A measurement framework for evaluating model-based test generation tools

A. Sinha
C. E. Williams
P. Santhanam

This paper presents a measurement framework for evaluating model-based test generation (MBTG) tools. The proposed framework is derived by using the Goal Question Metric methodology, which helps formulate the metrics of interest: complexity, ease of learning, effectiveness, efficiency, and scalability. We demonstrate the steps involved in evaluating MBTG tools by describing a case study designed for this purpose. This case study involves the use of four MBTG tools that differ in their modeling techniques, test specification techniques, and test generation algorithms.

INTRODUCTION

Testing ensures that software meets its requirements and is thus a vital part of the software development life cycle. Because testing—be it integration, system, or acceptance testing—occurs late in the software development life cycle and because it is time consuming, the testing effort is often shortened in order to compensate for schedule slippages during earlier development activities. This results in insufficient testing of products before their release. Model-Based Test Generation (MBTG) has recently emerged as a possible approach to alleviating this problem by improving the effectiveness of the testing effort.

According to the consensus prevalent at the Workshop on Advances in Model-Based Software Testing (A-MOST 2005),[1] in a model-based testing technique the behavior of the application under test, as specified by the user, is used exclusively for generating a suite of tests for validating the application. The level of granularity of the input information provided by the user can vary widely. With this broad definition in mind, a number of studies on model-based testing are found in the recent literature.

Dalal et al.[2] highlight the advantages and challenges associated with the use of the Automatic Efficient Test Generator (AETG) technique on four industrial products. Pretschner et al. present a case study involving the use of AutoFocus[3] for testing an "infotainment" network. This case study highlights the model coverage, the implementation coverage, and the error-detection capabilities of AutoFocus. Veanes et al.[4] report on a case study that evaluates

the effectiveness of the tool Asml/Spec# in an online testing setup. Another related study by Paradkar[5] presents a case study that seeks to identify fault models for better fault detection effectiveness using MBTG.

Although these studies provide insights into specific MBTG techniques, to our knowledge there are no publications that propose a measurement framework for evaluating MBTG tools. Consulting companies, such as Grove Consultants[6] or Ovum Inc.,[7] often publish reports analyzing the capabilities of specific tools. Whereas these reports may serve well for a particular MBTG tool, they lack a framework for comparing tools.

■ In a model-based testing technique, the behavior of the application under test, as specified by the user, is used to generate a suite of tests ■

When managers and other decision makers in the information technology (IT) industry need to select an MBTG tool for their software development project, they are often at a loss, primarily because there are no measurement frameworks to compare the capabilities of various MBTG tools. In this paper, we provide one such measurement framework that is based on the Goal Question Metric (GQM) methodology.[8] Applying GQM in MBTG helps formulate the metrics of interest: complexity, ease of learning, effectiveness, efficiency, and scalability. We demonstrate how to use our measurement framework by describing a rudimentary case study for comparing four MBTG tools. Although this case study is not sufficiently extensive for reliable comparison data, its purpose is to describe in detail the steps involved in comparing MBTG tools. The four tools chosen for this study (Archetest version 0.5,[9] ASMLT version 2.0,[10] TestMaster release 1.9.2,[11–13] and HOTTest release 0.1[14]) differ in their modeling techniques, test-case specification techniques, and test generation algorithms.

The rest of the paper is organized as follows. In the next section, "Design of the measurement framework," we briefly describe the GQM approach, and we show how GQM-driven questions lead to the set of metrics of interest: complexity, ease of learning, effectiveness, efficiency, and scalability. In the section "Case study," we describe the four MBTG tools used, the three target applications that are tested, and the process we follow, and we discuss possible pitfalls when analyzing measurement data. In the section "Case study measurements," we describe and discuss the measurement data. The last section contains a summary and final comments.

DESIGN OF THE MEASUREMENT FRAMEWORK

The measurement framework was designed following the GQM methodology.[8] GQM defines a measurement model on three levels:

1. *Conceptual (goal)*—This defines the purpose of the study. The goal is to analyze some object of study (e.g., process, product) toward a specific purpose (e.g., characterize, evaluate, predict, motivate, improve) with respect to a focus (e.g., effectiveness) on behalf of (from the point of view of) a stakeholder (e.g., customer, organization).
2. *Operational (question)*—Questions are formulated to identify the information that is needed to achieve the goal. The set of questions is used to define models of the object of study and characterize the way a specific goal is achieved.
3. *Quantitative (metric)*—This consists of sets of metrics, where each set of metrics is based on the models and is associated with a question that has to be answered in a measurable way.

The purpose of the our framework is to compare selected MBTG tools for test generation abilities. We define the goal in accordance with the template prescribed by the GQM methodology as follows: "*Analyze MBTG test generation tools to characterize them with respect to their test generation ability from the point of view of testers of IT systems*."

In using our measurement framework for evaluating a set of MBTG tools, we do not consider specific requirements that a project may have, as we want the framework to be universally applicable, but the framework can be easily adapted, following the GQM methodology, to suit individual needs of projects.

From questions to metrics

At the operational level, we ask the following questions:

1. How *complex* is the tool to use?
2. How easy is it to *learn* the tool?
3. How *effective* is the tool?
4. How much *effort* is needed to test applications?
5. How does the test generation effort *scale* with application size?

It is understood that the higher the *complexity* of using the tool, the lower its *usability*. A higher complexity also means a higher investment in human resources due to the higher skill level required. *Ease of learning* is another vital issue that governs applicability. Lower ease of learning implies an increased effort in educating the testers on the test generation technique. Complexity and ease of learning determine directly the adoptability of the technique.

Questions 3 and 4 determine the return-on-investment from using the test generation technique. The *effort* needed for testing and the *effectiveness* of the test technique jointly determine the productivity of the tester. The gain in effectiveness should not be at the cost of a heavy increase in effort.

