

# A KNOWLEDGE MANAGEMENT APPROACH FOR INDUSTRIAL MODEL-BASED TESTING

Dmitrij Koznov

*St.-Petersburg State University, Faculty of Mathematics and Mechanics  
Department of Software Engineering, 198504 Bibliotechnaya sq., 2, St.-Petersburg, Russia,  
dkoznov@yandex.ru*

Vasily Malinov, Eugene Sokhransky, Marina Novikova

*DataArt, Inc., 475 Park Avenue South, Floor 9, New York, NY 10016, USA  
{malinov, novikova\_sokhransky}@dataart.com*

**Keywords:** Knowledge Management, Model-Based Testing, Partial Specifications, Requirement Recovering, Architecture Recovering, UniTask.

**Abstract:** This paper offers a knowledge management method for industrial model-based testing, which based on partial specifications and “attached” to the software development process that uses it. Partial specification means formal description of considerable/potentially problematic properties of a system, and is used for further automated testing. That allows reducing expenses of testing compared to developing full formal specifications. The “attached” nature of the method means that the team of testers can work independently of the basic process, without imposing on it any specific limitations connected with model-based testing. The method intends for lightweight processes where a lack of documentation and formal described requirements are absent. The paper also presents approbation of the method while testing an industrial Web-application by means of model-based testing technology UniTask in DataArt Inc. software company.

## 1 INTRODUCTION

Formal methods are particular kind of mathematically-based techniques for the specification, development, verification and testing of software systems. Model-based testing is one of the formal methods and uses formal system specifications for automatic generation of tests and testing environment. The advantages of formal methods are commonly known – these methods provide high guarantees of correctness of objects which they applied for. However there are barriers to their wide dissemination in industry, as discussed in (Hinchey, M., et. al., 2008), (Knight, J., 1998). This problem is so important that a special conference ISoLA is organized to surmise efforts of the academic and industry communities to resolve it. It would be said, that one of the main obstacle for moving formal methods from universities to industry is a lack of proper knowledge management methods

in this area. Scientists offer their methods mainly focusing on languages, algorithms, and other knowledge that could be expressed formally and is absolutely explicit. But a lot of tacit knowledge is not taken into account: motivations, physiological issues, personal background, etc. There is no enough attention paid to development and dissemination of patterns, best practices, guidelines for special kinds of software and for different types of development processes. There are no estimations of formal methods effectiveness for various projects sizes. A reasonable implicit work that should be done to use formal methods efficiently stays also in shadow. For example, the size of formal specifications is often comparable with the size of tested system; or using formal methods often puts forward additional requirements to the development process, e.g. model-based testing requires well-defined and documented requirements, which is not always done in real projects. So research focus in the formal

methods should be really shifted from what to apply to how to apply. It would be say using semiotic lexicon that formal methods need pragmatic above syntax and semantic.

This paper offers a knowledge management method for model-based testing based on partial specifications and “attached” to the development process that uses it. Partial specifications are actively used in the context of formal methods and are not an unambiguously defined notion – see surveys (Easterbrook, Callahan, 1997), (Johnsen, Owe, 2002), (Hendrix, Clavel, Meseguer, 2005). Here partial specification shall mean formal description of software system developed from model-based testing perspective, and describing not the system as a whole but only some set of system’s features, which important for the customer and/or potentially problematic and labour-intensive for testing. So we considerably reduce testing expenses and provide suitable quality. If we would like to provide exhaustive testing covering than the size of formal specification became near to the size of code, and only a few software projects can provide correspondent testing resources. The “attached” nature of the method means that the team of testers can work independently of the basic process, without imposing on it any specific limitations connected with model-based testing. That allows localizing specific work for utilization mathematized tools without making the process more complicated, in particular with detailed description of the requirement. That will also allow performing model-based testing for system with different readiness degree in the outsourcing mode. On the other hand this approach requires the use by testers of special knowledge mining methods to get the needed information about the systems. The central approaches here are requirement recovering and architecture recovering.

The paper also presents approbation of the method while testing an industrial Web-application with the help of model-based technology UniTask (Bourdonov, 2002), (Kuliamin, 2003) in DataArt Inc. software company<sup>1</sup>.

