The UML 2.0 Testing Profile

Paul Baker[1], Zhen Ru Dai[2], Jens Grabowski[3], Øystein Haugen[4], Serge Lucio[5], Eric Samuelsson[6], Ina Schieferdecker[2,7], and Clay E. Williams[8]

[1]Motorola Labs, Basingstoke, Hampshire, RG22 4PD, England

[2]Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

[3]University of Göttingen, Institute for Informatics, Lotzestrasse 16-18, 37083 Göttingen, Germany

[4]University of Oslo, Department of Informatics, P.O. Box 1080 Blindern, 0316 Oslo, Norway

[5]IBM Rational Software, 20 Maguire Road, Lexington, MA 02421, USA

[6]Telelogic AB, Kungsgatan 6, PO Box 4128, SE-203 12 Malmö, Sweden

[7]Technical University Berlin, Faculty IV, Franklinstrasse 28/29. 10623 Berlin, Germany

[8]Center for Software Engineering, IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA

Abstract

Testing often accounts for more than 50% of the required effort during system development. However, testing is often not well integrated with other development phases. One reason for this is that designers, developers and testers all use different languages and tools, making it difficult to communicate with each other and to exchange documents. The UML 2.0 Testing Profile bridges the gap between designers and testers by providing a means to use UML for test specification and modeling. This allows the reuse of UML design documents for testing and enables test development in an early system development phase. The testing profile provides support for UML based model-driven testing. This paper presents the concepts defined in the UML 2.0 Testing Profile and explains their usage by applying those to an example of a simplified Automated Teller Machine (ATM).

Keywords

UML 2.0, UML Profile, Test Languages, Test Specification, Test Design, Grey-Box Testing, Black-Box Testing, JUnit, TTCN-3

1 Introduction

The Unified Modeling Language (UML) is a graphical language to support the design and development of complex object-oriented systems. While it is flexible in addressing the major object-oriented concepts, test specification and testing issues are beyond the scope of UML version 1.4 [10]. In late 2001, the Object Management Group (OMG) issued a Request for Proposals (RFP) to develop a testing profile for version 2.0 of UML (UML 2.0) [5,9]. A profile is a domain specific extension of UML using a standardized extensibility mechanism.

A total of seven organizations responded to the RFP for the testing profile: Ericsson, Fraunhofer FOKUS, IBM, Motorola, Rational, Softeam, and Telelogic. Early in the process, these organizations decided to collaborate and produce a joint submission rather than submitting independently. This offered several advantages:

- the resultant profile would have a broader scope,
- the best ideas would be incorporated into a single document, rather than competing with one another in rival proposals, and
- the work of developing a profile for UML 2.0 would be distributed across a larger team.

Meanwhile, the UML 2.0 Testing Profile development has come to its finalization and the profile has become an official standard of the OMG [8]. This paper describes the key capabilities that the testing profile provides and presents an example, which illustrates the use of the testing profile.

The paper is organized in the following manner: in Section 2, we present an overview of testing concepts that are represented in the testing profile. In Section 3, we provide an example that illustrates many of the concepts introduced in Section 2. Section 4 discusses the implementation of test models developed with the testing profile. The paper concludes with Section 5, which presents a summary of the profile.

2 Introducing Testing Concepts into UML

The UML 2.0 Testing Profile provides concepts that target both the pragmatic and systematic development of concise test specifications and test models for black-box and grey-box testing [1,2,8,11]. In particular, the profile introduces concepts covering: test architecture, test behavior, test data, and time. Together, these concepts define a modeling language for visualizing, specifying, analyzing, constructing, and documenting the artifacts of a test system. The philosophy we adopted for the development of these test concepts has been to make use of existing UML 2.0 concepts wherever possible, thereby minimizing the introduction of new concepts. We identified the supplementary concepts of the testing profile by analyzing existing test specification and test implementation techniques, including JUnit [7] and the Testing and Test Control Notation (TTCN-3) [3,4].

