The TOBIAS test generator and its adaptation to some ASE challenges
Position paper for the ASE Irvine Workshop

Y. Ledru
Laboratoire Logiciels Systèmes Réseaux/IMAG
BP 72, F-38402 Saint-Martin-d'Hères CEDEX, FRANCE
Yves.Ledru@imag.fr

Abstract

In the past decade, a scientific community has emerged around the notion of "Automated Software Engineering". This community has made several advances in two kinds of challenges: the complexity of processing software engineering information, and the difficulty to capture knowledge about software. This position paper first recalls these challenges. It then describes how these challenges influenced the design of the TOBIAS test generation tool.

1 Challenges of ASE research

Automated Software Engineering tries to develop software tools that help in the software development activities. Such tools start from digital information and try to produce other digital information for the software engineer. Most of the time, this digital information is a structured or a formal document. Structured or formal documents include: source code, tests, formal specifications, but also semi-formal specifications (e.g. UML diagrams) or structured documents (e.g. XML documents). At longer term, research work on natural language recognition (both spoken and written) may increase the spectrum of potential input documents.

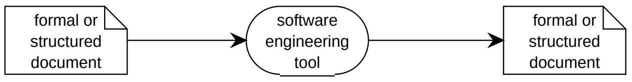

formal or structured document
software engineering tool
formal or structured document

Figure 1. Structure of an ASE tool

The first challenge faced by ASE tool designers is to design efficient and powerful tools. This is not a trivial task, and it is confronted to fundamental problems.

- The complexity of the input documents: real-life applications usually involve thousands or millions of artifacts (lines of code, diagrams elements, ...). This complexity is inherited by the tools that process these artifacts, and it requires optimisations even in the case of linear algorithms.
- Many algorithms in this field have a non-linear complexity. They must face the challenge of combinatorial explosion (e.g. in model-checking techniques).
- In many other cases, the input documents use languages whose expressiveness makes processing activities undecidable (e.g. theorem proving on first order predicate logic).

These three fundamental problems are at the heart of ASE research, and significant advances have been made in each of these fields. One of the main results of the ASE community was the identification of domain specific knowledge which helps face these complexity and decidability problems. By accumulating knowledge about a class of problems, it is possible to design domain specific tools, or tools which use a domain knowledge base. Such tools exploit the domain information to narrow their search for solutions and converge more rapidly.

This introduces a second challenge: how do you capture domain knowledge and other relevant information. Although much effort has been done in order to educate software engineers to the virtues of specification and documentation, many software development activities still correspond to CMM level 1. Approaches to software engineering like Extreme Programming even try to reduce the production of documents to a minimum, taking for granted that software engineers only like to write code! Also the widespread use of computer technology has turned millions of people into amateur software engineers, with poor education in software development techniques.

Two kinds of answers have been proposed to this second challenge.

- The first answer is to provide useful and efficient tools. It is important that tools bring some benefits and are applicable to real-size software. Benefits can be of two kinds: either an improvement in productivity, or a speed-up of the process, or improvement in quality. If any of these factors (productivity, time, quality) is a critical factor for a company, it will motivate efforts to adopt the new technology. The capabilities of software engineers to adapt themselves and their process to new technology should not be underestimated. Software engineers are confronted to a constant evolution of target technologies (programming languages, hardware and software platforms). They have the ability to learn new specification languages if they perceive the induced benefits.

- The second answer is to design formalisms or tools that are easy to use for the software engineer. Much work has been done in trying to design graphical formalisms, supported by GUI tools. This approach was successful in several projects. For example, the AMPHION project uses a graphical relational language as input that was used with success by experts in astrophysics to formalize their problems.

- The third answer is the integration of the tool in the development process. Many novel approaches are based on new activities and new notations. They require a revolution in the way software is developed. In most cases, a company cannot afford such a revolution because it not only requires to educate its engineers, but also to deeply reorganise the company itself or its processes (and hence loose some maturity during a transition period).