<sup>1</sup> <http://www.dataart.com/>

## 2 BACKGROUND

### 2.1 Knowledge Management in Software Engineering

Knowledge Management (KM) comprises a range of practices used in an organization to identify, create, represent, distribute and enable adoption of insights and experiences (Nonaka, 1991). Core components of KM include people, processes, technology (or) culture, depending on the specific perspective (Spender, Scherer 2007).

Software engineering is a complex business that involves many people working in different phases and activities. The knowledge in software engineering is diverse and its proportions immense and grow. Software engineering involves a multitude of knowledge-intensive tasks: analyzing user requirements for new software systems, identifying and applying best software development practices, collecting experience about project planning and risk management, and many others (Birk, et. al., 1999). Software companies have problems keeping track of what this knowledge is, where it is, and who has it. A structured way of managing knowledge could help them to improve development process essentially, to make easier introducing of the new technologies and to meet clients requirements more thoroughly. Survey of KM methods and tools applied for Software Engineering could be found out in (Engelhart, 2001).

In (Engelhart, 2001) identified following categories of software engineering tasks to which KM is applicable:

1. Tasks performed by a team focusing on developing a software product based on customer requirements.
2. Tasks that focus on improving a team’s ability to develop a software product (that is improving tasks in the first category).
3. Tasks that focus on improving an organization’s or an industry’s ability to develop software.

The method offered in the paper focuses mainly on the task 1, partly including task 2, and not including task 3.

### 2.2 Model-Based Testing

Model-based testing is software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test (Utting, Leguard, 2007). The model is built before or parallel to the development

process of the system under test, and it can also be constructed from the completed system. Usually the model is created mostly manually, but there are some attempts to create the model automatically, for instance out of the source code.

### 2.3 UniTask: Model-Based Testing Technology

This technology having been developed, implemented and used for many years (Bourdonov, 2002), (Kuliamin, 2003). It assumes making contract specifications with consequent automatic generation of tests and oracles. Test scenarios are organized on the bases of finite state machines. The technology provides an opportunity to use common programming languages for development of formal specifications, which are extended with some additional constructions. Hence if the tested system is developed using languages Java, C# etc., the test specifications shall be developed in the same languages. The technology provides a range of software products for different development platforms. We used product @Chaise for Microsoft .NET Studio/C# environment.

### 2.4 Requirement Recovering

Approach FOREST (Kuliamin, Pakulin, Petrenko, 2005) was developed by the authors of UniTask and in fact is a requirement recovering method providing suitable input for model-based testing. It implies that good written sources of information, such as documented requirements, standards (e.g. telecommunication standards) are available.

Approach AMBOLS (Liu, K., 2005) is intended for restoring requirements for legacy information systems with consequent substitution of the old system with the new one. The source of information for AMBOLS is the system users and the system itself that is treated as a working application. The source codes and documents are desirable but not required as in practice they are considerably incomplete or unavailable. The basic tools used in AMBOLS are visual models methods and organizational semiotic methods.

In opposite AMBOLS the approach presented in (El-Ramly, Stroulia, Sorenson, 2002) is an example of an automated approach to requirements recovering on the basis of analyzing the source texts of the system.

All these approaches aim at finding all the requirements, and are not oriented to partial requirement recovering in accordance with some special criterias. Nevertheless these approaches may be used in our method, especially FOREST for

processing documental requirement sources, AMBOLS and other similar ones for applying visual models.

### 2.5 Architecture Recovering

There is a huge number of methods in this sphere. One of the most general contexts of architecture recovering is software evolution. (Mens, T., Demeyer, S., 2008) is one of the recent papers on this subject giving references to further reading. We would like to mention specially two more papers (Jansen, Bosch, Avgeriou, 2008), (Koznov, Romanovsky, Nikitin, 2001) dedicated to architecture recovering of “living” systems, i.e. in a situation where all sources of information about the architecture (except documentation) are available and system is actively developed and maintained.

### 2.6 Partial Specifications

The idea to build partial specifications in the context of formal methods is not new – see (Easterbrook, Callahan, 1997), (Johnsen, Owe, 2002), (Hendrix, Clavel, Meseguer, 2005).