The staff responsible for selecting a tool needs to ascertain that the testing technique *scales* to large applications. In other words, they need to make sure that when the size of the application grows, the effort required stays manageable.

Metrics and measurement models

At the quantitative level, GQM deals with metrics. These metrics help answer the questions asked at the operational level. The metrics that are part of this study follow:

1. *Complexity*—The complexity of the test generation process is associated with the difficulty in using the tool due to the number of concepts that must be learned in order to produce a correct test model. A concept may be an operator, a function, or a modeling mechanism that must be learned in order to use the tool. The complexity measure *CPLX* is determined by using a graph that depicts the dependencies between concepts. Such graphs, which we refer to as *semantic dependency graphs*, relate the various concepts, which are represented as nodes. The nodes of a semantic graph can represent any of the following concepts:

    a. *Modeling concepts*—These are concepts related to test models; for example, for a model based on the Unified Modeling Language** (UML**), these can be use case diagrams, class diagrams, and activity diagrams.
    b. *Linguistic concepts*—These are concepts related to the specification language. These include language-related constructs and special functions and operators that must be learned in order to use the tool; for example, in ASML, the language constructs supporting abstraction mechanisms are linguistic concepts.
    c. *Technique concepts*—These relate to the test generation technique; for example, in TestMaster, constraint-related concepts that define the context of any state are technique concepts.

    The complexity of a concept is assigned one of three values: 1 (easy), 2 (moderate), or 3 (difficult). The difficulty is viewed from the perspective of the user of the tool. A concept is assigned a complexity value of 3 if it is likely that the user has no prior experience with the concept. It is assigned a complexity value of 2 if it is likely that the user has some experience with the concept or with a related concept. If it is likely that the user has working experience with the concept or with a related concept, then the concept is assigned a complexity value of 1.

    Complexity of any node in a semantic dependency graph is calculated as follows:

    $$CPLX_{node} = \sum_{\text{all child nodes}} CPLX_i + CPLX_{concept}\,. \qquad (1)$$

    This metric represents the complexity of the concepts involved in the use of each of the tools and is not necessarily equivalent to ease of learning. Although ease of learning depends on conceptual complexity, it also depends on the quality and amount of support available for learning the tool.

2. *Ease of learning*—Ease of learning is defined as the time a user needs to achieve a specified proficiency level with the tool. Measuring proficiency in using any one concept involves counting the number of correct uses of that concept with two different applications: a "learning application" (used only for learning) and the target application (the one used for testing). Thus, the proficiency in using concept i is

calculated as

$$prof_i = \frac{n_i}{N_i}, \tag{2}$$

where n_i is the total number of correct uses of the concept i by the user with the learning application, and N_i is the total number of times the concept is used in the target application. The proficiency with a tool is calculated as

$$PROF = \frac{\sum_{i=1}^{m} prof_i}{m}, \tag{3}$$

where m is the total number of concepts used. Ease of learning is calculated as

$$EASE = \frac{PROF}{t_{Learn}}. \tag{4}$$

The learning time t_{Learn} is the time needed for the user to achieve a given proficiency. Clearly, ease of learning contributes positively toward usability.

3. *Effectiveness*—The effectiveness of a test technique is determined by its ability to uncover faults. A fault can be viewed as the failure of the application to satisfy a specified requirement. Thus, the number of faults is equal to the number of requirements that the application fails to satisfy. Furthermore, effectiveness is calculated as the fraction of requirements tested by the test suite generated using the technique under study.[15] Thus, effectiveness for a test suite is defined as:

$$Effectiveness = (Number\ of\ Requirements\ Covered) / (Total\ Number\ of\ Requirements\ in\ the\ Application) = \frac{r}{R}. \tag{5}$$

Both r and R refer to atomic requirements. An atomic requirement is a requirement that describes a single nondecomposable feature of the system. Atomic requirements do not subsume, nor are they composed of other requirements. For example, the requirement "On clicking the PLOT button, DAT will generate the query results in chart form" is implemented with the atomic requirements "On clicking the PLOT button, DAT will generate a graph"; "On clicking the PLOT button, DAT will open a window displaying the generated graph"; and "On clicking the PLOT button, DAT will switch the active window."

4. *Efficiency*—Efficiency is a measure of the ease of testing a system after a user's learning is complete. Efficiency is calculated as

$$EFF = \frac{Size\ of\ the\ Testing\ Assignment}{T_{MM}}, \tag{6}$$

where T_{MM} is the net effort in the testing process. The size of the testing assignment can be measured as follows:

$$Size\ of\ the\ Testing\ Assignment = Application\ Size * C_{Coverage} * C_{Complexity}, \tag{7}$$

where *Application Size* is the size of the system under test in lines of code or function points[16]; $C_{Coverage}$ is the coverage coefficient of the test model as given in Equation 5; and $C_{Complexity}$ is the complexity of the application. The complexity of the application can be measured by using various metrics identified in the literature. In this paper, however, we compute relative measures of complexity in which this factor cancels out.

The effort in testing T_{MM} is given by

$$T_{MM} = T_M + T_G + T_S + T_X, \tag{8}$$

where T_M is the time for test modeling, T_G is the time for test generation, T_S is the time for setting up the execution environment, and T_X is the time for test execution. Efficiency contributes positively towards the usability of the tool. A high efficiency enhances the usability of the test tool.

5. *Scalability*—Scalability is defined as the property of the tool that keeps the testing effort manageable when the size of the application increases. It can be measured by the increase in application size per unit increase in effort; therefore, we define scalability as:

$$S_i = \frac{Increase\ in\ Application\ Size}{\Delta T_i}. \tag{9}$$

ΔT_i is the increment in one of the components T_{MM}, T_G, T_M, T_S, or T_X (see Equation 8). Correspondingly, S_{MM} can be scalability with regard to net effort, S_G can be scalability with regard to test generation effort, S_M can be scalability with regard to modeling effort, and so on. Thus, scalability of the tool with regard to effort in modeling is defined as

$$S_M = \frac{Increase\ in\ Application\ Size}{\Delta T_M}. \tag{10}$$

CASE STUDY

In this section we describe a case study in which the framework was used to evaluate four tools: Arche-

test, ASML, HOTTest, and TestMaster. The goal of the case study was to demonstrate how one evaluates a number of MBTG tools using the framework. Because the study was not based on a sufficiently large sample, the tool comparison data can be viewed only as tentative and in need of additional experimentation.

