercases, methods like cause/effect analysis or CECIL should be applied.

Generate Test Cases

Based on the test model, test cases can be generated automatically. Depending on the model type and the generation tool, the user can choose between various generation strategies and coverage criteria. While many tools can only generate abstract test cases consisting of keywords or instructions for manual testing, others can produce complete, executable scripts.

Roles
Test Analyst

Input
Detailed Test Model-/-Test Script for each Keyword

Output
Executable Test Scripts

Practical Tips
- Choose test coverage based on model type and criticality of the associated requirements.
- Combine different test methods to increase defect detection potential. Use random generation in addition to systematic methods
- The integration should make it possible to transfer the generated test cases directly to the test management tool.

Execute Test Cases

In some projects, the test generator produces textual instructions for manual testing. However, more frequently, the test execution is performed automatically by a tool. In both cases, appropriate interfaces for sending test commands to the system under test and verifying the results have to be available.

Roles
Tester, Test Analyst

Input
Executable Test Scripts

Output
Test Protocols, Defect Reports

Practical Tips
- Don't expect test automation to work immediately – it's rather an iterative process. Verify if the generated test cases work as expected (GUI object recognition, timing, etc.). Correct the model where necessary until the tests are running smoothly
- Test scripts derived from a model usually require that the system under test is in a well-defined state at the beginning of each test and behaves in a predictable way.
  For instance, alterations in the database or unforeseen pop-up messages may cause the scripts to fail if they are not considered in the model
- In cases of deviation from the expected behavior, it has to be determined whether the problem was really caused by a bug in the software or on the contrary by a defect in the test model.

Create Test Report

The test protocols created during the test run have to be transformed into a concise test report that informs the responsible managers about the current project status.

Roles
Test Manager

Input
Test Protocols, Defect Reports

Output
Test Report

Practical Tips
- For MBT it is important that statistics of the model coverage are included (e.g., the percentage of states, transitions, or scenarios that have been covered by the tests)
- Many test management tools can generate such reports, provided that they have access to all the required data. Again, a good integration with other tools is important.

Maintain Test Model

For each new version that has to be tested, the tests have to be adapted and repeated. Updating each single test script by hand is impractical as it can require an enormous effort. On the other hand, updating a model consisting of reusable components is much less tedious, since only a few parts have to be changed before generating perfectly updated test scripts.

Roles
Test Analyst

Input
Test Model, Change Requests

Output
Updated Test Model

Practical Tips
- It is essential that the test team is informed about all changes in the system under test. Even small modifications that are invisible to the user can cause an automated test run to fail
- The effort invested into a well-structured model design will pay off now. If designed properly, only some parts of the model have to be adapted. Afterwards, all test cases can be updated automatically by re-generating them.

Model-based Testing with Atos TEMPPO Designer

While it is relatively easy to find tools for automated test execution, there are significantly fewer that focus on model-based test generation. They can be categorized according to the method used for test generation: *data-oriented* tools and sequence-oriented tools. A tool that unites both features and can also be easily integrated into a project's tool framework is **Atos TEMPPO Designer**; this tool was previously called **IDATG** (Integrating Design and Automated Test case Generation).

When the research project IDATG started in 1997, the aim was to develop a simple tool for facilitating the maintenance of SilkTest® and WinRunner® scripts for GUI testing. Over the years, IDATG was steadily expanded to support an increasing number of test methods and output formats, as well as testing via non-GUI interfaces. In 2004, IDATG became part of the test framework of the European Space Agency ESA (this success story can be found in Dorothy Graham's new book *"Experiences of Test Automation"* [Graham et al. 2012]).

