White paper

integrating model-based testing in a project's tool landscape

Your business technologists. **Powering progress**

Atos

Introduction

Software testing is a knowledge-intensive task that is driven by the information flow generated from the creativity, motivation and cooperation of the involved people.

Repetitive mechanical activities, manual steps in automated processes, an unaligned infrastructure and disruptive workflows create painful blockers for this information flow. As some test-related tools have their own proprietary data-storage approach, the resulting isolation makes sharing information increasingly difficult, and tools that are not aligned properly may lead to unneeded complexity and manual test-related activities [Hüttermann 2011].

Hence, the creation of test cases alongside the corresponding test data is increasingly supported by model-based testing (MBT) tools. MBT promises (1) the detection of faults at an early stage in the development cycle and (2) reduces the maintenance effort of test scripts as the test cases are deduced from the model. Mature processes [van Ewijk 2011] and a smooth integration of the underlying tools [Turlo & Safronau 2011] allow test engineers to get rid of mechanical (test-related) activities and focus their time on the development of models for the generation of test cases.

Impacts of an inadequate software testing infrastructure

The infrastructure for testing has considerable influence on the costs for finding and fixing errant behavior in software. Based on software developer and user surveys, in the US, the annual cost of inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion. Over half of the costs are borne by software users in the form of error avoidance and mitigation activities. The remaining costs are borne by software developers; this reflects the additional testing resources that are consumed due to inadequate testing tools and methods [NIST 2002].

The authors of [Turlo and Safronau 2011] present a well-timed survey (see Figure 1) highlighting that many everyday routine activities in automated testing are still performed manually. The results provide empirical proof of inadequate testing infrastructure due to a lack of appropriate integration functionality.

In the automated testing, which operations do you still have to perform manually after the build is ready? (multiple answers)

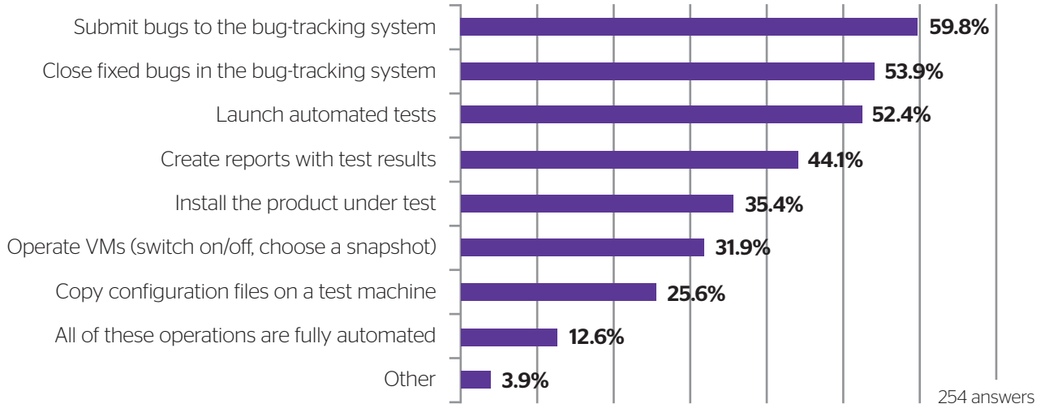

Figure 1: Poorly integrated test infrastructure – top testing activities performed manually during automated testing [Turlo and Safronau 2011]

The online survey on challenges of automated software testing was provided to 11.895 readers of the testing blog hosted at www.habrahabr.ru

From May to June 2011, 299 respondents participated in the survey, offering valuable insights into the level of maturity of automated software testing and tool infrastructure in use. Regarding the job distribution of the respondents, 49.3% are testers, 15% – test leads, 11,9% – heads of QA/testing departments, 23.8% – other, mostly software developers. As for the type of software their organization develop, 65.8% of the participants answered "web" applications, 58.5% – desktop software, 22.1% – mobile applications, and 12.4% – embedded software (the question allowed for multiple answers). Almost 45% of the respondents actively use automated tests in the process carrying out testing and, around 27% use them occasionally, while the remaining 28% – rarely or never.

To evaluate the extent of manual effort expended in automated testing, the participants were asked to answer the following question: "During the automated testing, what are the operations that you still have to perform manually?" This particular question received 247 answers, - 45 respondents chose to not provide a response.