For the definition of test architectures we introduced concepts needed to specify the structural aspects of a test system including:

- The *test context*, which allows users to group test cases, to describe a corresponding test configuration and to define the test control, i.e. the required execution order of the test cases. For example, Figure 2 presents the test configuration and test control for the `ATMSuite` test context shown in Figure 1.
- The *System Under Test* (SUT), where one or more objects within a test specification can be identified as the SUT. For example, Figures 2 and 3 illustrate how the concept of an SUT enables the visualization of test configurations and the test composition.
- *Test components*, which are defined as objects within the test system that can communicate with the SUT or other components to realize the test behavior.
- The *scheduler*, which controls test execution and test components. A scheduler is responsible for the creation of test components, a synchronized start of the test behavior on the different test components, and the detection of test case termination.
- A means for evaluating test results derived by different objects within the test system in order to determine an overall verdict for a test case or test context. We refer to this evaluation process as *arbitration*. Users can either use the default arbitration scheme specified by the profile (i.e. a verdict can only get worse. Meaning that whenever a test component detects a fail, the overall verdict can be at most fail), or define their own arbitration scheme using an *arbiter*.

The test behavior specifies the actions and evaluations that are necessary to evaluate the test objective. A test objective describes the aim of a test. UML interaction diagrams, state machines, and activity diagrams can be used to define test stimuli, observations from the SUT, test control/invocations, coordination and actions. When such behaviors are specified, focus is usually given to the definition of normative or expected behaviors. The UML 2.0 Testing Profile introduces concepts for handling unexpected behaviors providing the means to define more complete, yet abstract test models. This simplifies validation and improves the readability of test models.

The handling of unexpected messages is achieved through the specification of *defaults*. The concept of default is taken from TTCN-3 in which a separate behavior is triggered if an event is observed that is not explicitly handled by the main test case behavior. The partitioning between the main test behavior and the default behavior is up to the designer. Within the testing profile, default behavior is applied to static behavioral structures. For example, defaults can be applied to combined fragments (within interactions), state machines, states, and regions. Figure 3 illustrates an example of a default application. The corresponding default definition is shown in Figure 4.

In addition to defaults, the profile introduces other concepts that are necessary for test specification, such as:

- A *test objective*, allowing the designer to express the intention of the test.
- A *test case*, which is an operation of a test context specifying how a set of cooperating components interact with the SUT to realize a test objective.
- *Verdicts*. A verdict is a predefined enumeration specifying possible test results e.g. pass, inconclusive, fail, and error.

- A *validation action*, which can be performed by the local test component to denote that the arbiter is informed of a local test result.
- A *test log*, which provides together with a *log action* a means to log entries during the test execution for further analysis.
- A *finish action* to denote the completion of the test case behavior of a component, without terminating the component.
- A *determAlt* operator (to be used in interactions) to specify the deterministic and sequential evaluation of guards of alternative behavior. The determAlt operator always selects the alternative of the first guard, which is fulfilled.

Another important aspect of test specification is the definition and encoding of test data. For this, the UML 2.0 Testing Profile supports *wildcards*, *data pools*, *data partitions*, *data selectors*, and *coding rules*. Wildcards are very useful for the handling of unexpected events, or events containing many different values. Therefore, the UML 2.0 Testing Profile introduces wildcards allowing the specification of: (1) *Any value*, denoting any value out of a set of possible values, and (2) *Any or omitted values*, denoting any value or the lack of a value (in the case where multiplicities range from 0 upwards). Data pools, data partitions, and data selectors support the repeated execution of test cases with different data values to stimulate the SUT in various ways. Data pools are associated with test contexts and may include data partitions (equivalence classes) and concrete data values. Data selectors provide different strategies for data selection and data checking. The testing profile also supports the notion of coding rules allowing the user to define the encoding and decoding of test data when communicating with the SUT.