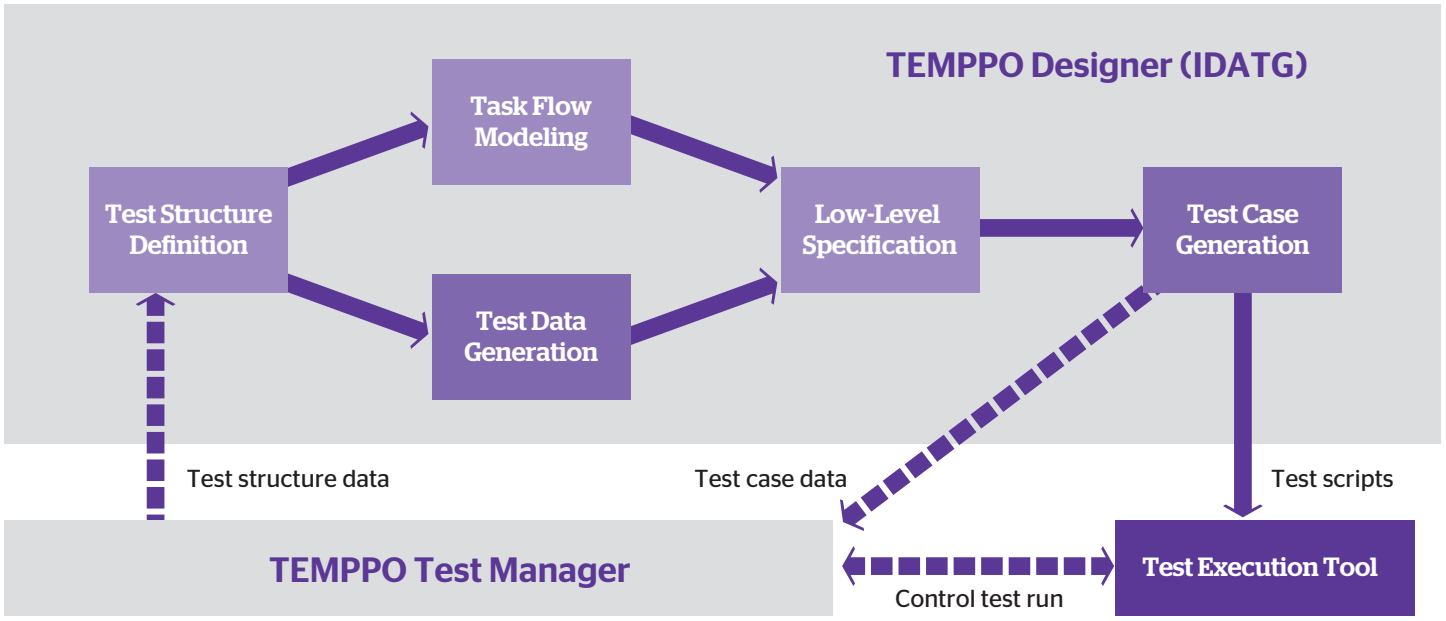

Figure 2: Model-Based Testing with TEMPPO Designer

Interfaces for Model Creation

Many MBT tools do not have a model editor of their own, but instead rely on other sources from which models are imported. However, reusing models that have not been created with a focus on testing has several disadvantages. Quite often such models are not formal enough (e.g. plain text annotations) or they contain a lot of information that is irrelevant to testing while missing important data. To overcome these problems, TEMPPO Designer has a built-in model editor that provides an independent way of creating test models. It uses a specific notation that is simple to learn, and it focuses on the necessities of testing (see Figure 4 in the next chapter). Test sequences are represented as *task flow diagrams* that consist of simple test steps (blue) and parametrizable building blocks (yellow) that represent reusable sub-sequences of steps. For GUI testing, steps can be assigned to GUI objects using the built-in GUI Spy.

Apart from common testing methods like equivalence partitioning, a number of innovative test strategies have been developed for the tool. These include CECIL (Cause-Effect Coverage Incorporating Linear boundaries [Beer & Mohacsi 2008]) and a hybrid algorithm for random testing (Mohacsi & Wallner 2010).

Interfaces for Test Execution

A major drawback of many test generators is that they only produce abstract test cases that still require manual completion, for instance by implementing a script for each keyword. A distinguishing feature of TEMPPO Designer is that its output are complete, executable scripts for a variety of popular test execution tools including HP QuickTest Professional® and the Micro Focus tools SilkTest® and TestPartner®. However, the tool can also be used for producing test cases that are executed manually or over a non-GUI interface.

This direct integration between test generation and execution has proven to be a considerable advantage in that it delivers a significant reduction in test maintenance costs; i.e., instead of having to update each single test script for every new version of the system under test, it usually suffices to change a few building blocks inside TEMPPO Designer and let it generate new test cases. A case study from an ESA project has shown that the additional effort for introducing model-based test automation paid off after only four test repetitions (Graham et al. 2012).

Interfaces for Test Management

Integration of test generation with test and requirements management is equally important. As its new name suggests, TEMPPO Designer is a part of the TEMPPO suite that also includes the test management tool **TEMPPO Test Manager**. Information can be passed in both directions: Test structures and requirements created in Test Manager are passed to Designer. After the test model has been created, the generated test cases are passed back to TEMPPO Test Manager which ensures proper versioning, traceability, execution planning, and reporting. There is also an interface for HP Quality Center® that allows the user to import TEMPPO Test Designer test cases via Excel.