Obviously, the cost of integrating new techniques will be compared to the expected benefits. From there, several approaches can be adopted by ASE tool designers: some will try to maximize the benefits of their tools, whatever be the cost, others will develop more modest tools which bring less benefits at a lower cost.

These challenges are not new and they have already been reported in various studies related to ASE, software engineering or formal methods. The rest of this paper will present the TOBIAS tool[1] , aimed at the generation of large testsuites and will discuss how these challenges are taken into account by the tool.

2 TOBIAS

TOBIAS is a tool for the automatic generation of test cases from a given test pattern. Writing test cases is a very tedious and repetitive task, especially when we need a large set of test cases. This is where TOBIAS helps produce a

[1]TOBIAS has been developed within the COTE project of the french national network in software technology (RNTL).

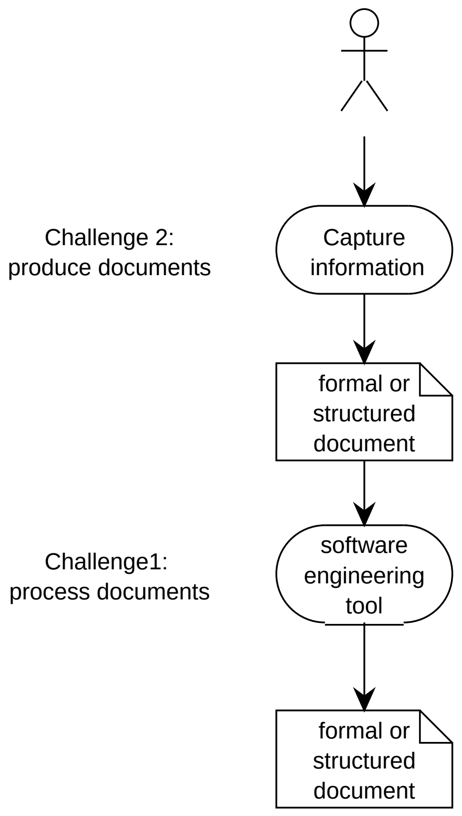

Challenge 2:
produce documents
Capture information
formal or structured document
Challenge1:
process documents
software engineering tool
formal or structured document

Figure 2. Challenges of ASE research

large set of similar test cases. We have experimented that many test cases feature the same sequence of operations but with different parameter values [1]. Other sequences may also differ by exchanging an operation with a similar one.

TOBIAS allows the user to define a set of relevant values for each operation parameter or to identify sets of similar operations (named “groups” in TOBIAS). These form the basis for the definition of test patterns (named “test schemas” in TOBIAS). A test schema is a bounded regular expression over operations and groups. Test schemas are then unfolded by TOBIAS into a large set of test cases.

We expect that TOBIAS will help test engineers generate more tests cases and in a more systematic way for about the same effort than “manually” written test cases. Generating more tests may increase the confidence in the testsuite. Generating these more systematically will help cover more behaviours of the system, including situations that could be overlooked or forgotten by the test engineer. So we may reasonably expect that TOBIAS increases the chance of detecting errors.

In a recent experiment [3], we generated a large test suite (4320 test cases) and compared it to a manually produced test suite (45 test cases). Our experiment showed that

- The testsuite generated by TOBIAS discovers more errors than the manual testsuite. It also exercised some known errors in several different ways, making the testsuite more robust towards evolutions of the specification.
- Writing TOBIAS test schemas requires a similar effort than writing the small manual testsuite.

