FORMAL METHODS: FOR ALL OR FOR CHOSEN?

Victor V. Kuliamin, Vitaliy A. Omelchenko
Institute for System Programming, Russian Academy of Sciences (ISPRAS), Solzhenitsyna, 25, Moscow, Russia

Olga L. Petrenko
Moscow Institute of Open Education, Moscow, Russia

Keywords: Formal methods, Teaching formal methods, Active learning, Cooperative learning, Critical thinking pedagogical framework.

Abstract: The article presents an approach to teaching formal methods that may make them accessible for ordinary software engineers, especially those who are not skilled in the underlying mathematics. The approach is based on two ideas. First, we propose modification of course contents to hide the underlying mathematical techniques under some terms and actions familiar to the engineer or student learning the corresponding application domain. This usually requires availability of mature tools supporting formal methods under consideration. Second, we modify the presentation of course material and focuse mostly on active learning and more deep students' involvement in the learning process. This approach was successfully used in traditional courses and trainings in UniTESK, a test development technology based on formal specifications.

1 INTRODUCTION

We are witnessing a strange trend in software development. Modern development techniques and tools make possible development of extremely complex software systems. At the same time the quality of software is perceived mostly as poor. Even without taking into account the critical software working in avionics, healthcare, or infrastructure systems, where single failure can lead to loss of human life or catastrophic losses for a country, the economic impact of software faults reaches the dozens of billions dollars per year only in USA economy (Tassey, 2002).

It seems that the application of traditional software development 'best practices' could not provide the proper way to manage the sky-rocketing software complexity. To attain the acceptable level of software quality more systematic and rigorous approaches should be applied in industrial software development. The only visible candidates on this role are formal methods (FM).

Formal methods have been extensively developed over many years. From original toy examples (Floyd, 1967; Dijkstra, 1974) they evolve into the wide variety of methods supported by tools. They have been successfully applied in industrial software projects. One of the largest such projects was the development of a controller system for the Parisian driverless metro Meteor by MATRA based on B specifications and theorem proving with further code generation (Abrial, 1996). In some domains formal notations and methods based on them became traditional, e.g. SDL and TTCN with techniques based on them in the telecommunication software development.

Nevertheless, formal methods are still not widely used in the software industry. The most commonly referred reason for this situation is that formal methods require use of difficult mathematics and correspondingly skilled staff. While the numbers software projects and of software developers involved grow each year with increasing speed, the number of people who can adequately operate the techniques underlying most formal methods grows much slowly. Although a lot of courses in FM are conducted in universities, the effort needed to prepare a professional in formal methods is huge and not all the people who can be successfully used as software engineers can be trained in formal methods with the same success.

So, for successful introduction of formal methods the industry requires now so many FM specialists that

V. Kuliamin V., A. Omelchenko V. and L. Petrenko O. (2009).
FORMAL METHODS: FOR ALL OR FOR CHOSEN?.
In *Proceedings of the First International Conference on Computer Supported Education*, pages 217-222
DOI: 10.5220/0002153902170222

current education system cannot produce. One can try to solve this problem by increasing labor effectiveness, polishing up existing techniques and professional skills, but in this article we propose to increase number of FM specialists using innovative education technologies. We defend the point that the problem of deep involvement of mathematics in teaching formal methods can be overcome. The idea is to put most of the burden of formal manipulation with symbols on tools, while giving people more skills of whole picture comprehension along with abilities to find key issues in the problem domain.

To do this we should revise the courses used and try to make them more comprehensible and useful for people with no direct aptitude in mathematics by putting focus on key issues in practically successful formal techniques. This revision should involve both the contents of FM courses and maybe the user interface of the tools used and the course presentation and techniques used in education process.

2 COURSE ADAPTATION PROBLEMS

We suppose that there is no universal way to teach any student any course. So, every course needs some adaptation to the audience, in particular, whether it consists of mathematically talented people or not. This adaptation should not be based only on interests of students, because it usually leads to decrease of education level.

Here we consider adaptation of FM courses for people that have no mathematical aptitude. On the base of our experience we suggest two ways of course adaptation.

- Modification of course contents. This possibility is concerned with user interface of the formal method under consideration, so we further call it ‘turning to the user’.
- Modification of course presentation, its organization and teaching methods used. This possibility is concerned with involvement of students into education process, transformation of this process into active learning process, so we further call it ‘turning to the student’.

3 TURNING TO THE USER

The comprehension of a course depends on many factors. Since formal methods are not a fundamental science (it is mathematics in this case), the formal methods course contents can vary in wide range depending on the students’ background and aptitude. The contents depend on the number of hours we have and position of the course in the curriculum. We can try to escape the predominance of mathematical notation and techniques by means of course contents modification.

This approach is closely related with user interface of the corresponding tools. For tool users it means that the representation of a method is simplified and its mathematical background is hidden under some simple operations, which can be performed automatically in most cases or require user to answer some clear questions. For FM students this means that the contents of the course can be changed to focus on practical aspects of using a method with the help of simple terms and tools adequately supporting the method and the underlying mathematical techniques can be leaved out of scope or moved to advanced part of the course.

