Test-Driven Conceptual Modeling: A Method and a Tool

Albert Tort

Universitat Politècnica de Catalunya
atort@lsi.upc.edu

Abstract. Conceptual Schema-Centric Development (CSCD) has been qualified as a grand challenge for many researchers in Information Systems (IS). It reformulates the historical goal of automating IS building but emphasizing that the Conceptual Schema (CS) should be the center of the development. CSCD requires explicit, complete, executable and correct CSs. In this context, testing becomes critical in the conceptual modeling activity. Test-Driven Conceptual Modeling (TDCM) is proposed as a new development method to obtain a correct and complete CS. TDCM applies to conceptual modeling the essential ideas of Test-Driven Development(TDD), an emerging test-first eXtreme Programming (XP) method. Our research proposal pretends to contribute to the IS research community by developing the TDCM method and a tool to put it into practice.

1 The research topic

1.1 Conceptual Schema-Centric Development

Conceptual Schema-Centric Development (CSCD) [20] reformulates the historical goal of automating Information Systems (IS) building. CSCD emphasizes that the system's conceptual schema should be the center of the development. In this approach, the Conceptual Schema (CS) becomes the only external description to be defined. It can be executed in the production environment by using a virtual machine or by an automatic translation into software components. To achieve this goal, CSCD requires CSs to be explicit (written in a formal modeling language), complete (all the general static and dynamic knowledge about the domain needed by the IS to perform its functions is specified), executable and correct. eXtreme Conceptual Modeling (XCM)[11] is a similar approach.

1.2 Test-Driven Development

Test-Driven Development (TDD) [1, 3, 12, 13] is an emerging eXtreme Programming (XP) [4] practice. From the IS point of view, TDD is a development method in which an IS is implemented in short iterations in each of which the programmer first writes a test. Then, the knowledge acquired by writing the test guides the changes to be done in the code to pass it. Finally the code is refactored. TDD tests are automatically executed after every change. Constant test feedback allows detecting errors in the code as soon as a new change causes them.

1.3 Test-Driven Conceptual Modeling

We propose a new method for obtaining a CS by introducing the TDD approach to the field of conceptual modeling. We name it Test-Driven Conceptual Modeling (TDCM). TDCM pursues ensuring the correctness and completeness of the resultant CS. A CS obtained by applying TDCM can be used in conjunction with CSCD or as a solid base for design and implementation phases.

TDCM is a development method in which a system's CS is obtained by performing three kinds of tasks:

- Write a test the CS should pass.
- Change the schema to pass a test.
- Refactor the schema to improve its qualities.

These tasks are performed in short iterations. In each TDCM iteration, developers first write a test. A test is, in fact, an executable form of a functional requirement validation criterion in a concrete scenario. Martin et al. state that "you can specify system behavior by writing tests and then verify that behavior by executing the tests" by using the Möbius strip approach [15]. The task of writing a TDCM test provides a way of thinking what changes in the schema should be done to pass it.

Tests are preserved and automatically executed after every change in the CS. In this way, TDCM helps ensuring that once each iteration finishes, the CS satisfies the already processed requirements. Otherwise, errors caused by new knowledge added to the CS are detected immediately.

Finally, refactoring is applied to improve the quality of the CS without changing the knowledge specified in it. Automatic tests execution checks that the knowledge represented in the CS is not affected by refactoring.

2 Expected contributions

This PhD research proposal pretends to make the following contributions:

- A method for the development of conceptual schemas using the TDCM approach.
- A TDCM supporting tool to allow the use of the method in practice.

In the programming field we can find experimental evaluations of TDD both in academic [17, 21] and industry contexts [5, 7, 8, 16]. The conclusions reached by these experiments are not coincident in all cases. The variety of experiment designs explain differences and encourage more experimental evaluations to be definitely conclusive about TDD benefits. Nevertheless, these TDD observations in practice drive most of these researchers to claim TDD benefits such as more testing deep, quality improvement and developers confidence.