Often partial specifications are understood as a simplified way for creating full system specifications. Approaches described in (Letichevsky, Kapitonova, 2004), (Falcone, Fernandez, Mounier, 2007), (Petrenko, Yevtushenko, 2005) are based on that. Besides in the context of building a full formal model partial specifications are also used to have the opportunities of independent work with different items of a software code, which is used for example when creating models of object-orientated applications (Johnsen, Owe, 2002).

Partial specifications are also often used to extract a specifications of system/component interface to use ones for black-box verification and testing – see e.g. UniTask approach. In (Acharya, Xie, Pei, 2007) partial formal specifications are built and used for testing the interaction between the system and outside modules.

(Tichomirov, Kotlyrov, 2008) considers a situation when a new component is added into an existing system. The paper puts forward the idea of developing formal specifications only for the part of the system that directly interacts with the new component. Further model-checking of the component/system interface is performed on the basis of such partial specifications. This paper is the nearest to our ideas, although it is used for model-checking and not for model-based testing. It is possible to say that we generalize it, as we permit

arbitrary system properties, not only an interface of the system and the new component.

### 3 METHOD

We suppose that there is a software company which has an experience and/or desire to apply some model-based testing technology in one of its project. We use term 'desire' instead 'needs' due to which there are a lot of alternatives to provide software quality, and not every company is ready to use formal methods for that purpose. It means there should be some motivations to apply some special technology or a class of such technologies and some preliminary work should be done – and all that is beyond the method.

Our method should be applied as following sequence of steps.

1. *General studying of the system*: initial acquaintance with the system, overview of all the system requirements, identification of requirement sources.
2. *Elaboration of requirements*: defining quality level which is significant for the customer or influences the general system robustness, and extracting exact important/potentially problematic system properties.
3. *Making decision*: making decision to use model-based technology, taking into account technology availability, project needs, resources for testing.
4. *Studying*: carrying out necessary learning of the testers.
5. *Development of formal partial specification* of the system, i.e. formalizing its properties defined above.
6. *Testing process setup*: deployment and setting of the testing software environment, setup of the whole testing process.
7. *Testing process execution*.

#### 3.1 General Studying of the System

The main focus of this step is understanding by testers system in general and as a whole, overview all requirements. For these purpose different requirement recovering methods presented above can be used depending on the availability of some or other sources of information, type and size of the system under testing. Expediency and intensiveness of using this step depends on the degree of "intimacy" of the testing team with the project. Here it is possible that testers either participate in the project from the very beginning or join it at a certain

stage. In the first case this step of the method is not required as the system is studied in a natural way. In the second case it is necessary. Besides, testers can be either a part of a project team or an stand alone team (e.g. outsourcing testing). In the latter case the importance of this step of the method is especially great.

The system study should not be too long and the following step should be made as soon as possible.

#### 3.2 Elaboration of Requirements

It is not possible to find out and correct all errors in software. As a role in each particular project there is some implicate quality needs and correspondently some amount of project recourses which could be consumed for testing. This step aims at clarifying this knowledge, and balancing expectations and available recourses. In practice, as everybody knows, not absolute but some real quality is required in every particular case, this quality level is unique for each project, hence more or less resources are provided for testing.

When selecting system prosperities for intensive testing it is important to pay attention to the system functionality, the quality of which is important or critical for the customer, as well as to the problematic from the quality point of view single components/group of components. In the first case requirement recovering methods mentioned above should be used, because we intend out approach for lightweight processes where a lack of documentation and formal described requirements are absent. In the second case the above specified architecture recovery methods turn to be useful when testers study the system. In such a situation the requirement recovering alone will not be enough. In both cases testers could use software architectures, project managers, developers as a main knowledge sources. Also running system might be another source of information.

#### 3.3 Making Decision

In this step the quality requirements to the software should be analyzed, and comparing resources available for testing (people and their qualification, time etc.) with testing technologies, existing tests packages and other knowledge available in a company should be carried out. As a result we make a decision about using various means and technologies for testing various system properties. It may be possible that system's complexity and quality requirements are not so serious and manual testing would be enough.

If we make a decision about using model-based testing technology, then we have to take into account that we are talking about new software application for testing our target software. This application includes testing specifications, mediators, scenarios, etc., and should be designed, developed, and debugged properly. That is why it is important to evaluate the labour intensiveness of its implementation thoroughly comparing planned costs and available resources.