Practical Example

We are going to demonstrate the most relevant steps of the MBT workflow using a simple application for managing tennis players and tournaments. While this is a hypothetical example, it includes many experiences from real projects. It also introduces the usage of the Atos TEMPPO tool suite, and shows how to efficiently use it for model-based test design and generation.

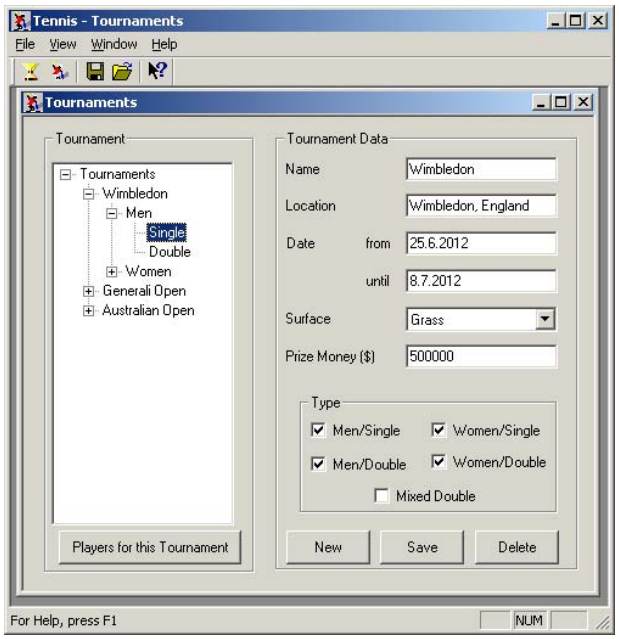

Figure 3: Example Application "Tennis"

Ensure Testability of Software Requirements

In the first application design, the use of the GUI library "FlashyGUI" was envisaged by the designer who had found it on the Internet and was eager to try it out. Fortunately, the test automation specialist was involved in the design review and pointed out that this technology was not supported by any test automation tool on the market, thus making automation impossible. Finally, it was decided to use a standard Win32 GUI instead.

Define Test Structure

Once the requirement and design specifications had been reviewed, the test analyst began creating a tree-like test structure in TEMPPO Test Manager. He started with typical usage scenarios like "Create Player," then added tests for non-functional requirements (e.g., performance of the search function) and error cases (e.g., assigning a male player to a female tournament). Based on his ample experience from past projects he also planned a few scenarios simulating unusual user behavior.

Create Model-based Test Design

Using TEMPPO Designer, the test analyst began to break down each scenario of the test structure into a rough sequence of test steps. While doing this, he identified recurring steps like "Edit Player" and defined them as re-usable building blocks with parameters. These rough building blocks were further refined, for instance, "Edit Player" consisted of the blocks "Edit Personal Data," "Edit Career Data," and "Edit Ranking." This refinement was iteratively repeated until ending up with single-user actions like entering the first name or clicking the OK button.

The user action represented by each step could be entered in a generic "event language", for instance *Click* or *Input ("Serena")*. However, instead of concrete texts, mostly references to parameters were used to make the model more flexible: *Input (#@EditPersonalData:FirstName#)*

Semantic conditions were also considered in the model. For instance, a middle initial was only entered if the corresponding parameter was not empty. This could simply be specified in the form of a Boolean expression: *#@EditPersonalData:Initial# != ""*.

Also, the expected results and reactions of the system were specified. This could easily be done with events like *Verify (CAPTION, "Player data has been saved.")*.

However, the specification documents on which the model was based seemed to be incomplete. For instance, it was not clear how the system should react if a man was added to a female tournament. Maybe the designer had assumed that the programmer would surely consider this obvious case and it was therefore unnecessary to mention it. Fortunately, the test analyst could contact the designer directly and ask him. It turned out that this case had simply been forgotten in the design and that the programmer, who had no interest in tennis at all, hadn't thought of it either. Since this defect had been found in an early stage of the development, it could quickly be fixed and there was little harm done. However, if it had found its way into the finished product, this embarrassing mistake could have damaged the company's reputation.