The four MBTG tools

Figure 1 shows the test generation process. As shown in the figure, a natural language specification of the requirement for the application under test is the needed input to the application-modeling process, whose outcome is the application model. In the test-modeling phase that follows, the application model is augmented with various testing objectives provided by the test specification in order to derive a test model. This model is then used for test generation, the result of which is a test suite that can be run on the executable code with results that can be verified against application specifications.

The application model in Archetest consists of UML-based high-level use cases and domain models. In contrast, ASMLT uses application models specified by using abstract state machine language (ASML). TestMaster uses extended finite state machines (EFSMs) to specify application models, whereas HOTTest uses application models specified by using a strongly typed domain-specific language (DSL), such as HaskellDB.[17] When discussing the evaluation of the four tools, we sometimes refer to them as the four modeling techniques.

In Archetest, use cases are formalized by using a test case specification that adds five key concepts to standard use cases: preconditions, parameters, test data, results and the execution template. *Preconditions* state what must be true in the system for the use case to be eligible for execution. *Parameters* represent input to the use case from an actor. Parameters may have partitions associated with them. *Partitions* are logical values that testers typically use to think about test data. *Test data* associates physical values with partitions that can be used in actual testing of the system. *Results* are the named outcomes of executing the use case. They are guarded by constraints that indicate the conditions under which they occur, and also have associated update statements that change the state of the system. Finally, the *execution template* defines how the use case is realized in terms of the APIs of

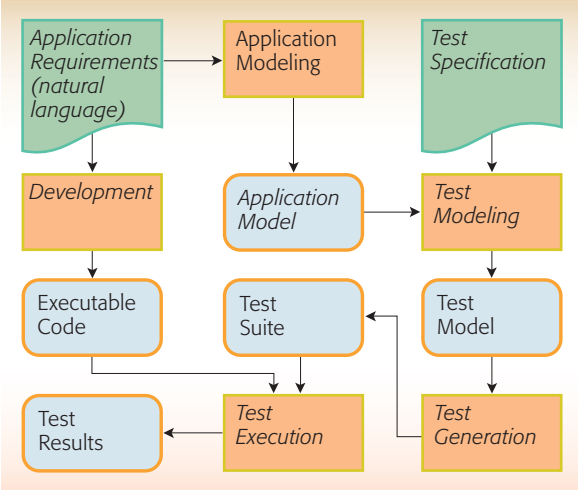

Figure 1
The test generation process

the implemented system. To each use case in an Archetest test model, fault modeling techniques are first applied to determine interesting test variations. Integer programming techniques are used to determine efficient ways to flow through the use cases as test cases are generated. Finally, the test cases are generated by using a series of graph traversal techniques.

In ASMLT, the test specification consists of an abstraction property that guides the tool to generate a finite state machine (FSM) from an ASML model. The tool triggers all possible transitions to determine the available system states. Based on the abstraction property, these states are abstracted into hyperstates. The hyperstates are finite in number and are used to form the states of the FSM. Then ASMLT generates test cases by using a "Chinese Postman" tour (the shortest closed path that goes through all edges of a directed graph) to traverse the states of the FSM. The traversal of the FSM produces paths that represent a sequence of method calls.

For TestMaster, the test is specified by providing context to transitions in the EFSM model. A transition in an EFSM model has fields that can specify events, state updates, enabling conditions, and likelihood. A transition is triggered by an event, provided that the enabling condition is satisfied. Test case generation thus involves identifying input sequences that enable the transitions and guide the system through a path defined by EFSM states. Each

input sequence so constructed signifies a test case for the system under test. TestMaster can generate test cases following user-specified path coverage schemes, such as Full Cover, Transition Cover, and Profile Cover. It uses a combination of basic graph-coverage algorithms like depth-first search, breadth-first search, and minimum spanning tree to derive the paths through EFSMs.

HOTTest automatically identifies and extracts possible functional flows in the DSL specification and then extracts variables from functions and associates type and value to each of them. Following a translation scheme, a behavioral model based on an EFSM representation of the system is created. The type safeness of DSL functions allows derivation of useful domain-specific system axioms. HOTTest identifies the DSL operators in the model and, based on the associated axioms, embeds new states in the EFSM-based test model. The modified EFSM generates test cases following a technique similar to that of TestMaster. The generated test cases check if the application satisfies the axiomatic property. Because each domain-specific axiom is related to a certain domain-specific requirement, the test cases automatically check for certain domain-specific requirements.

Case study instruments

The case study involves a human subject, a summer intern working in software testing at the IBM Thomas J. Watson Research Center. The intern had previous formal training in software testing and practical experience comparable to that of an average tester in a software firm (2–3 years). The subject was assigned the task of generating tests for three applications: DAT (Data Analysis Tool), SSP, and SearchPUBS. SSP was used as a learning application, that is, the target application for the learning phase. DAT and SearchPUBS were the target applications for the case study; DAT was the target application for all metrics; SearchPUBS was used together with DAT for evaluating scalability.

DAT is an application that analyzes data on software defects collected during software development and maintenance. The tool, which is an essential tool for IT organizations, is based on the Orthogonal Defect Classification (ODC[18]) methodology for capturing defect information. It is implemented in Java**, has a graphical user interface, and generates and executes Structured Query Language (SQL) queries for a database that contains defect reports collected during various phases of the software-development life cycle. The data set that results from the queries is analyzed for possible leads on issues that face most IT organizations, such as product stability, test effectiveness, and customer usage.