The selected software features to be tested basing on model-based technology are analyzed of the following parameters.

1. Presence of a great number of behaviour paths – that means that it is expedient to use model-based methods.
2. Expenses of implementation and debugging of test specifications as well as implementation of access to the system and testing. The costs may overpass the benefits.

The rest properties would be tested manually. Careful selection of software features, which are reasonable to be tested using mode-based approach, is also very important. This approach provides for a good coverage of software behavior around selected features but it takes a reasonable efforts to be implemented, and a lot of computational resources are consumed to perform final testing. So deficiencies reduce efficiency substantially in this case. Unfortunately there are no suitable sources of such knowledge in model-based testing community. This knowledge often appears as a tacit one of technology authors, and could be extracted only when they participate in testing directly. This situation is one of the main reasons of large labour costs and limited success of model-based testing performed by non-authors of corresponding technologies.

### 3.4 Studying

Teaching model-based testing technologies is a very important aspect for successful application. This field is so that it is not easy to learn in process. Very often it is necessary to teach as fundamentals of model-based testing approach as operation rules for relevant instruments. However the exact scope of teaching depends on primary background of testers and can be reduced if testers are competent in computer science (especially in mathematical logics). Here we do not speak about general training in the whole company, which is necessary for the familiarization of the stuff with model-based approach. It is assumed that teaching is carried out

for testers' team to cover only some knowledge gaps.

### 3.5 Development of Formal Partial Specification

Design and development of test specifications take place at this step. Each feature (or 'family' of tightly coupled features) selected at the previous steps should be placed into a separate component, independent from any other. Nevertheless this may not be the case and different features will be dissipated about the whole formal specification. It would happen as a result of the "attachment" of the specification code to the software interface elements of the tested application. Our recommendation is, as far as it is possible, to create different components for different tested features even if they use the same entities of this interface. These components are convenient for debugging, testing, using and managing.

### 3.6 Testing Process Setup and Execution

Steps 6 and 7 of the method have no specific and should be done as usual.

## 4 EXAMPLE

We have applied the above offered method for testing of a Web-application with the help of UniTask technology. This Web-application is intended for creating and editing requests for placement of advertisements through the Internet in different printed media. By the moment of starting our test the application had been already delivered to the customer and worked.

The application was developed by usual for such projects scenario. The customer specified the purpose of the system and its basic functions that had to be delivered. After that the first system prototype was developed and shown to the customer he proposed some modification, etc. The system documentation was not adequate and in fact was almost completely missing. The main sources of information for the testers were the working application itself and the project manager who agreed to answer our questions. As a result we defined the following potentially problematic system properties:

- The user interface of the application that looked rather complicated and with high degree of probability could contain mistakes.

- Presence of a great number of attributes in the application core object (advertisement) made it possible to assume the possibility of situations when due to absence of required checks several incompatible attributes are selected and the integrity of the application data is distorted.
- Presence of a great number of steps and branches in the main wizard of the application resulted in the probability of incorrect transitions. This was especially true about reverse transitions.

These features assumed a great number of scenarios, due to which their “manual” testing with good coverage was a rather labour intensive task (that is why after the system delivering mistakes were left there).

We developed testing specifications and set up UniTask. As a result 11 mistakes were found, 4 of which completely destroyed the advertisement edited by the application, and to continue work it was necessary to restart the application and start the whole session anew.

The labour costs have been structured as follows:

- General studying of the system – 5%;
- Elaboration of requirements – 5%;
- Making decision – 10%;
- Studying – 14%;
- Development of formal partial specifications – 20%;
- Testing process setup – 30%;
- Testing process execution – 16%.

Finally, this activity took 31% of the total project resources. Testing expenses the application before it was delivered to the customer were 18%. Thus the total costs for testing were 49%. It is possible to reduce expenses significantly reusing UniTask in the same company and for the same project types: the economy would be achieved because there is no large-scale manual testing, less resources are spent for studying, and due to reuse of formal specifications and components of testing software and other related knowledge. We hope under these conditions the total test expenses would come up to 20%, but additional experiments should be performed to state for certain.

## 6 CONCLUSIONS

A lot of teams practically use the idea of this method, but it has not been defined and structured up to date and actually was used as a tacit

knowledge. It turns out to be a barrier for beginners to use model-based testing approach and hinders development of knowledge libraries – both inside of the companies and in the context model-based testing community.