According to the poll, only 12.6% of respondents claim that all the operations related to test execution are performed automatically in their organization, i.e., without human intervention. As Figure 1 shows, the most common tasks that a tester has to carry out manually are submitting and closing bugs in the bug tracking systems (59.8% and 53.9%, respectively), launching automated tests (52.4%) creating reports (44.1%), and installing the product under test (35.4%). As the question allowed for multiple answers, the total percentage exceeds 100 %. These routine menial operations, depending on the number of tests to be executed and the frequency of new builds, might consume as much as 25% of tester work time and result in inefficient use of testing resources.[1]

These results along with findings in [NIST 2002] are evidence of the lack of proper cohesion between tools and systems used in software development and testing processes, namely build machines, versioning control systems, virtualization servers, bug tracking systems, and test automation tools. Improving the integration of the testing tools reliefs the test engineer from various manual tasks and thus his/her time and skills can be better allocated, for example, to making model-based testing (including the creation of adequate models) an integral part of the development and test lifecycle.

With the proliferation of test process improvements we have an instrument for systematically setting up a test process with adequate levels of maturity. For example, leveraging model-based testing requires establishing a certain maturity regarding different aspects of the test process [TE 2011, van Ewijk 2011]. As in enterprise application integration (EAI), on top of well-defined and established processes, automation and integration of tools are most beneficial. This observation holds true for the test process as well. Our hypothesis is that applying the successful integration concepts from the EAI field [Peischl et al. 2011] and integrating various testing tools will result in increased efficiency in software testing [Turlo and Safronau 2011].

Workflow and practical tips for integrating MBT in a project

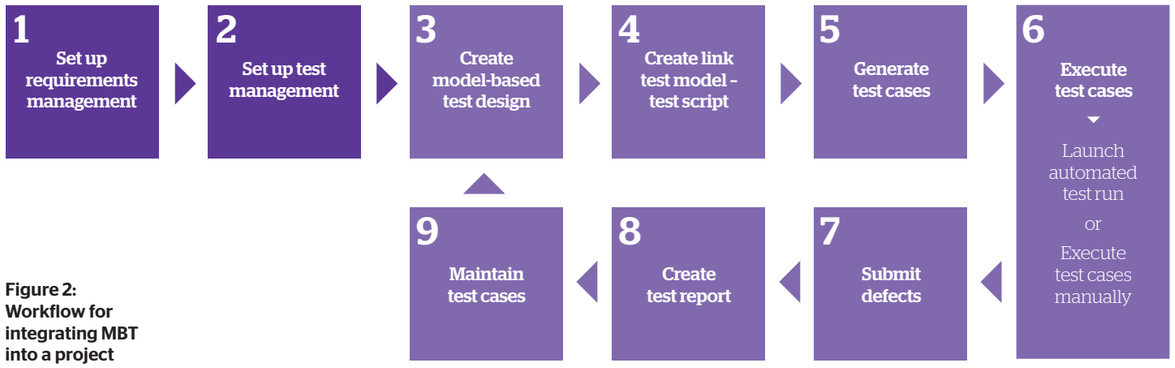

Figure 2: Workflow for integrating MBT into a project

In this section we describe the steps required to successfully introduce MBT into your project and also provide some 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 artefacts 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 2):

1. Set up requirements management

- 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.
- Also involve test experts in early reviews in order to avoid testability problems later on.

2. Set up test management

- Based on the requirements, create a test structure consisting of test packages and test cases. 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.
- 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!

3. Create model-based test design

- A proper integration with the test management tool is important, e.g., for importing information about requirements and test packages.
- 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 modelling 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.
- 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.

4. Create links between the test model and executable test script

- 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.

5. Generate test cases

- 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.

6. 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.
- The test execution tool should be fully integrated into the test management framework. It should be possible to start test runs and store the results automatically from within the test management tool.
- 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.

7. Submit defects

- 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 conversely by a defect in the test model.
- The flawless integration between test execution, test management, and defect management is necessary to efficiently submit and resolve new defects.

8. Create test report

- The execution logs created during the test run have to be transformed into a concise test report that informs the responsible managers about the current project status. For MBT it is important that statistics of the model coverage are included.
- 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.

9. Maintain test cases

- For each new version that has to be tested, the tests have to be adapted and repeated.
- 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.
- 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.