SSP is a small application for generating queries for PUBS, a Microsoft Access** database that contains information on authors and their publications. The information is stored in three tables: AUTHORS, TITLES, and TITLEAUTHOR. SSP, which generates and executes queries on PUBS, prompts the user for search options. The user may search by first or last name of the author, by location, or by book title. The application returns either the name of the author and the corresponding publication or the message "No such entry." SSP was modeled by the subject while learning the tool. It was chosen because it is a relational-database-based application, similar to DAT.

SearchPUBS, which is also based on PUBS, is a Visual C++** application (about 4200 lines of code) that queries the database for author information. The user can form queries by using a dialog-based system that provides 12 criteria for searching.

The following are some artifacts that were part of the case study:

1. *DAT Product Description (A1)*—A1 is a set of documents that describe the features of DAT and also contain screen captures and directives for users.
2. *DAT Requirements Documents (A2)*—A2, the sole basis for creating the test models, is a natural language specification of the functional and nonfunctional requirements of DAT, specified in accordance to the IEEE format.[19] A2, which is derived from A1, was created by the subject under the supervision of an ODC expert.
3. *Parsed Requirement List (A3)*—A parsed requirement list contains atomic requirements (as previously defined). A3 is derived from A2 by an ODC expert. Each requirement in A3 is classified by an expert as either domain specific or generic. For example, if a requirement is specific to the domain of relational database applications, then it is classified as domain specific.

Case study: process

The objective of the case study is to have the subject learn to use the four tools and generate tests for DAT

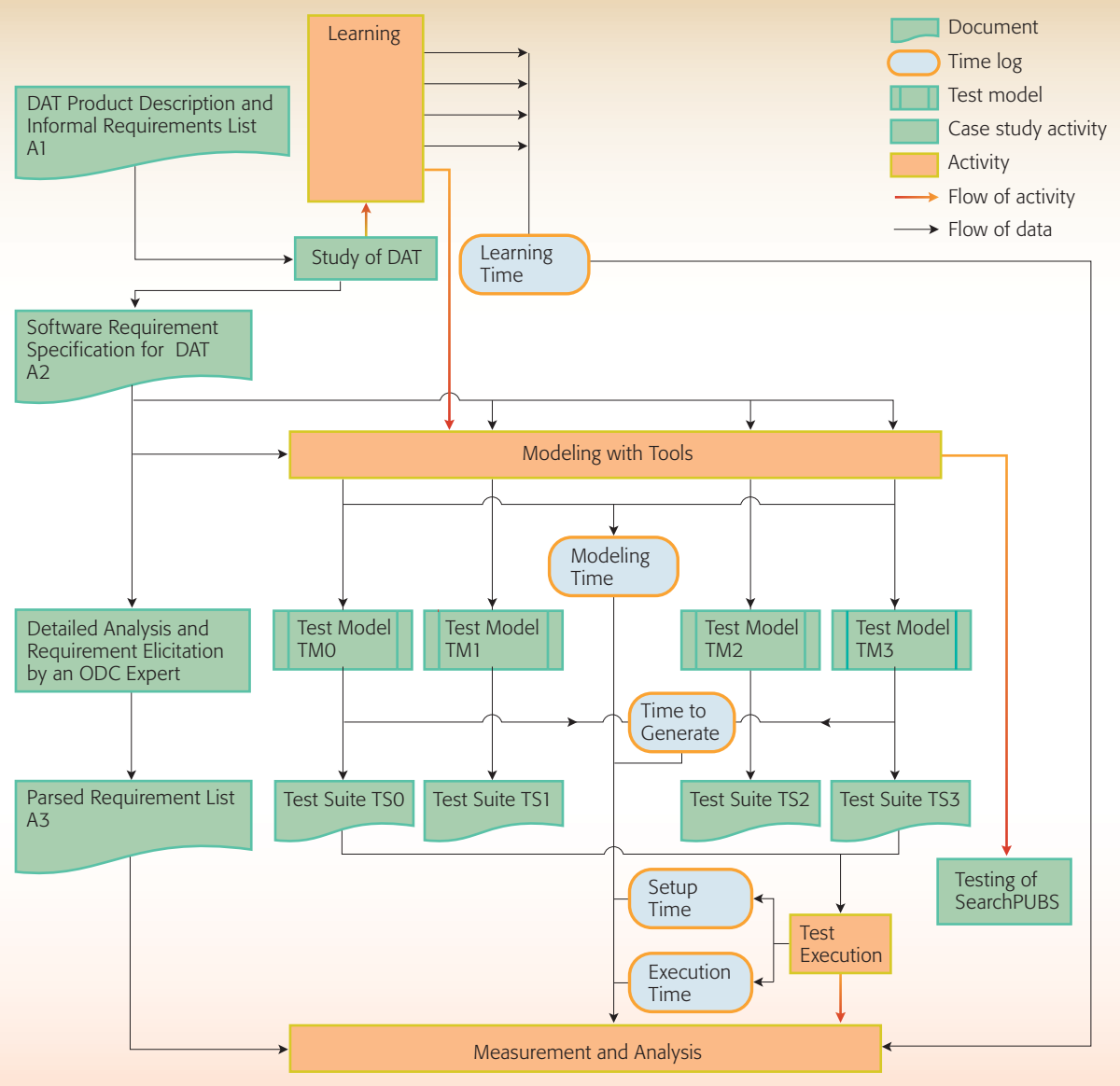

Figure 2
The case study process

by using these tools. We describe here the steps involved. ***Figure 2*** shows the process for carrying out the case study.

First, the subject receives training in ODC and then develops the natural language specification A2 for DAT from A1.

Next, the subject learns to use the four tools in an arbitrarily selected sequence. For practice, during the learning process the subject uses SSP. The subject reads the technical manuals and the user support documents available for each tool. A log is maintained in order to record the learning time and also to document any pertinent observations during the learning phase. After perusal of the documents, the subject uses each tool to generate tests for SSP. The results of the test generation step are used to evaluate the proficiency achieved in each tool by the subject. The proficiency is assessed by an expert tester who is not involved in the case study.