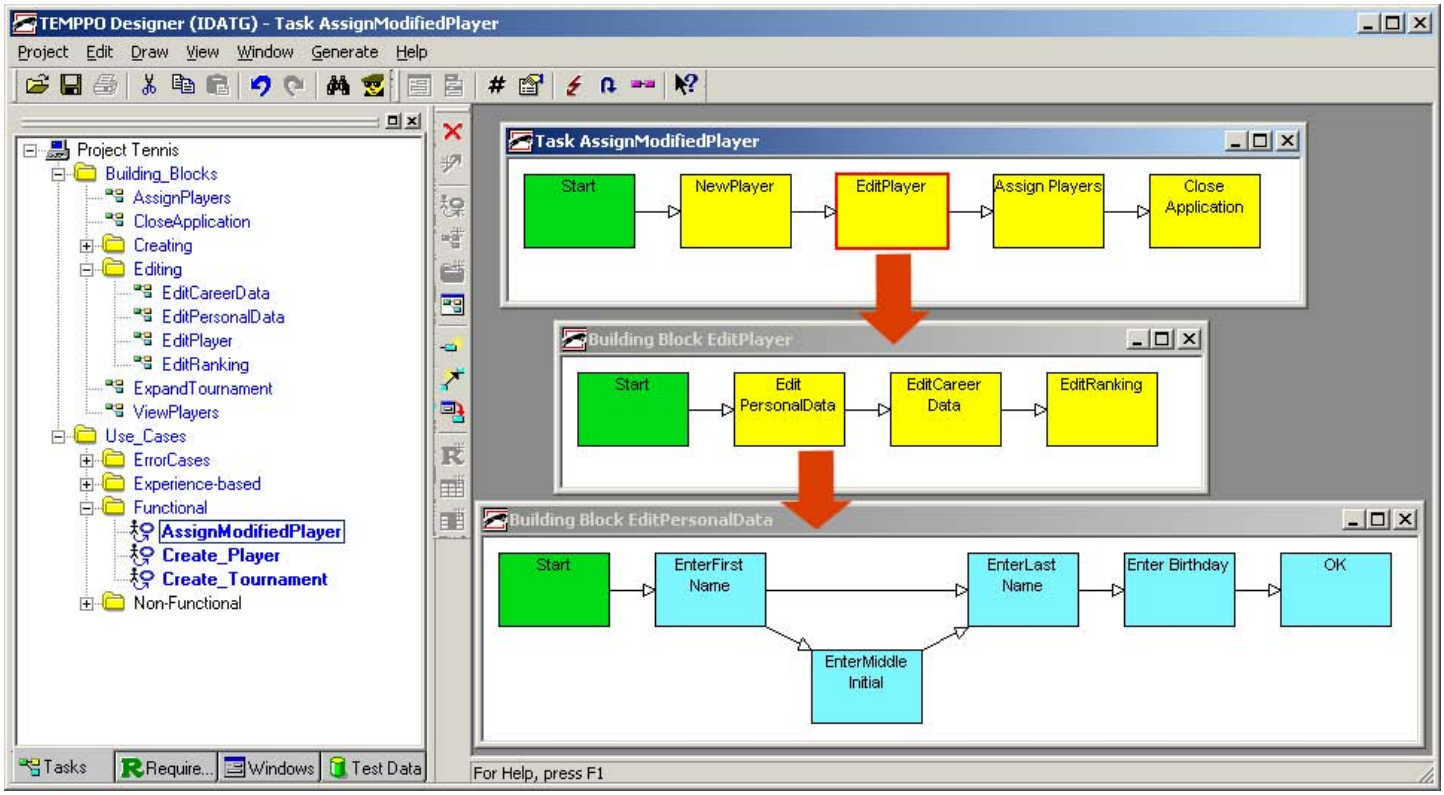

Figure 4: Task Flows in TEMPPO Designer

Link Test Model to Executable Test Scripts

Once the GUI of the tennis application had been implemented, its window could be recorded with the built-in GUI Spy of TEMPPO Designer. Each blue test step was assigned to a certain object in the screenshot, for instance, "Enter-FirstName" to the input field "FirstName". Since the GUI Spy did not only capture the appearance but also the technical details of the GUI objects, no further information was needed to produce a complete object repository for the test execution tool (in this case, HP QuickTest Professional®).

Also, the generic events like *Input* would later be translated automatically into the tool-specific language (for QuickTest, *Input* would be translated into the *Set* command). Only a few steps involved more complex actions that could not be expressed in the generic language but had to be defined in QuickTest code. An example was checking the alphabetical order of the player list, a verification that had to be implemented as a QuickTest function. Apart from this, no other action was required to link the model to the finalized test scripts.