Finally, timing concepts are provided to complete the concepts needed for precise and complete test modeling. These concepts supplement the simple time concepts defined by UML 2.0. The two new timing concepts added are:

- *Timers*, to manipulate and control test behavior as well as to ensure the termination of test cases.
- *Time zones*, to group components within a distributed system, thereby allowing the comparison of time events within the same time zone.

The concepts discussed above provide the capabilities required to construct precise test specifications using UML 2.0. The testing profile includes both structural and behavioral elements and provides key domain concepts from testing that make test specification efficient and effective.

3 The Testing Profile at Work

This section illustrates how the testing profile can be used to test the logic of an Automated Teller Machine (ATM) system. Since we want to test the logic only, the hardware and the communication with the bank will be emulated.

The SUT (i.e. the ATM) is specified in the `ATM` package (Figure 1) in terms of classes, interfaces, etc. The test related elements are defined in a separate package, which imports the `ATM` package to gain access to its public definitions. This enables access to the definitions to be tested, while prohibiting changes to them.

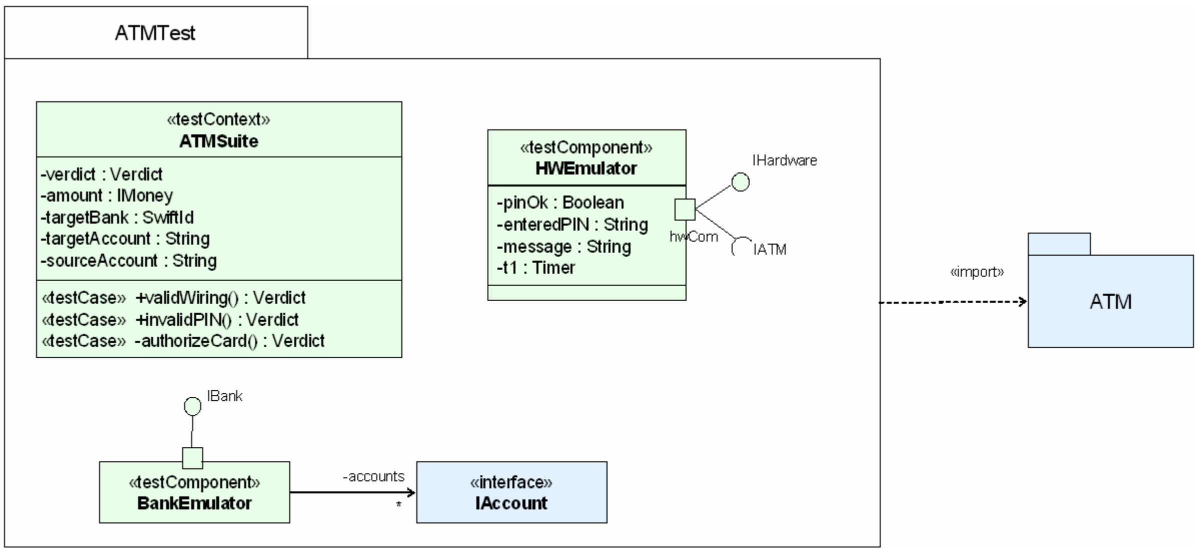

Figure 1: Package structure

The `ATMTest` package (Figure 1) contains one *test context* and two *test components*. The test context is a key concept used to specify test behavior, configuration and control. The test context contains three *test cases* specifying test behavior. The composite structure of the test context is used to specify the initial test configuration. It shows how the test components are connected to the SUT and each other.

Figure 2 illustrates the composite structure of test context `ATMSuite`. Two test components are connected with the SUT through different ports, and one of them uses an additional property that handles `CardData`. In addition, a *coding rule* is used to specify that communication between the ATM and the bank is encrypted.