Examples of successful tool integration

Automated testing control in action, Octopus automated testing control system.

Let us consider a typical testing infrastructure of a software development company. Assuming that the build process is automated, it includes a build system, a file server, a versioning control system, a bug-tracking system, a test automation tool and virtual and/or physical machines. Integrating these elements of test infrastructure under a common user interface (UI) will produce a number of benefits, both in terms of testing process enhancement and cost cuttings. The purpose is not to create a universal infrastructure solution, but rather to link together existing software applications with the help of a control system that can automate interactions between them and provide a single UI for easy test configuration and management.

Today in the majority of companies applying automated testing, these systems and tools are autonomous, and the parameterization of test execution is done manually or using configuration files. When a tester receives a notification of build completion, he or she starts preparing the environment: configures virtual (or physical) machines, copies files, tests and programs from the file server to a test machine, and installs the application. Then, the tester runs a test suite, and reports detected bugs to the bug tracking system

The attempts of small and medium companies to realize a testing control system that fully automates this process are often incomplete or even fail completely as testing demands expertise and significant investments. Applied Systems Ltd., a software services provider for a leading European manufacturer of emission analysis and measurement systems, is one of the organizations demonstrating a success story.

The organization develops a complex customizable platform with hundreds of modules and agents and multiple configurations. With at least three produced builds per day, stringent quality requirements and limited testing resources, Applied Systems had an important challenge: to ensure regression testing is performed on every build. Six years of experience in automated testing and several attempts later, the company has finally implemented a closed-cycle system, named Octopus, which will run assigned tests on every produced build without tester intervention.

The first release of Octopus had interfaces for the software tools used at Applied Systems: Microsoft® Team Foundation Server 2008 as a build machine, version control system, and defect management tool, Microsoft® Hyper-V as a virtualization server; and HP QuickTest® and Microsoft® Visual Studio as test automation tools. In the following version the most popular free software was added: ThoughtWorks CruiseControl.NET (build system), CollabNet Subversion (version control), VMware Server 2.0 (virtualization server), Mantis Bug Tracker and Bugzilla (bug tracking systems), and AutoIt (automation tool). Further plans include expanding this list and supplementing it with case generation tools.

If you are ready to undertake the challenge of realizing a similar solution in your company, take into account the following key principles for designing its architecture:

- Consolidate all tools engaged in testing via application programming interfaces (APIs) in a single UI. The front-end interface will serve as a convenient proxy for flexible parameterization of test execution. As a result, the testers will be spared the trouble of adjusting configuration files, transferring them from one directory to another, and switching between different applications.
- From the very beginning, provide for a scalable and expandable architecture of the system for automated testing control. On the one hand, testing volume is likely to increase over time and you will need to add more resources like virtual machines; on the other hand, the company might upgrade to a new version of the bug-tracking system or add a new test automation tool.
- To ensure the correct environment for test run execution, enable automatic system recovery of the test lab. In the case of virtualization, apply snapshoting technology to revert your virtual machines (VMs) to a specific state of the environment. To roll back a physical machine to a clean state, utilize the VHD (virtual hard disk) technology.
- It is useful if you realize the conversion of test results from "native" format into a well-structured, uniform cumulative report in HTML format. First, it is easier to quickly view where the bottlenecks are; second, a manager or customer will be able to access the report without having to install additional software; third, the results are immediately available at any moment during test execution.
- To ensure closed cycle testing, develop algorithms that perform tasks related to testing: queue the new builds for testing, launch VMs, prepare the environment, start test scripts, report defects to the bug tracking system, and generate a test report. Automatic submission of errors is a very useful feature in data-driven testing where each incorrect result should be registered. All the events should be logged to make the testing system transparent and to facilitate debugging.

Building your own system for automated testing control is not easy, but the payoff makes it worthwhile, especially if your company works on long-term projects. The organization will save time on non-productive tasks, make effective use of limited resources and enjoy improved testing processes. In the case of Applied Systems, automatic preparation of the environment resulted in saving two hours of tester time per day via automatic bug submission – at least a minute per reported defect. The execution of a test run on four concurrent VMs permitted the probing of each build with regression tests with the same amount of physical and human resources.