2.1 Principles of TOBIAS

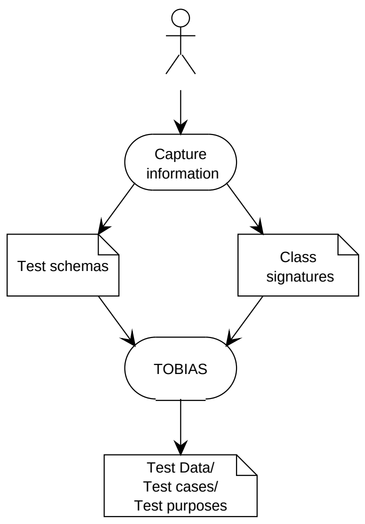


Figure 3. Basic view of TOBIAS

TOBIAS takes two inputs (Fig. 3):

- the signatures of the classes of the application under test
- a test schema

and produces sequences of method calls which can be used as test data, test cases (if an oracle is available) or test purposes such as the ones required as input of the TGV tool [2].

For example, let us consider a simple class "IntegerSet" with two methods: "add(v:int)" and "remove(v:int)". Starting from this signature and the following test schema:

```
add(x)ˆ1..3;remove(y)ˆ0..2
where
x : {0,1,2,3}
y : {0,1,2}
```

TOBIAS will generate all sequences which feature one to three calls to "add" with values 0 to 3, followed by zero to two calls to "remove" with values 0 to 2. In total, this schema generates 1092 different sequences ((4 + 4*4 + 4*4*4) * (1 + 3 + 3*3)). Fig. 4 shows some of the generated sequences.

```
1: add(0)
2: add(1)
3: add(2)
4: add(3)
5: add(0); add(0)
6: add(0); add(1); add(2)
7: add(2); add(1); add(0)
8: add(0); remove(0)
9: add(0); remove(0); remove(0)
10: add(1); remove(0)
11: add(0); add(1); add(2);
    remove(0); remove(1)
```

Figure 4. Some test cases generated by TOBIAS

They correspond to classical test cases like adding or removing several elements, but also adding or removing twice the same element, or trying to remove an absent element. Actually, in order to turn the output of TOBIAS into test cases, you need an oracle which will evaluate the effects of method calls and deliver a verdict.

The test schema (4 lines) specifies this large set of test cases, and TOBIAS helps the software tester to construct this test suite at a lower cost than manual production of the test cases. This simple example shows how test schemas generate test cases by iterating over parameter values and the number of successive calls to the same method. TOBIAS also allows to iterate over a set of instances of the class or over a set of methods.

TOBIAS is a tool that amplifies the work of the test designer. The tool is based on a simple idea: to exploit similarities between test cases in order to specify these by a generative pattern. In the next sections, we will see how it addresses the challenges of ASE.

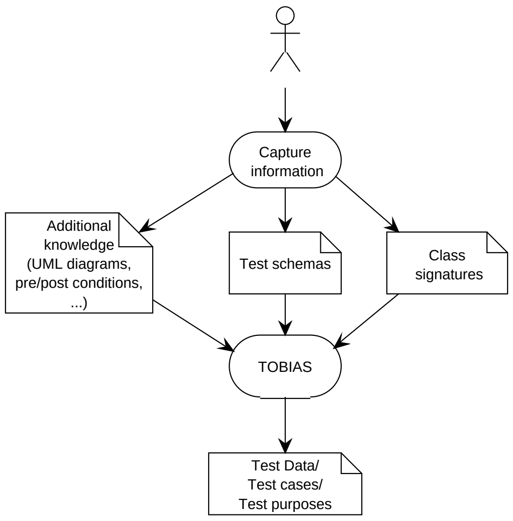
3 Tobias and complexity

Unfolding a TOBIAS test schema is not intrinsically difficult, but the output is subject to combinatorial explosion: the tool generates a large number of test cases, and it is precisely the purpose of the tool. Having a large number of systematically generated test cases helps finding more errors because it exercises the combinatorial complexity of the application. Still, the testsuite may not be arbitrarily large because its execution may require untractable resources for minor additional benefits. Therefore, the challenge of TOBIAS users is to produce an optimal number of test cases.

To this end, it is necessary to select a subset of the generated testsuite. Two kinds of test cases should be eliminated from the testsuite:

- redundant test cases:
- non-conform test cases.