To see whether it is possible to decrease the level of mathematics involved in courses in formal methods we have compared several of them.

1. Modeling and Analysis of Complex Systems.
   The course AA244 (ASM Course, 2001) is conducted for 4-th year students of Department of Computer Science, University of Pisa, Italy. It is developed by E. Borger and consists of theoretical lectures covering the following topics.
   (a) Current state of software engineering. Abstract State Machines as a ground foundation for rigorous system engineering.
   (b) Definition of ASM. ASM notation. ASM semantics. Multi-agent and asynchronous ASMs. Models of concurrency. ASM as universal algorithmic models.
   (c) Capturing requirements in ASM. Simulation and reasoning on ASM models. Stepwise refinement method.
2. Modeling and Computation.
   The course CSC264 (VDM Course, 2003) is conducted for 2-nd year students of School of Computing Science, University of Newcastle upon Tyne, United Kingdom. Its authors are J. S. Fitzgerald, M. S. Kouthy, and S. Riddle. The course includes theoretical and practical parts and covers the following topics.
   (a) Software development and software lifecycle. Challenges in software development.
   (b) Formal models of software. Their characteristics and impact on the development activities.
   (c) Modeling techniques. Formal specification languages.

(d) Vienna Development Method. VDM constructs. VDM types. Recursive data types. State based modeling.

(e) Validation. Model consistency. Animating, testing, and proving of models. Logical frameworks. Propositional logic. Calculus of sequents. Logic of partial functions. Reasoning about VDM models.

(f) Concurrency models. Models of computation. Finite state automata.

3. Formal Specifications of Software.
The course (FM Course, 2008) is developed by A. Petrenko and is conducted for 4-th year students of Computing Mathematics and Cybernetics Faculty of Moscow State University, Russia. The course consists of theoretical part and practical studies covering the following topics.

(a) Software lifecycle, its main phases and activities. Software requirements. Software quality.

(b) Formal methods in software development. Formal specifications and their impact on different activities. Specification techniques. Refinement and abstraction.

(c) RAISE method. Iterative refinement. RSL language. RSL notation. RSL types. Variant types. RSL expressions. Parallel computation.

(d) Analytical verification of software. Hoare logic. Floyd methods.

(e) Testing based on formal specifications. Test oracle. Test adequacy criteria. Partitioning of operation domain. Test sequence construction on the base of automata models.

(f) Formal specifications of programming languages. Syntax, static and dynamic semantics. BNF. Attribute grammars. Correctness checking. Code generation. Methods of dynamic semantics description. Operational semantics.

4. Formal Software Specification Using RAISE.
The course (RAISE Course, 2003) is conducted for graduates in computer science or other computing-related discipline in International Institute for Software Technology, United Nations University, Macau. It is developed by C. George and A. E. Haxthausen. The course includes both theoretical and practical parts and covers the following.

(a) Software lifecycle. Formal methods in general. Characteristics of formal methods. Formal specifications.

(b) RAISE method. Iterative refinement and verification. Specification techniques.

(c) RSL notation. RSL types. Variant types. RSL expressions. Channels and concurrency. Modularity.

This helps to find common part of most such courses and gives ideas to lighten the mathematics involved. The contents of all four these courses are based of the following common scheme.

1. To make software development less error-prone and the resulting software more manageable and comprehensible we can use *formal models* of software under development or analysis. They help to explore the properties of actual software or to construct it in such a way that it would have the wishful properties.

2. Formal models are developed in a specialized *formal frameworks* providing a language to describe such models and methods of deducing or checking models' properties, or checking their counterparts (whatever it means) in actual software. These are the methods usually called *formal methods.*

3. *The iterative refinement* of models is used to obtain the models more and more close to the actual software or to the software we need and at the same time retaining the properties of the previously constructed models.

This general scheme is illustrated on Fig. 1.

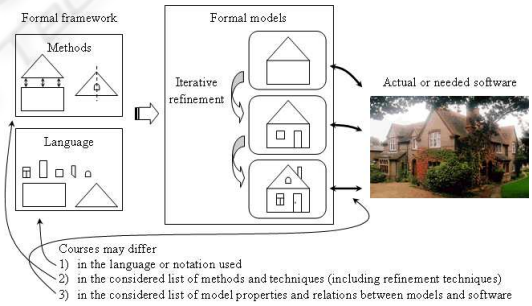

Figure 1: Common contents of courses in formal methods.

The formal frameworks considered in different courses differ – in the first one it is Abstract State Machines (ASMs) (Borger and Stark, 2003), in the second one it is Vienna Development Method (Bjorner, 1979; Fitzgerald and Larsen, 1998), in the third and the fourth it is RAISE Method (RAISEGroup, 1995). But the common scheme found gives us some hints on how to modify its contents to make it more comprehensible for people not skilled in mathematics. We notice that most elaborated mathematics is involved in formal frameworks used. Usually they are based on formal languages with specific notation and include methods to manipulate formal constructions that look

terrible for non-mathematicians. And the use of mathematics in work with formal models mostly depends on the language and methods used. So, we have two possibilities concerning with those issues in the inverse order.