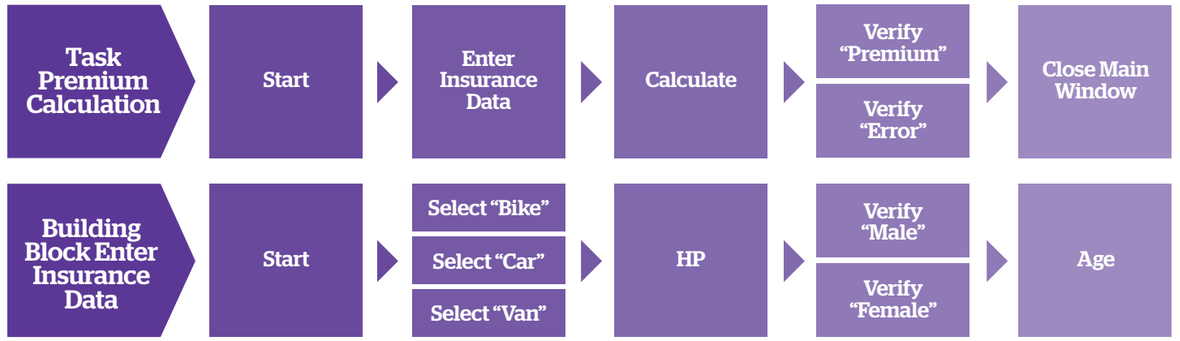

Figure 3: Task-Flow models in TEMPPO Designer

Integrating MBT into the tool chain, Atos TEMPPO Designer (IDATG)

While it is relatively easy to find tools for automated test execution, there are significantly fewer that focus on 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]).

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 (see Figure 3) 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 focuses on the necessities of testing. Test sequences are represented as *task flow diagrams* that consist of simple test steps (blue) and parameterizable 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 via 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 an interface for HP Quality Center® that allows the user to import TEMPPO Test Designer test cases via Excel. Future plans focus on an even stronger integration of TEMPPO Designer with test execution tools, especially with Ranorex® and HP QuickTest®. For instance, it will be possible to import and reuse test sequences and GUI information recorded with Ranorex® in order to accelerate model creation and increase the number of supported GUI technologies even further.

Integrating GUI test automation, Ranorex® test automation framework

The Ranorex® test automation framework is used to automate tests for web-based applications and client-based applications by simulating user actions through the graphic user interface (GUI). Ranorex® can easily be integrated into continuous integration systems as Ranorex® Studio creates an executable from a test suite project (see Figure 4).

This executable can be triggered to run by the system integrating the Ranorex® test.

It is also possible to run one specific test case, execute a predefined run configuration, and set global parameters if the test project follows the data driven approach: there are many other options available when executing the Ranorex® Test Suite using command line arguments.

To evaluate the outcome of the test, the integrating system usually examines the return value of the executable or its output text (e.g. "TEST FAILED" for failure). With Ranorex® the return value "0" signals the successful execution of the test script and a return value of "-1" signals a failure in execution. Additionally, each test run can be accompanied by an XML-based report file providing detailed information for each single step executed within a test case.

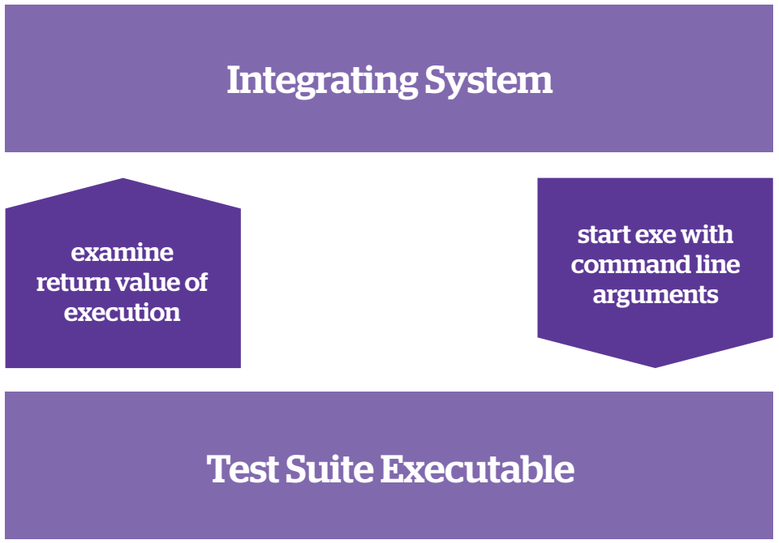

Figure 4: Integration of (generated) test cases with Ranorex® Test Automation Suite

Conclusion

In this article we discuss the impacts of an inadequate software testing infrastructure and show how model-based testing (MBT) can be efficiently integrated into the tool landscape of a test project.