Modeling of DAT immediately follows the learning phase. As in the learning phase, a log is maintained

for recording the time and any observations of interest made by the subject during the modeling process.

The scalability of the tool is evaluated next, when the subject models the target application, SearchPUBS.

Test cases are derived independently for DAT and SearchPUBS by using each test generation tool. The test suite and results of the test execution are recorded and analyzed. The test suites are assessed against the independently prepared parsed requirements list (A3). The execution time and the test generation times for each model are recorded. The test results are also analyzed.

■ The proposed framework is derived by using the Goal Question Metric methodology, which helps formulate the metrics of interest ■

The final step in the case study consists of measurement and analysis. The data recorded during the test modeling phase, the test generation phase, and the test execution phase are compiled for the four techniques and are analyzed.

Threats to validity

Some factors that affect the validity of the results of any experimental study can be viewed as threats to validity. They can be classified as either internal or external threats. Internal threats to validity refer specifically to threats that determine whether an experimental treatment or condition makes a difference and whether there is sufficient evidence to support the claim.[20] Internal threats to validity are identified in Reference 20 as the following:

- History, which refers to the specific events that occur between any two sets of measurement and that may affect the outcome
- Maturation, which indicates the processes affecting the subjects that are functions of time (e.g., hunger, aging)
- Testing, which refers to the effect that taking a test before the experiment has on the outcome of the experiment
- Experimental mortality, which refers to the loss of subjects during the experiment
- Instrumentation, which refers to the changes in instruments, observers, or scorers that may affect outcomes

Because the case study was not conducted in a controlled environment, there was no threat due to history or maturation. The effect of testing was reduced by isolating the study of the system from the study of the test design tools. The order in which the tools were studied might cause some concern. However, because proficiency is evaluated on the correct usage of the concepts and not on the correctness of the model itself, the effect of this factor is null. Further, the test of proficiency for each tool before modeling of DAT ascertained that the measurement results were not biased because of the continuous learning of the subject. The analysis and measurements were performed after completion of the modeling process; this was to ensure there was no threat due to instrumentation on the results.

The subject was asked to develop a natural language specification of the requirements of the application to be tested before development of the test model. This ensured that the subject's rate of learning had leveled off, that is, had reached an asymptotic level, before the application modeling phase. An experiment involving a single subject ensures minimal variation due to personal bias and abilities. Although experimental mortality was a major threat to validity, the entire duration of the experiment was about four months, which was manageably small. Further, the subject was naturally motivated because the study was part of his summer project requirements.

External validity means the results of the experiment or case study can be applied directly to other similar scenarios. The case study presented here has only one subject performing the test generation activities. This limits the applicability of the results to other cases. Because the subject (as described in the section "Case study instruments") is a member of the representative population, the results may indicate possible trends. For a wider applicability of the results, the experiment needs to be repeated with a large number of subjects. This constitutes an external threat to the validity of our results.

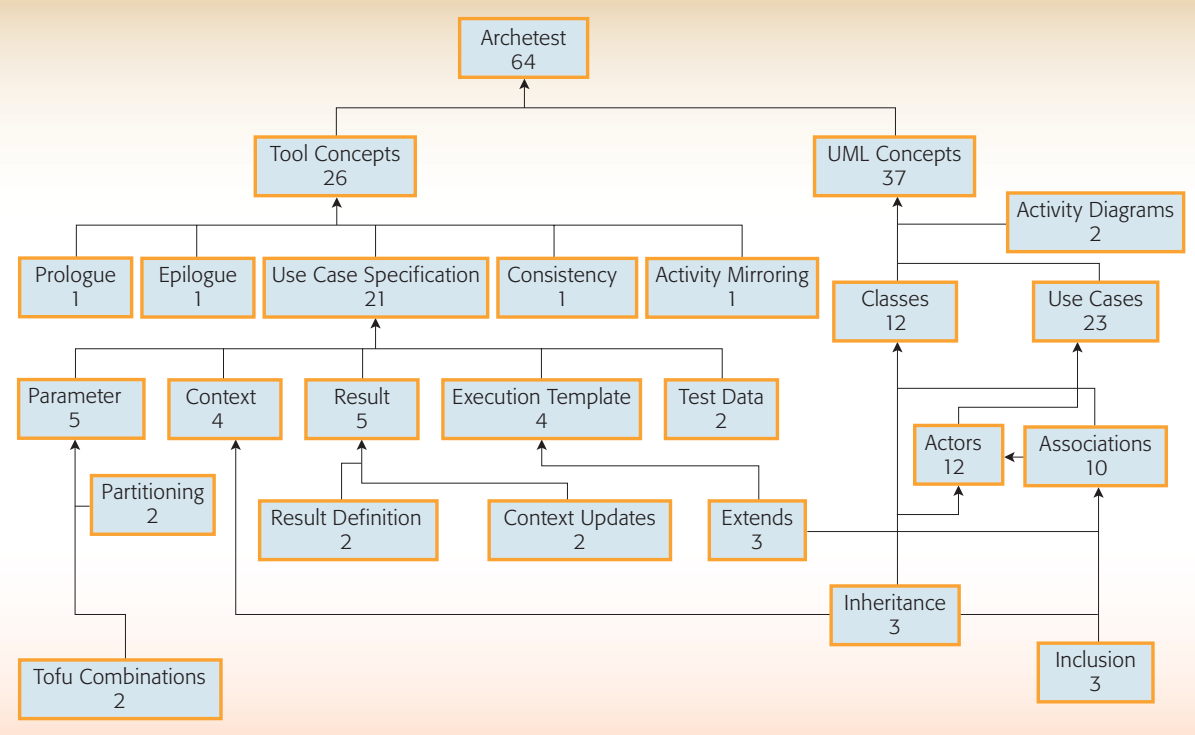

Figure 3
Semantic dependency graph for Archetest (novice perspective)

CASE STUDY MEASUREMENTS

In this section we describe the way readings are taken for each metric identified in the section "Metrics and measurement models" and present the numerical results obtained.