Prepare Test Data

Most information about the tennis players was imported from an Excel sheet showing the current ATP ranking. In addition, equivalence partitioning and boundary analysis were applied to generate test data, thus covering all relevant cases for fields like the Prize Money. In more complex cases, dependencies were specified, e.g., that the start date of a tournament must lie before its end date. Finally, it was specified which data was used in each test step simply by inserting references to the data tables.

Generate Test Cases

Once the first version of the model was completed, our test analyst was able to start the script generation. He chose a combination of graph-oriented tests covering all step connections, data-oriented tests covering all tennis players, and a few random sequences each consisting of 1000 steps. The output was a complete set of test scripts and an object repository for HP QuickTest Professional®. Also, information about the generated test scripts was transferred to TEMPPO Test Manager.

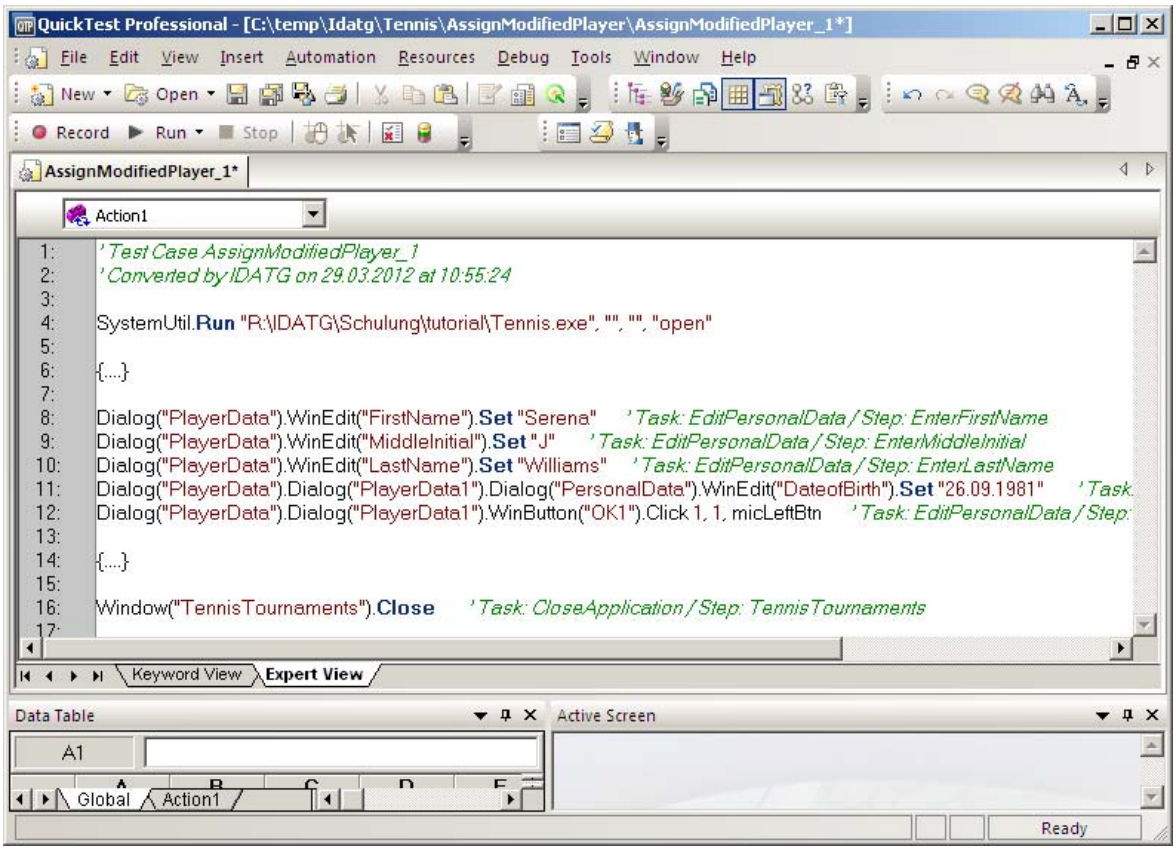

Figure 5: Excerpt from a Generated Script in QTP

Execute Test Cases

Via TEMPPO Test Manager, QuickTest was opened to start a first trial run of the scripts. In the course of this it was noted that a few scripts failed because sometimes a timeout occurred when the tennis application connected to its database. The test analyst had to insert a higher delay value in the corresponding step of the test model and re-generate the scripts. In the next trial run it turned out that the tree control displaying the tournaments had changed its ID and could thus not be found any more by QuickTest. The test analyst solved this by removing the ID from the set of properties that was used for object identification. Again, this was done in the model, not in QuickTest itself. Finally, after re-generating the QuickTest files, the tests did run smoothly.