For example, in Fig. 4, test cases 1,2,3 and 4 are redundant, because they correspond to add a single element and the result of the test should not be influenced by the actual value of the parameter. Similarly, test cases 6 and 7 are also redundant. Test cases 9 and 10 could correspond to non-conform test cases if the pre-condition of remove requires that the element that is removed is an element of the set.



Capture information
Additional knowledge (UML diagrams, pre/post conditions, ...)
Test schemas
Class signatures
TOBIAS
Test Data/ Test cases/ Test purposes

Figure 5. Providing more information to TOBIAS

In order to detect redundancy and non-conformance, additional information must be provided to the tool (Fig. 5). This information can take two forms.

- Several specification documents (UML diagrams, state machines, pre-/post-conditions,...) can be exploited in order to detect and eliminate non-conform test cases. This process can become difficult if these specifications are complex or written at a very high level of abstraction. They may require the use of verification techniques with their usual cost. Still, if the tester does not need to detect every non-conform test case, he may use simpler information or less expensive algorithms which will detect some kinds of non-conformances. For example, TOBIAS will soon take into account the relations of the UML class diagram in order to detect test cases that feature non-conform communication between instances [2]. This kind of verification is quite elementary but may lead to the elimination of a significant number of non-conform test cases. We are also working on the integration of TOBIAS with the CASTING tool [4] which takes into account pre-/post-conditions and a state transition diagram to detect non-conformant test cases. Here, the tool involves a more complex computation, using constraint logic programming techniques.
- The language for expressing test schemas can be extended to allow the test engineer to provide a finer description of the test schema and to express test hypotheses. For example, in the IntegerSet class, values of integers are not significant. This equivalence can be provided as a test hypothesis and used by TOBIAS to avoid redundant test cases. Currently, we are experimenting an extension of the language that allows the test engineer to specify constraints on the values of the parameters used in a test case. For example, the user can express that the x_i parameters are pairwise distinct, which would eliminate test case 5.

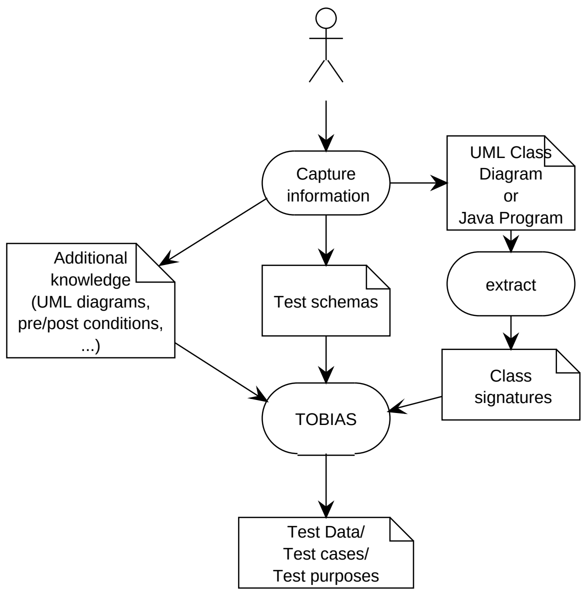
To summarize this point, TOBIAS is exposed to the combinatorial complexity of the generated testsuite. This complexity is intrinsic to the tool because we wanted a tool that would systematically test combinations of values and method calls. Still, it has to be controlled because too large testsuites may require untractable resources without providing additional benefits. Therefore several techniques are used to generate more pertinent test cases, but these techniques require additional domain information about the tests or about the application. As for many ASE tools, mastering the complexity requires additional domain knowledge. In the next section, we will see how this knowledge is captured in the context of TOBIAS.