This method good suit for lightweight development processes, with absent of strict process procedures and poor documentation support. But due to iterative nature of such processes, the method should be applied for the stable components or to the end of a project.

It is planned to research measures thoroughly necessary for successful reusing model-based testing knowledge within the company. It is also necessary to research various kinds of software from model-based testing perspective, in order to make explicit maximum of tacit knowledge.

We also plan to transfer the method to the model-checking approach, which is the most demanded software verification method. Expenses of developing formal models in the context of this approach is also a considerable barrier to its wide-spread practical use.

## REFERENCES

- Hinchey, M., et. al., 2008. Software engineering and formal methods. *Commun. ACM* 51(9): 54-59.
- Knight, J., 1998. Challenges in the Utilization of Formal Methods. *FTRTFT*: 1-17.
- Easterbrook, S., Callahan J., 1997. Formal Methods for Verification and Validation of partial specifications: A Case Study. // Virginia University symposium, Vol. 1. P. 26-37.
- Johnsen, E., Owe, O., 2002. Composition and Refinement for Partial Object Specifications. // Parallel and Distributed Processing Symposium. P. 210-217.
- Hendrix, J., Clavel, M., Meseguer, J., 2005. A Sufficient Completeness Reasoning Tool for Partial Specifications. // Proceedings of the 16h International Conference on Rewriting. LNCS, Vol. 3467. Springer. P. 165-174.
- Bourdonov, I., et. al., 2002. UniTesK Test Suite Architecture. *FME* 2002: 77-88.
- Kuliamin, V., et. al., 2003. The UniTesK Approach to Designing Test Suites. *Programming and Computer Software* 29(6): 310-322.
- Nonaka, I., 1991. The knowledge creating company. *Harvard Business Review* 69 (6 Nov-Dec): 96-104.
- Spender, J.-C., Andreas G.S., 2007. The Philosophical Foundations of Knowledge Management: Editors' Introduction. *Organization* 14 (1): 5-28.
- Engelhart, P.M., 2001. Knowledge Management in Software Engineering: a State-of-the-Art-Report. Air Force Research Laboratory Information Directorate/IFED. Rome, NY. 57 p.

- Utting, M., Legeard, B., 2007. Practical Model-Based Testing: A Tools Approach, Morgan-Kaufmann.
- Kuliamin, V., Pakulin, N., Petrenko, O., et. al., 2005. Requirement formalization in practice. Preprint of RAS (In Russian).
- Liu, K., 2005. Requirements Reengineering from Legacy Information Systems Using Semiotic Techniques, Systems, Signs and Actions - An International Journal on Communication, Information Technology and Work, 1(1): 36-61.
- El-Ramly, M., Stroulia, E., Sorenson, P., 2002. Recovering software requirements from system-user interaction traces. SEKE. P. 447-454.
- Mens, T., Demeyer, S., 2008. Software Evolution. Springer.
- Jansen, A., Bosch, J., Avgeriou, P., 2008. Documenting after the fact: Recovering architectural design decisions. Journal of Systems and Software archive, V. 81, Issue 4, P. 536-557.
- Koznov, D., Romanovsky, K., Nikitin, A., 2001. A Method for Recovery and Maintenance of Software Architecture. Ershov Memorial Conference: 324-327
- Letichevsky, A.A., Kapitonova, J., 2004. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. // Proceedings of International Work-shop, WITUL. P. 30-38.
- Falcone, Y., Fernandez, J., Mounier, L., et. al., 2007. A Compositional Testing Framework Driven by Partial Specifications. // TestCom FATES. P. 107-122.
- Petrenko, A., Yevtushenko, N., 2005. Testing from Partial Deterministic FSM Specifications. // IEEE Trans. Computers, vol.54, No. 9. P. 1154-1165.
- Acharya, M., Xie, T., Pei, J., 2007. Mining API patterns as partial orders from source code: from usage scenarios to specifications. // Proceedings of SIGSOFT Seminar. P. 25-34.
- Tichomirov, V., Kotlyrov, V., 2008. An approach of integration testing. // Systemnoe programirovanie. Saint-Petersburg State University. Vol. 3. P. 109-120. (In Russian).