Once the test analyst decided that the scripts were ready, the tester could begin his task. He ran the entire test suite and evaluated the test protocols that were produced by QuickTest. Several deviations from the expected behavior were found, for instance, it was possible to enter a negative value for the prize money. One of the random sequences even caused the application to crash when it tried to assign Serena Williams and her sister Venus to the same tournament.

Create Test Report

The test results and defect reports were then comprised by the test manager into a test report using the reporting capabilities of TEMPPO Test Manager. The test manager also included information about the model coverage, especially the percentage of covered scenarios, step connections, and data records. This gave him an excellent position in the next meeting with the customer who wanted to know exactly about the test progress and was impressed by the accurate data.

Maintain Test Model

Every two weeks, a new version of the tennis application had to be tested. Since there were more than 500 test cases, adapting them by hand for each version would have been a nightmare. But thanks to the model-based testing approach, the test analyst only had to adapt a few building blocks in the model and re-generate the scripts. Even when suddenly a major change of the rules for Grand Slam tournaments was announced by the International Tennis Federation, it sufficed to adapt a few steps and conditions.

Conclusion, References and Acknowledgment

Conclusion

In this article, we presented a workflow for MBT that considers the traceability between requirements, the test model, the GUI, and the test cases.

The workflow is supported by the Atos TEMPPO framework which has the advantage of using a simple model notation specifically designed for the purposes of testing. Also, instead of generating only abstract keywords from the model, the tool produces complete, executable test scripts for tools like HP QuickTest Professional® (QTP).

The generation of test cases is systematically increasing the effectiveness of finding relevant defects in the SUT. Thus, the confidence in the product quality is increased and the number of unplanned releases and hot fixes can be reduced significantly. The effort required for defining and maintaining the test cases can be limited to a reasonable level.

Applying MBT using TEMPPO Designer has become a major focus of Atos. A global Competence Center for MBT has recently been established with the task of promoting the topic within the company. Our future plans include a closer coupling of TEMPPO Designer with other test tools. The newest version already supports the import of object repositories from QTP. The next step will be the import of recorded actions that can then be converted into parametrized, reusable building blocks.

References

[Graham et al. 2012] D. Graham and M. Fewster, Experiences of Test Automation, Chapter 9, p. 155-175, Addison-Wesley, 2012.

[Beer & Mohacsi 2008] A. Beer and S. Mohacsi, Efficient Test Data Generation for Variables with Complex Dependencies, Proceedings of the IEEE ICST, Lillehammer, Norway, 2008.

[Mohacsi & Wallner 2010] S. Mohacsi and J. Wallner, A Hybrid Approach to Model-Based Random Testing, Proceedings of the Second International Conference on Advances in System Testing and Validation Lifecycle, 2010, pp. 10-15, 22.-27. August 2010.

About the authors

Armin Beer has been working in the area of test management and test automation for about 20 years. He is currently an independent consultant for the test management group at the social insurance institution BVA. He is also participating in the research project SoftNet of the Technical University of Graz and lecturing at the University of Applied Sciences in Vienna and the Technical University of Graz. Armin's email address is beer@arminbeer.at.

Stefan Mohacsi studied computer science at the Technical University of Vienna. In 1997, he joined Siemens and became project manager of the research project IDATG. Today, Stefan is senior consultant for model-based testing at Atos. In addition, he is a member of the Austrian Testing Board and has held numerous lectures at international test conferences. Stefan's email address is stefan.mohacsi@atos.net.

Acknowledgment

The research herein is partially conducted within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

Some parts of this work are based on an article we published in the March 2012 edition of the *Testing Experience* magazine.

About Atos

Atos is an international information technology services company with annual 2011 pro forma revenue of EUR 8.5 billion and 74,000 employees in 48 countries. Serving a global client base, it delivers hi-tech transactional services, consulting and technology services, systems integration and managed services. With its deep technology expertise and industry knowledge, it works with clients across the following market sectors: Manufacturing, Retail, Services; Public, Health & Transports; Financial Services; Telecoms, Media & Technology; Energy & Utilities.

Atos is focused on business technology that powers progress and helps organizations to create their firm of the future. It is the Worldwide Information Technology Partner for the Olympic and Paralympic Games and is quoted on the Paris Eurolist Market. Atos operates under the brands Atos, Atos Consulting & Technology Services, Atos Worldline and Atos Worldgrid. For more information, visit: atos.net

For more information:
Please contact **dialogue@atos.net**