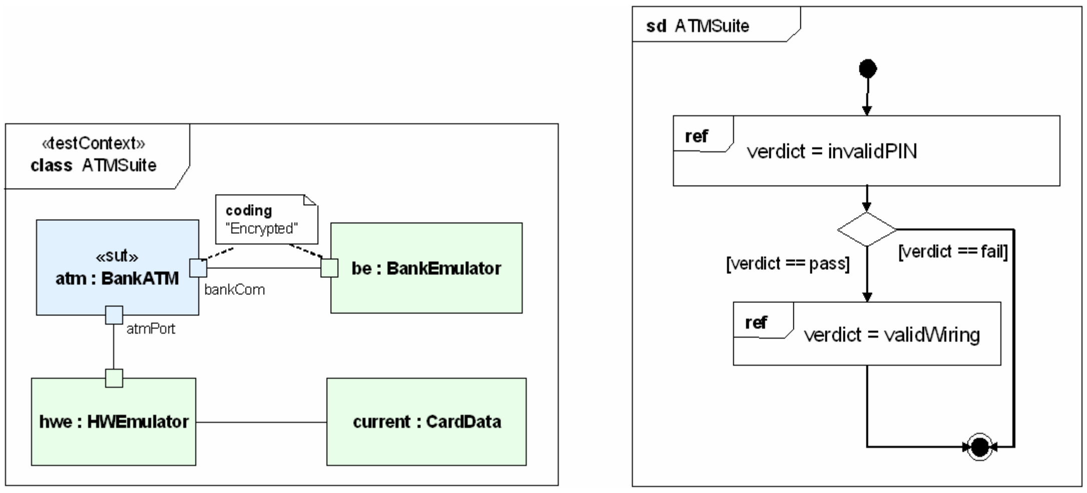

Figure 2: Test configuration and test control test context ATMSuite

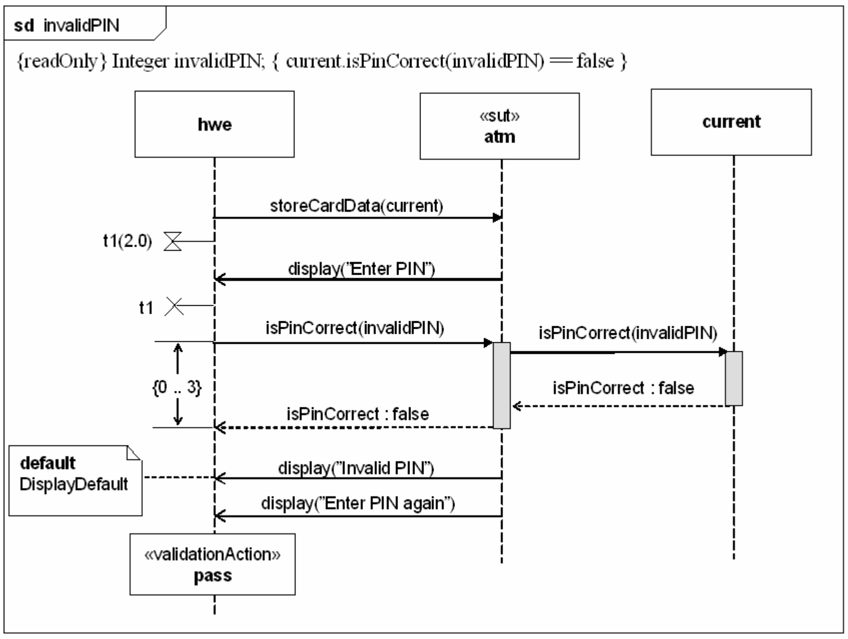

Figure 3: Test case

The behavior of the test context can be used for test control by specifying how test cases should be invoked. This is also shown in Figure 2. The `validWithdrawal` test case will only be executed if the `invalidPIN` test case has passed. An attribute of the test context is used to store the result of the tests. Test cases can also be invoked by calling them from any other class if they are publicly accessible accessible like `validWiring` and `invalidPin` are, but not `authorizeCard`.