Our method pretends to make possible the use of the essential TDD techniques in a conceptual modeling environment working with UML/OCL CSs. Once developed the TDCM method and its supporting tool it will be possible to evaluate it. At this moment we consider a complete evaluation as further work.

3 TDCM today

As far as we know, the application of the TDD approach to the development of CSs has not been explored yet.

The number of CASE tools that help specifying CSs has increased in the last years. The most well-known commercial CASE tools (Poseidon, Magic Draw, Rational Rose, etc.) help drawing schemas but they offer rather limited verification and code-generation functionalities, specially for OCL expressions.

In the testing side, several researchers have proposed different techniques in the context of Model-Driven Development (MDD) to introduce approaches for testing models. The MODEST method [18] proposed by Santos Neto et al., the Test-Driven Modeling (TDM) approach [24] by Zhang, the TOTEM approach [6] by Briand and Labiche and the UMLAnT Eclipse plugin [22, 23] presented by the Colorado State University are examples of these efforts mainly focused on automatically deriving some tests from design models.

In the programming context, we find JUnit [14] as an important reference tool which provides an easy-use framework to support the TDD method for Java systems development.

Other tools like USE [10], can be considered precursors of CSCD tools in the academic context. USE helps in the validation of UML/OCL schemas [9] by creating possible system states (called snapshots) and checking whether those snapshots are valid instantiations of the schema. It also offers some functionalities to query the model and executing transformations on it. However, USE does not provide support on automated collections of tests.

None of these tools satisfy completely the requirements demanded by TDCM.

4 The research approach

In order to achieve the proposed contributions we plan structuring our research in the following main stages:

– *Definition of the TDCM purpose*: In this stage, we define the objectives of the method (what is the problem and why we need a new method to solve it?), the properties of its results (the desired output to be obtained) and the context of its application.

– *Method requirements elicitation*: Once defined the method strategy, we can identify the method requirements, taking into account the TDCM purpose and the experiences about related methods and techniques.

– *Method formalization*: We propose the elaboration of a metamodel to formally define TCDM static concepts and the process guidelines to apply it. This task requires clarifying concepts and name them. We will reuse as much as possible the general UTP definitions about testing [2]. We need to answer questions like the following: What are CS tests? In what language (an already existing or a new specific one) is more appropriate to write TDCM tests? What is the optimal order to test the requirements?

– *Development of a TDCM tool*: The requirements elicitation for the TDCM tool should contemplate the necessity of considering those which are not directly associated to the theoretical method but they are necessary to ensure the viability of its application. Some of the requirements are mandated (necessary to put the method into practice). Others are optional. The implementation should be planned taking into account the requirements priorization. We should also consider the exploration of already existing modular and extensible platforms which could be the base for our tool implementation.
– *Elaboration of a refactoring patterns catalog*: The study of criteria to evaluate the quality of CSs and patterns to improve it is a parallel research line to be considered. Some of these refactoring patterns will be inspired in those already used in programming or ontologies. Others will be specific to TDCM.

5 Results achieved so far

During our initial research stage, our main efforts have been focused on the definition of the TDCM purpose, the context of its application, the study of concepts associated to it and the elicitation of the method requirements. We have also studied the adaptability of TDD practices to the aim of defining a system's conceptual schema and the role of TDCM in the evolution of ISs.

The following are some of the main requirements already discussed. They are also useful to complement the description of TDCM.

5.1 The starting point

Conceptual modelers usually obtain, during the requirements engineering phase, different specifications. The most common are the domain CS (a draft model of knowledge about the domain) and the use case specification, which can be obtained following different techniques depending on each project. Usually, these specifications are not complete and cannot be used in CSCD environments. From these preliminary specifications, TDCM guides the obtention of a complete system's CS. This context of application is the TDCM starting point.