Complexity

Complexity measures can be calculated from the perspective of either a *novice* or *expert*. A novice is not familiar with any tool-related concepts but is familiar with basic principles of test generation. An expert is well-versed in the modeling principles underlying the four tools (ASML for ASMLT, UML for Archetest, Haskell for HOTTest, and FSMs for TestMaster) and has experience in the design of test cases.

Next, the semantic dependency graph for each tool is constructed, where the nodes are the concepts and the edges represent the dependency relationship between concepts. ***Figure 3*** for instance, shows the semantic dependency graph for Archetest (novice perspective). The graph for the expert perspective differs only in the complexity values associated with each node. The complexity is computed bottom up, according to Equation 1: the complexity values of the child nodes determine the complexity value of the parent node. The complexity of the root is the complexity of the test generation technique. The complexity measures of the four tools are shown in ***Table 1***.

Table 1 Complexity measures for MBTG tools

	Archetest	ASMLT	TestMaster	HOTTest
Complexity (Novice)	64	68	79	68
Complexity (Expert)	33	28	38	31

Ease of learning

Calculating ease of learning is based on proficiency and learning time. The learning time consists of the time to review the learning materials and the time for exercise. Therefore, the learning time is calculated by summing the time spent during the learning sessions and the time spent in developing the model for the application SSP.

Table 2 MBTG tools: major concepts and their proficiency measures

Archetest		ASMLT		TestMaster		HOTTest	
Concepts	*Prof.*	*Concepts*	*Prof.*	*Concepts*	*Prof.*	*Concepts*	*Prof.*
Prologue	0.9	State variables	1	States	1	Function	0.9
Epilogue	1	Stopping conditions	0.9	Models	0.8	Polymorphic types	1
Consistency	1	Update procedures	0.9	Randomizations	0.2	User-defined types	0.9
Activity mirroring	1	Partial updates	0.9	Type	0.9	Basic types	1
Parameter partitioning	0.9	Methods	1	Array I/O	0.2	Records	1
Tofu combinations	1	Values	0.9	Scope	0.2	Sequential flow	1
Result definitions	1	Constraints	0.9	Initialization	0.3	Juxtaposition	1
Context updates	1	Variables	1	IMCF	0.2	Composition	1
Execution templates	1	Condition loops	1	Table models	0.3	Recursion	0.8
Test data	1	Sets	0.9	Context	0.3	Pattern matching	0.95
Inheritance	0.9	Variables	1	Events	1	Case constructs	1
Inclusion	0.9	Constants	1	Action	0.5	If	1
Extension	0.9	Hyperstates	0.8	Predicate	0.3	List comprehensions	0.9
Actors	1	Abstraction	1	Argument	0.5	Relations	0.9
Associations	1	FSM generator	0.9	Parameters	0.4	Attributes	0.8
Activity diagrams	1	Types	0.8	Constraints	0.3	Expressions	1
Classes	1	Instantiation	0.8	Likelihood	0.5	Query	0.9
		Sequences	1	Path constraints	0.2	Restrict	0.9
		Maps	0.9	"@ constraints"	0.2	Project	0.95
		Non-determinism	1	Test information	0.5	Set operators	1
		Enumerations	0.9	Test file set up	0.3	Logical operators	1
		Classes	0.9	Shallow paths	0.5	Boolean connectors	1
		Parameter generation	1	Deep paths	0.5		
				Coverage scheme	0.5		

Table 3 Ease of learning measures for MBTG tools

	Total Learning Time	Learning Sessions	Modeling of SSP	Proficiency	Ease of Learning	Relative Ease of Learning
Archetest	14:44:00	3:56:00	10:48:00	0.97	1.58	91.23
ASMLT	22:21:00	13:18:00	9:03:00	0.93	1.00	57.78
TestMaster	8:21:00	5:00:00	3:21:00	0.44	1.27	73.51
HOTTest	13:10:00	5:00:00	8:10:00	0.95	1.73	100.00

Table 4 Effectiveness and domain-specific effectiveness measures for MBTG tools

		R	*r*	*D*	*d*	Effectiveness (percent)	Domain-Specific Effectiveness (percent)
Technique	Archetest	1260	1050	602	392	83.33	65.12
	ASMLT	1260	965	602	307	76.59	51.00
	TestMaster	1260	843	602	185	66.90	30.73
	HOTTest	1260	1216	602	577	96.51	95.85

Calculating the complexity of the modeling process for each tool first requires an examination of all the concepts involved.

The proficiency calculations are based on the concepts associated with each tool. For each concept, we count the number of usages. A wrong usage of a concept is defined as a usage instance that creates a fault in the test model. Numbers of such faulty instances are also counted. ***Table 2*** lists the major concepts of each modeling technique and the proficiency measure *Prof* achieved in each concept by the subject. The proficiency achieved by the subject is calculated using Equation 2. Whereas Table 2 logs the proficiency achieved by the subject in individual concepts, ***Table 3*** displays measurements of proficiency and learning time. The measure for ease of learning is calculated using Equation 4. Table 3 depicts the ease of learning and related values for the four techniques and includes a *relative* ease of learning measure in which the highest ease of learning value (for HOTTest) is assigned 100.

Effectiveness

The number of requirements covered by a test model is calculated by examining the test suite generated by using a given technique (following transition coverage) and identifying the requirements tested by the test suite. A requirement is said to be tested by the test suite if there are test cases in the test suite that would fail if that particular requirement were not satisfied by the application. The number of requirements and the number of domain-specific requirements covered by using each technique, along with the corresponding effectiveness measures, are presented in ***Table 4***.