MBT has gained a solid foundation in recent years. Nonetheless, in order to fulfil its primary promise of reducing costs by detecting faults at an early stage and reducing test maintenance effort, it is necessary to professionally integrate MBT with other tools in your project (e.g. for test control and execution, test management, and requirements management).

Our survey indicates that poorly integrated test infrastructure prevents professional testers from focusing their activities on more intellectual tasks such as the creation of models for generating test cases.

We summarize the key factors that have to be taken into account for integration and point out successful integration scenarios relating to three concrete tools:

- the model-based test design and generation tool TEMPPO Designer (IDATG)
- the system for automated testing control known as Octopus
- the Ranorex® GUI test automation tool.

Professional tool integration will free test engineers from burdensome manual activities, thus paving the way for maximizing the benefits from MBT in your test project.

References

[NIST 2002] The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards & Technology, Program Office Strategic Planning and Economic Analysis Group, May 2002.

[Turlo & Safronau 2011] V. Turlo and V. Safronau, Dealing with Challenges of Automating Test Execution, Proceedings of the Third IEEE International Conference on Advances in System Testing and Validation Lifecycle, October 2011.

[TE 2011] Improving the Test Process, Testing experience – The Magazine for Professional Testers, www.testingexperience.com, June 2011, ISSN 1866-5705.

[van Ewijk 2011] A. van Ewijk, TPI Next, Geschäftsbasierte Verbesserung des Testprozesses, dpunkt Verlag ISBN-10: 3898646858.

[Peischl et al. 2011] B. Peischl, R. Ramler, T. Ziebermayr, S. Mohacsi, C. Preschern, Requirements and Solutions for Tool Integration in Software Test Automation, Proceedings of the Third IEEE International Conference on Advances in System Testing and Validation Lifecycle, October 2011.

[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.

[Hüttermann 2011] M. Hütterman, Agile ALM - Lightweight tools and Agile strategies, August 2011, ISBN 9781935182634.

[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.

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

About the authors

Bernhard Peischl is the coordinator of the competence network Softnet Austria. He is responsible for managing the network's R&D activities and is in charge of a number innovation projects dealing with software testing, verification and debugging. Bernhard received a MS in telecommunications engineering (2001) and a Ph.D in computer science (2004) from the Technische Universität Graz, Austria.

Rudolf Ramler is key researcher and project manager at the Software Competence Center Hagenberg, Austria. His research interests include software testing, quality management and measurement. Rudolf works as a consultant in industry projects and is a lecturer with the Upper Austria University of Applied Sciences and the Vienna University of Technology. He studied Business Informatics and holds a M.Sc. (2001) from the Johannes Kepler University of Linz, Austria.

Vitalina Turlo is a Product Manager at Applied Systems Ltd. She holds an M.A. in International Business from Belarus State Economic University and has the international background in business analysis and research through her experience of studying and working in the United States and South Korea. Vitalina is in charge of coordinating the work of Applied Systems' automated testing competence center and crafting a product strategy for Octopus.

Stefan Mohacsi joined Siemens in 1997 and became project manager of the research project IDATG (Integrating Design and Automated Test Case Generation). Its most notable application has been as a core part of the test automation framework of the European Space Agency (ESA). 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.

Valery Safronau is the Head of Testing department at Applied Systems Ltd. He specializes in building and streamlining testing processes, improving test infrastructure, developing automated tests using different frameworks. He is the author and architect of Octopus system for automated testing control. Valery was a speaker at numerous specialized software testing events.

Tobias Walter is studying Informatics at Technical University of Graz. Besides studying he is working at Ranorex® as technical support engineer. Next to his support activities he is also writing articles published in Ranorex® blog and Ranorex® user guide.

Acknowledgements

The work presented herein has been partially carried out 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 support of the Center for Innovation and Technology (ZIT).

This article was published in Testing Experience - The Magazine for Professional Testers

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**