The behavior of *test case* `invalidPIN` is depicted in Figure 3. The *test objective* of this test case is:

> *"Verify that if a valid card is inserted, and an invalid pin-code is entered, the user is prompted to re-enter the pin-code."*

This test case involves one of the test components specified in the composite structure of the test context, the hardware emulator hwe. The bank emulator is not needed since no communication with the bank is necessary in order to validate the PIN code. The `current` property is used as a utility. Figure 3 also shows the usage of the testing profile concepts *timer*, *default* and *validation action*:

- The timer `t1` is used to specify that we expect the user to be prompted to enter a pin code within 2 seconds from inserting the card. If the timer times out, the specification is violated and a default specific to the hwe will be applied (not shown here).
- The default `DisplayDefault` is applied to the reception of a display message. The default mechanism will be explained below.
- A validation action is used at the end of the test case to set the *verdict* to `pass`. This action reports the verdict to the *arbiter*, which calculates the overall verdict of the test case. At the end of the test case, the arbiter implicitly returns the overall verdict.

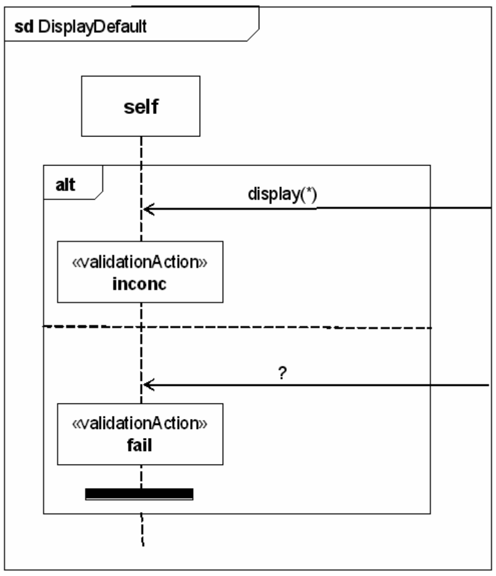

Figure 4: Default definition

The test case specification in Figure 3 depicts the normal or expected behavior only. If unexpected events, or expected events with incorrect value specifications occur during the test execution then a default will be invoked.

Figure 4 illustrates the specification of the default `DisplayDefault`. It is applied to the reception of the `display("Invalid PIN")` in the `invalidPIN` test case (figure 3). The default will be invoked if the display message is received with a different parameter value or if another message is received at this point in the test. Default `DisplayDefault` shows the use of two further testing profile concepts: *wildcards* and *finish action*. If the display message is received with any other parameter value than `"Invalid PIN"` (see figure 3) or without any value (indicated by the wildcard '`*`'), the result is inconclusive and the test case execution continues in the test case behavior. If any other message (indicated by the wildcard '?') is received, the test fails and this test component finishes the execution of the test case.

Even though the ATM example in this section only provides test behavior in form of sequence diagrams, the testing profile is not restricted to sequence diagrams. All UML behavior diagrams can be used for test specification. In addition, the testing profile definition [8] also provides examples of test specifications for different kinds of testing, including component, integration, system and load testing.

4 Test Implementation

The UML 2.0 Testing Profile provides two mappings towards test execution environments:

(1) JUnit [7] is an open source unit testing framework, which is widely used by developers who implement unit tests in Java. We provide a mapping from the testing profile to the JUnit framework. There are instances where no trivial mapping exists to the JUnit framework. In these cases, existing JUnit extensions such as for repeated test case runs or for active test cases can be used.

(2) TTCN-3 [3,4] is widely accepted as a standard for test system development in the telecommunication and data communication area. TTCN-3 is a test specification and implementation language to define test procedures for black-box testing of distributed systems. Although TTCN-3 was one basis for the development of the testing profile, they differ in some aspects. However, all UML 2.0 Testing Profile specifications (with small exceptions) can be represented by TTCN-3 modules and executed on TTCN-3 test platforms.