The effectiveness values for the test generation techniques are calculated using Equation 5. The coverage attained by the test models is measured by using the parsed requirement list (A3), and it is computed by calculating the fraction of requirements in A3 covered by the test suites. The net number of atomic requirements in A3 for DAT is $R = 1260$. The number of requirements covered by the test models is given in the column with the heading *r*. The number of domain-specific requirements in A3 is $D = 602$ (they were identified to be specific to the domain of database applications). The number of domain-specific requirements covered using the technique is *d*. Of the 602 net domain-specific requirements, 210 were not explicitly mentioned in the original requirements document (A2). Some of the 210 implicit requirements were identified by the expert; others were identified during testing with HOTTest.

Efficiency

The net effort in testing T_{MM} is calculated using Equation 8. The time to model T_M, which is calculated from the user logs during test modeling, includes the debugging time along with the actual modeling time. The test cases were generated on a Pentium** 4 1.6 GHz machine with 256 MB of RAM. The time to generate test cases T_G was recorded by noting the machine time at the beginning and at the end of the test generation activity. The time for setting up the test environment T_S included the time to set up the test harness (Rational* Functional Tester[21]) and the time to derive the test scripts from the test cases. *N* is the number of test cases in the test suite produced from the model allowing the maximum coverage.

Efficiency of the tools is calculated using Equation 6. The coverage attained by the test model is measured by using the parsed requirement list (A3) and is discussed in the section "Case study instruments." Because the target application (DAT) is the same for all test models, we do not evaluate $C_{Complexity} * Application\ Size$. This factor eventually cancels out when we calculate relative measures. ***Table 5***

Table 5 Efficiency measures for MBTG tools with DAT as target application

	T_M (mins)	T_G (mins)	T_S (mins)	T_X (mins)	N	T_{MM} (mins)	Coverage	Efficiency	Relative Efficiency (percent)
Archetest	1554.00	62.00	80.00	940.00	3000	2636.00	83.33	0.032	100.00
ASML	1356.00	1584.00	1822.00	1309.73	4180	6071.73	76.59	0.013	39.90
TestMaster	3030.00	105.00	132.00	1322.89	4222	4589.89	66.90	0.015	46.11
HOTTest	1093.00	320.00	212.00	2052.96	6552	3677.96	96.51	0.026	83.00

Table 6 Efficiency measures for MBTG tools with SearchPUBS as target application

	T_M	T_G	T_S	T_X	N	T_{MM}
Archetest	518	2.00	15.00	10.03	32	545.03
ASMLT	546	17.00	95.00	23.50	75	681.50
TestMaster	201	0.70	45.00	6.89	22	253.59
HOTTest	490	12.00	50.00	26.63	85	578.63

presents the results of the efficiency calculations. Relative efficiency measures are obtained by assigning the value 100 to Archetest.

Scalability

Scalability of the MBTG tools is measured by comparing their efficiency on two target applications, DAT and SearchPUBS. The measurement results for DAT are presented in Table 5; those for SearchPUBS are shown in *Table 6*.

Scalability of effort is calculated for various contributors to the test effort in accordance with Equation 9. Relative scalability measures are obtained by assigning the value 100 to the highest scalability value among the four tools. *Table 7* presents the scalability measures for the various tools for different effort contributors and the net effort.

CONCLUSION

In this paper we present a measurement framework for evaluating MBTG tools. We describe a case study in which the framework was used to compare four MBTG tools. Although this case study is limited in that the data are not statistically significant for the purpose of comparing the tools involved, the experiment is used as a vehicle to describe in detail the steps involved in applying our measurement framework.

We found that the complexity of use is approximately the same for the four tools. This probably indicates that all MBTG tools face a similar barrier for entry. There is roughly a twofold increase in complexity when a novice, rather than an expert, uses the tool. Thus, a person trained in the basics of a testing technique is likely to perceive the complexity to be half of that perceived by a person who is new to testing.

Both ASML and HOTTest involve text-based modeling, which usually results in higher modeling efficiency when compared with graphic-based modeling. It appears that the difference in modeling time between these two modeling approaches increases faster than linearly (perhaps even exponentially) with the size of the system under test. However, graphic-based models are easier to debug than text-based models; therefore, it is important to design the user interface to the specific needs within the test environment. Whereas some tasks are easier when performed with a text-based tool, others benefit from the graphic-based approach. Other factors like compactness of the representation and quick and informative feedback to the user of the tool may also play an important role.

We observed that graph-coverage-based test-generation techniques fail for larger state spaces. This result, which is indeed expected, reasserts that it is nearly impossible to scale up test generation techniques based on graph-coverage algorithms. Another important finding was that implicit requirements, which are automatically uncovered by HOTTest and which are not included in the application requirement document, are quite numerous: the ratio of known and implicit requirements was almost 10:3.

Table 7 Absolute and relative scalability measures for MBTG tools

	Absolute Scalability					Relative Scalability				
	S_M	S_G	S_S	S_X	S_{MM}	S_M	S_G	S_S	S_X	S_{MM}
Archetest	9.65E-04	1.67E-02	1.54E-02	1.08E-03	4.78E-04	58.20	100.00	100.00	100.00	100
ASMLT	1.23E-03	6.38E-04	5.79E-04	7.77E-04	1.86E-04	74.44	3.83	3.76	72.30	38.7919
TestMaster	3.53E-04	9.59E-03	1.15E-02	7.60E-04	2.31E-04	21.31	57.53	74.71	70.67	48.2202
HOTTest	1.66E-03	3.25E-03	6.17E-03	4.94E-04	3.23E-04	100.00	19.48	40.12	45.89	67.4654

We believe that the future of model-based test generation techniques lies in developing tools with good user interfaces that use intelligent test-generation techniques, rather than brute-force graph-coverage schemes, and that are able to automatically use domain-specific knowledge when performing test generation. We are working on a release of Archetest that incorporates improvements based on these findings.