4 Capturing knowledge for TOBIAS

Fig. 3 shows that TOBIAS requires two kinds of inputs: a test schema and signatures of the classes. In many applications, these signatures can be extracted either from the code of the application under test, or from its UML specification. The current version of TOBIAS is based on the UML class diagram because the tool was developed in a project where the availability of such documents is taken for granted. Very soon, we plan to allow this extraction from Java source code also (Fig. 6).

If signatures are extracted from existing UML specifications or from Java source code, TOBIAS can be used by

[2]Actually, a test schema allows to specify not only the method invoked but also the instance which will activate the method call, and the instance which will process the call.



Capture information
UML Class Diagram or Java Program
Additional knowledge (UML diagrams, pre/post conditions, ...)
Test schemas
extract
Class signatures
TOBIAS
Test Data/ Test cases/ Test purposes

Figure 6. TOBIAS with additional knowledge

simply providing a test schema. The example of Sect. 2.1 shows that test schemas are expressed in a few lines. The current version of TOBIAS also provides a graphical user interface to help the user define a test schema. For example, the interface prompts the user for values of the parameters, or for names of the instances which will execute these methods. It must be noted that the test schema encourages the user to provide specific information about the application under test. For example, the choice of values for the method parameters forces the user to analyse which values are of interest; the quality of the information provided by the user has a direct effect on the quality of the test suite.

TOBIAS has been designed to allow a new user to start using the tool at low cost. Starting from an existing class diagram, the user only has to provide a first test schema. He will immediately get a first large sequence of method calls. We expect that the user will then try to refine this sequence either by using the extensions of the schema language (e.g. constraints), or by providing more information to the tool about conformance issues. This conformance information is actually a specification of the application under test. The planned extensions of the tool will try to exploit existing UML diagrams of the application under test (class diagram, state transition diagram, OCL pre/post-conditions). If such specifications don't exist, we hope that the benefits of producing more pertinent and conformant test suites will encourage the software engineer to specify parts of the application. Again, we plan that the tool will be able to bring small but useful results from simple elements of the specification (e.g. the relations in the class diagram), and more precise and complete results from detailed specifications.

Another way to encourage the user to provide such specifications is to integrate TOBIAS with other test generation tools which are based on the same kinds of diagrams. In the COTE project, TOBIAS will be combined with UMLAUT/TGV and CASTING, which both exploit elements of UML specifications. Specifications, preferably executable ones, can also be used as a basis for the test oracle, in order to turn sequences of method calls into real test cases.

In summary, our approach is to provide the first benefits at the single cost of expressing test schemas. It exploits the availability of several documents (source code, UML specifications), and hence should be easier to integrate in the company process. Then, it encourages the evolution of the processes by delivering new benefits for additional elements of specification.

5 Conclusion

This paper has shown how the TOBIAS test generator tries to face two of the major challenges of each ASE tool: complexity and information acquisition. Our approach is to start with simple solutions that fit into standard software development processes. Then we intend to gradually incorporate more refined processing based on more precise information.

Acknowledgments

TOBIAS is the result of joint efforts with Lydie du Bousquet, Pierre Bontron, Olivier Maury and several students of UJF. I'd also like to thank our colleagues of the COTE project, in particular our colleagues from Gemplus who provided motivation for developing the tool.

References

[1] L. du Bousquet, H. Martin, and J.-M.Jézéquel. Conformance testing from UML specifications - experience report. In *UML2001 Workshop on Practical UML-Based Rigorous Development Methods*, Toronto, 2001.

[2] T. Jéron and P. Morel. Test Generation Derived from Model-checking. In *Computer Aided Verification (CAV'99)*. LNCS 1633, Springer, 1999.

[3] O. Maury and Y. Ledru. Using TOBIAS for the automatic generation of VDM test cases. In *VDM workshop*, Copenhagen, Danemark, 2002.

[4] L. Van Aertryck, M. Benveniste, and D. Le Métayer. Casting: A formally based software test generation method. In *The 1st Int. Conf. on Formal Engineering Methods, IEEE, ICFEM'97*, Hiroshima, 1997.