More information about these mappings can be obtained from the testing profile definition [8]. With these mappings, the direct generation of executable test cases from UML 2.0 Testing Profile specifications is enabled. Tool environments for UML-based design and development processes can use the mappings to integrate with existing test execution, management and result analysis tools.

5 Summary

In this article we have provided a brief tour of the testing profile for UML 2.0. The testing profile was developed to provide test design and modeling capabilities that are consistent with UML 2.0. Wherever it was feasible, we reused the existing machinery of UML, adding additional concepts and capabilities only where they were required.

The profile defines testing concepts, including test context, test case, test component, and verdicts that are commonly used during testing. Behavioral elements from UML 2.0 can be used to specify the dynamic nature of test cases. These include interactions, state diagrams, and activity diagrams. Additional concepts for behavior include several types of actions (validation actions, log actions, final actions, etc.), defaults for managing unexpected behaviors, and arbiters for determining the final verdict for a test case. The definition and handling of test data is supported by wildcards, data pools, data partitions, data selectors and coding rules. Timers and time zones are also provided to enable specifications of test cases with appropriate timing capabilities.

We presented a small example that served to illustrate the key concepts of the testing profile. This example shows the test specification for ATM system by presenting test components, the test configuration, and selected test behavior. We briefly discussed the mapping capabilities of the profile for both JUnit and TTCN-3. The profile is designed so that additional mappings to other test technologies could be defined in an easy and concise manner.

The profile also contains a standalone metamodel as a separate compliance point. This allows non-UML tools to provide an implementation that is consistent with the UML based profile. The Eclipse Hyades project currently has implemented this standalone model as a basis for their test information model [6].

The UML 2.0 Testing Profile provides a coherent set of extensions to UML 2.0 that support effective test specification and modeling for grey-box and black-box testing. This is a significant enhancement to UML to support the testing portion of the system development lifecycle.

6 Bibliography

[1] B. Beizer. *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., 1995.

[2] Z.R. Dai, J. Grabowski, H. Neukirchen, H. Pals. *From Design to Test – Applied to a Roaming Algorithm for Bluetooth Devices.* Testing of Communicating Systems (Editors: R. Groz, R.M. Hierons). Proc. of the 16th IFIP Intern. Conf. on Testing of Communicating Systems (TestCom2004), LNCS 2978, Springer, March 2004, pp. 33-49

[3] ETSI European Standard (ES) 201 873-1 version 2.2.1 (2003-02): *The Testing and Test Control Notation version 3 (TTCN-3); Part 1: TTCN-3 Core Language*. Also published as ITU-T Recommendation Z.140.

[4] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, C. Willcock. *An Introduction into the Testing and Test Control Notation (TTCN-3).* Computer Networks, Volume 42, Issue 3, Elsevier, June 2003.

[5] Ø. Haugen, B. Møller-Pedersen, T. Weigert. *Structural Modeling with UML 2.0*. UML for Real (Editors: L. Lavagno, G. Martin, B. Selic). Kluwer Academic Publishers, 2003, pp. 53-76.

[6] Hyades: http://www.eclipse.org/hyades

[7] JUnit: http://www.junit.org.

[8] OMG ptc/04-04-02: *UML 2.0 Testing Profile*, Finalized Specification.

[9] OMG ptc/04-05-02: UML 2.0 Superstructure Specification.

[10] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. ISBN 020130998X, Addison-Wesley, 1998.

[11] I. Schieferdecker, Z.R. Dai, J. Grabowski, A. Rennoch. *The UML 2.0 Testing Profile and its Relation to TTCN-3*. Testing of Communicating Systems (Editors: D. Hogrefe, A. Wiles). Proc. of the 15th IFIP Intern. Conf. on Testing of Communicating Systems (TestCom2003), LNCS 2644, Springer, May 2003, pp. 79-94.