ACKNOWLEDGMENT

We thank Ms. Theresa Kratschmer for her valuable comments on an earlier version of this paper.

CITED REFERENCES AND NOTE

1. Workshop on Advances in Model-Based Software Testing (A-MOST), 27th International Conference on Software Engineering, St. Louis, Missouri, May 15–21, 2005 ACM, New York (2005).
2. S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-Based Testing In Practice," *Proceedings of the Twenty-First International Conference on Software Engineering*, May 16–22, 1999, Los Angeles, CA, ACM, New York (1999), pp. 285–294.
3. A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "One Evaluation of Model-Based Testing and Its Automation," *Proceedings of the 27th International Conference on Software Engineering*, May 15–21, 2005, St. Louis, Missouri, ACM, New York (2005), pp. 392–401.
4. M. Veanes, C. Campbell, W. Schulte, and N. Tillmann, "Online Testing with Model Programs," *Proceedings of the 10th European Software Engineering Conference (ESEC) and the 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (SIGSOFT FSE)*, Lisbon, Portugal, September 5–9, 2005. ACM, New York (2005), pp. 273–282.
5. A. Paradkar, "A Quest for Appropriate Software Fault Models: Case Studies on Fault Detection Effectiveness of Model Based Test Generation Techniques," *Proceedings of the First International Workshop on Advances in Model-Based Software Testing, 27th International Conference on Software Engineering*, St. Louis, Missouri, May 15–21, 2005 ACM, New York (2005).
6. Grove Consultants, http://www.grove.co.uk/.
7. Ovum, http://www.ovum.com/.
8. V. R. Basili, "Goal Question Metrics Paradigm," in Encyclopedia of Software Engineering, J. Marciniak, Editor, John Wiley and Sons (1994), pp. 528–532.
9. C. E. Williams, "Toward a Test-Ready Metamodel for Use Cases," *Proceedings of the Workshop on Practical UML-Based Rigorous Development Methods*, October 1–5, 2001, Toronto, CA (2001), pp. 270–287.
10. M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes, "Model-Based Testing with AsmL.NET," *Proceedings of the 1st European Conference on Model-Driven Software Engineering* (December 2003), http://www.agedis.de/conference/presentation.shtml.
11. *TestMaster User's Guide, Release 1.9.5*, Empirix Inc., New Hampshire, 1999.
12. P. Savage, S. Walters, and M. Stephenson, "Automated Test Methodology for Operational Flight Programs," *Proceedings of the IEEE Aerospace Conference* **4**, pp. 293–305, IEEE, New York (1997).
13. R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary, and C. Bourhfir, "Test Development For Communication Protocols: Towards Automation," *Computer Networks* **31**, No. 17, 1835–1872 (1999).
14. A. Sinha, C. Smidts, and A. Moran, "Enhanced Testing of Domain-Specific Applications by Automatic Extraction of Axioms from Functional Specifications," *Proceedings of the 14th International Symposium on Software Reliability Engineering*, November 17–20, 2003, Denver, CO, IEEE, New York (2003), pp. 181–190.
15. Usually the effectiveness of a test process is measured as the fraction of the total number of faults that are uncovered by testing. This value is estimated by embedding faults in the code to be tested and examining the results of the testing process. Because we use the target application in a black-box fashion, that is, without modifying the code, we had to create an alternative metric.
16. A. J. Albrecht, "Measuring Application Development Productivity," *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, Monte-

rey, California, October 14–17, IBM Corporation (1979), pp. 83–92.

17. D. Leijen and E. Meijer, "Domain-Specific Embedded Compilers," *Proceedings of the Second Conference on Domain-Specific Languages (DSL '99)*, Austin, Texas, USA, October 3–5, IEEE, New York (1999), 109–122.
18. K. Bassin, T. Kratschmer, and P. Santhanam, "Evaluating Software Development Objectively," IEEE Software **15**, No. 6, 66–74 (November/December 1998).
19. IEEE Guide to Software Requirements Specification, ANSI/IEEE Standard 830, IEEE (1984).
20. D. T. Campbell and J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research*, Houghton Mifflin, Boston, MA (1963).
21. IBM Rational Functional Tester, IBM Corporation.

Accepted for publication March 7, 2006.
Published online July 11, 2006.

Avik Sinha
IBM Thomas J. Watson Research Center, 19 Skyline Drive, 2NF07, Hawthorne, New York 10562 (avisinha@us.ibm.com).
Dr. Sinha is a post-doctoral researcher at the Watson Research Center. He holds a B.Tech. degree from Indian Institute of Technology, Kharagpur, and M.S. and Ph.D. degrees from the University of Maryland at College Park. His areas of interest include software engineering, model-driven software development, software testing, and domain-specific test generation.

Clay E. Williams
IBM Thomas J. Watson Research Center, 19 Skyline Drive, 2NB04, Hawthorne, New York 10562 (clayw@us.ibm.com).
Dr. Williams is a research staff member and manager of the Software Quality and Testing group. He has a Ph.D. in computer science from Texas A&M University. His areas of interest include software engineering, model-driven software development, software testing, and medical applications of information systems. He is a member of the IEEE and ACM.

P. Santhanam
IBM Thomas J. Watson Research Center, 19 Skyline Drive, GNB02, Hawthorne, New York 10562 (pasanth@us.ibm.com).
Dr. Santhanam has a B.Sc. degree from the University of Madras, India, an M.Sc. degree from the Indian Institute of Technology, Madras, an M.A. degree from Hunter College of the City University of New York, and a Ph.D. from Yale University. He joined IBM Research in 1985, where he is currently Senior Manager in charge of the Software Engineering department, whose mission is to develop tools and methodologies in support of the software development process. He has authored more than 40 technical papers on a wide variety of topics in peer-reviewed journals and conference proceedings. Dr. Santhanam is a member of the ACM and a senior member of the IEEE. ■