5.2 "Silent" automatic execution of tests

Another essential requirement to make useful the method-in-practice is the automation of tests execution after every change. We have concluded the necessity of providing a "silent" automatic execution of tests by requiring the attention of developers only when a test fails.

5.3 Reducing testing effort

There are also optional requirements to improve the usability of the TDCM tool. We propose test coverage analysis functionalities, warnings about the coherence between the behavioral and the static part of the CS [19] and automatic assistance (based on test patterns) for reducing the test design effort (by inferencing some conceptual schema changes from tests and vice versa).

References

1. Astels, D.: Test driven development: A practical guide. Prentice Hall (2003)
2. Baker, P., Dai, Z. R., Grabowski, J. et al.: Model-driven testing: Using the UML testing profile. Springer (2008)
3. Beck, K.: Test-driven development: By example. Addison-Wesley Prof. (2003)
4. Beck, K., Beedle, M., van Bennekum, A. et al.: Manifesto for Agile Software Development (2001) at: `http://agilemanifesto.org`
5. Bhat, T., Nagappan, N.: Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. Proc. ACM/IEEE 2006 International symposium on empirical software engineering (2006) 356-363
6. Briand, L., Labiche, Y.: A UML-Based Approach to System Testing. Proc. UML 2001, 4th International Conference on UML, Toronto, Canada (2001)
7. Canfora, G., Cimitile, A., Garcia, F. et al.: Evaluating Advantages of Test Driven Development: A Controlled Experiment with Professionals. Proc. 2006 ACM/IEEE International symposium on empirical software engineering, (2006) 364-371
8. George, B., Williams, L.: An Initial Investigation of Test Driven Development in Industry. Proc. 2003 ACM symposium on Applied computing (2003) 1135-1139
9. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL Models in USE by Automatic Snapshot Generation. Software and Systems Modeling, 4 (2005) 386-398
10. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming (2007)
11. Insfrán, E., Pelechano, V., Pastor, O.: Conceptual Modeling in the eXtreme. Information and Software Technology, 44 (2002) 659-669
12. Janzen, D., Saiedian, H.: Does Test-Driven Development really Improve Software Design Quality? Software, IEEE, 25 (2008) 77-84
13. Janzen, D., Saiedian, H.: Test-Driven Development Concepts, Taxonomy, and Future Direction. Computer, 38 (2005) 43-50
14. JUnit website at: `www.junit.org`
15. Martin, R. C., Melnik, G., Inc, O. M.: Tests and Requirements, Requirements and Tests: A Möbius Strip. Software, IEEE, 25 (2008) 54-59
16. Maximilien, E. M., Williams, L.: Assessing Test-Driven Development at IBM. Proc. 25th International Conference on Software Engineering, (2003) 564-569
17. Muller, M., Hagner, O.: Experiment about Test-First Programming. Software, IEE Proceedings, 149 (2002) 131-136
18. Neto, P. S., Resende, R., Padua, C.: A Method for Information Systems Testing Automation. LNCS 3520 (2005) 504-518
19. Olivé, A.: Conceptual modeling of information systems. Springer (2007)
20. Olivé, A.: Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. Advanced Information Systems Engineering: Proc. 17th International Conference, CAiSE 2005, Porto, Portugal (2005)
21. Pancur, M., Ciglaric, M., Trampus, M. et al.: Towards Empirical Evaluation of Test-Driven Development in a University Environment. EUROCON 2003.Computer as a Tool.The IEEE Region 8, 2 (2003)
22. Pilskalns, O., Andrews, A., Knight, A. et al.: Testing UML Designs. Information and Software Technology, 49 (2007) 892-912
23. Trong, T. D., Ghosh, S., France, R. B. et al.: UMLAnT: An Eclipse Plugin for Animating and Testing UML Designs. Proc. 2005 OOPSLA workshop on Eclipse technology eXchange (2005) 120-124
24. Zhang, Y.: Test-Driven Modeling for Model-Driven Development. Software, IEEE, 21 (2004) 80-86