7 Authors' biographies

Paul Baker

Paul Baker has been working in the field of software engineering for 22 years and is currently the manager of Motorola’s European System and Software Engineering Research laboratory. He completed his MSc in Software Engineering at the University of Oxford during which he studied formal semantics for scenario-based specification languages and automatic test generation algorithms. Since working at Motorola, Paul has been involved in model-driven development techniques, such as automatic code and test generation and more recently advanced techniques for system and software validation and verification. To support this work he has been involved in the standardisation of Message Sequence Charts (ITU Z.120), the graphical format for TTCN-3, and the OMG’s UML 2.0 Test Profile. Paul has a number of patents in the area of test generation, and the analysis of partial specifications.

Zhen Ru Dai

Zhen Ru Dai studied computer science at the University of Lübeck in Germany. At time, she is working at Fraunhofer Fokus in Berlin for her PhD. The main topic of her PhD is the model-driven test development with UML 2.0 and TTCN-3. Additionally, she is also working on further European projects considering testing and test methodologies. During the past few years, Zhen Ru contributed to the standardization of the graphical format for TTCN-3 and was an active member of the UML 2.0 Testing Profile consortium at OMG.

Jens Grabowski

Prof. Jens Grabowski works at Institute for Informatics at the Georg-August University of Göttingen in Germany, where he is head of a research group on software-engineering for distributed systems. His research interests include automatic test generation, test specification, test languages and test methodology. Jens Grabowski is an active member in the standardization of TTCN-3 by ETSI and of the UML 2.0 Testing Profile by OMG.

Øystein Haugen

Prof. Dr. Øystein Haugen is an associate professor at University of Oslo focusing on modeling languages and object orientation as well as formal description techniques. He has worked many years in industry where he has focused on introducing better methodology in engineering. He now teaches the course “Unassailable IT-systems”. He has earlier been responsible for the language on Message Sequence Charts (ITU, 2000) and the Interactions chapter of UML 2.0 (OMG, 2004).

Serge Lucio

Serge Lucio is a Sr. Product Manager at IBM Rational, responsible for Automated Software Quality products' strategy. For the past 3 years, he has been instrumental in the creation of software quality standards in the industry, co-leading the Eclipse-Hyades project and actively contributing to the UML (Unified Modeling Language) 2.0 at the Object Management Group. Prior to that he worked as a consultant, helping organizations improve their test and verification practices in the automotive, aerospace and telecom industries.

Eric Samuelsson

Eric Samuelsson is working at Telelogic's TAU G2 product development lab in Malmö, Sweden, as a project manager and language specialist. He is mainly working on techonologies for model-driven development and automatic code generation using UML 2. Eric is an active member in the standardization of the UML 2.0 Testing Profile at OMG.

Ina Schieferdecker

Prof. Dr.-Ing. Ina Schieferdecker is heading the Competence Center for Testing, Interoperability and Performance at the Fraunhofer Institute on Open Communication Systems (FOKUS), Berlin and Professor on Engineering and Testing of Telecommunication Systems at Technical University Berlin. Mrs. Schieferdecker works since 1994 in the area of design, analysis, testing and evaluation of communication systems using specification-based techniques like UML (Unified Modelling Language), MSC (Message Sequence Charts) and TTCN-3 (Testing and Test Control Notation). She is an active member in the standardization of TTCN-3 by ETSI and of the UML 2.0 Testing Profile by OMG. Mrs. Schieferdecker authored many scientific publications in the area of development and testing. She is co-founder of the Testing Technologies IST GmbH, Berlin.

Clay E. William

Clay Williams is a Research Staff Member and Manager of the Software Testing and Quality Research Group at the IBM T.J. Watson Research Center in Hawthorne, NY. He holds a Ph.D. in computer science from Texas A&M University. His interests include software engineering, model-driven development, and software validation and verification. He is a member of the ACM and IEEE.