First, we may make mathematical background of a formal method as stable as possible and hide it inside the tools supporting this method. The interface of the tools should be as close to the 'common life' as possible. Users would work with it by manipulating the familiar concepts and their relations, which are unambiguously mapped into formal framework and so correspond to formal concepts with relations between them. This solution can be applied if we perform refinements rarely and are interested in methods that allow mostly automatic operation with models and their properties. Examples of such methods are model checking and formal testing.

Second, we may try to 'decorate' formal framework with terms of common language in such a way that makes its theorems and techniques more clear to non-mathematician. Of course, such a decoration should be performed in a very accurate way, different for different problem domains, to eliminate problems of misunderstanding of terms and at the same time to provide these terms with precise meaning. This approach may simplify techniques used for proving of software properties.

4 TURNING TO THE STUDENT

The other direction of course adaptation is to create an effective learning environment different from the traditional university education system based on pure theoretical lectures and practical studies. While the traditional education process is based mostly on students' attention, perception, and memorization of the information ('the school of memory'), some modern education methods involve creative, dialogical thinking and social activity in the learning process ('the school of thinking'). Advanced learning techniques use the ideas of cooperation, dialog, and partnership between students and teacher.

This cooperation can be realized in the process of active learning, which requires teacher to change his attitude to the education. Good results can be achieved if the teacher realizes that student is the focal person of the process and that the main activity is learning, not teaching. The priorities are moved to independent learning and practical application of knowledge obtained in particular.

In active learning students are involved in active cognitive work on all the phases of education process. They independently discover the knowledge, which in traditional approach is presented by the teacher without active participation of the students themselves (Smirnov, 2001). It is well known that when active learning is used from 75% to 90% of the information given by the teacher is acquired. Compare this data with the corresponding data for traditional lecture, which helps to acquire not more than 20% of the information introduced in it (Hartley and Davies, 1978).

Since the university education is organized as a number of lectures and practical studies, we use the same organizational structure. Practical studies give more possibilities for student involvement in active learning process and can be conducted with use of cooperative learning techniques (Johnson et al., 1990), projects, multi-level education. We also use the following techniques in presentation of the course contents.

1. Work with special workbook (portfolio) for independent tracking of subject mastering and formulation of questions. This technique is used at the lectures as a form of the students' individual work organization.
2. Organization of cooperative work in groups during lectures. It is used to discuss the stated problems, to make decisions, and to find unclear issues by the end of the day (Rae, 2000).
3. Organization of pair work on lectures with individual tasks for students in a pair, for example, one student tries to explain the material to the other or one asks questions and the other answers them.
4. Individual system of practical studies that allows every student to go along his own way in mastering practical skills.
5. Use of anticipatory exercises during lectures, when students are asked to find a solution of some problem with the help of individual work or discussions groups before the knowledge on the given topic are introduced to them. Then a joint discussion of the problem is organized and new knowledge on the given topic is presented. After that the problem discussion and its results are analyzed.
6. Use of one of the basic models of active learning – Evocation-Realization of Meaning-Reflection (Meredith et al., 1997).
   Evocation is a process of actualization of students' knowledge on a given topics and their motivating to investigate problems arisen. Challenge prepares students for and sets their learning process on the information that will be introduced

later. During a lecture challenge can be represented for example as the task to draw the cognitive cluster of some concept or problem.
Realization of Meaning is an acquisition of new information or ideas. This phase implies introduction of new information. The realization of anything new can take place only during active learning activities. So, the special conditions should be created to stimulate active involvement of a student into the process of mastering new information.
Reflection is an introduction of new knowledge into one's own knowledge system.

To stimulate active involvement of a student into learning process we suggest not use traditional synopses writing on lectures. Instead we can use the 'log-book' technique for memorizing information presented on lecture. The 'log-book' consists of the following parts.

- keywords;
- questions, problems;
- diagrams, charts, drawings, models;
- associations, applications.

The lecture stops after 12 minutes since lecture started and asks students to fill the sections of the 'log-book'. They used about 3-4 minutes for that, then 5-6 minutes for discussion of the results. After that the lecture proceeds.

The other technique that can be used is presentation of a lecture with the help of marking table. The text of lecture is distributed among students and they are asked to mark it with the following marks.

$\surd$	–	I already know this
+	–	this is new for me
-	–	this contradicts to my knowledge
?	–	here I have some questions

After marking the text each student fills the table consisting of four columns corresponding to those marks. Then, the discussion of the questions and contradictions is organized in pairs and in groups. The lecturer sometimes takes the initiative and explains the necessary issues.

Such an organization of learning process helps to each student to move along his own path, his own learning trajectory. The important issue in the development of student's own knowledge system is reflection, which is underestimated in teaching formal methods, since reflection is usually related with the humanities. But the only reflection helps one to comprehend his activity in such a way that he is definitely aware of what and how he does. In other words, one is aware of plan and rules of his own activity. The purport of reflection as a learning activity is the refinement of one's knowledge, uncovering of its foundations and evolution process.

Other ways of lecture organization that helps to construct individual learning trajectory are also possible.

5 CONCLUSIONS

The article presents an approach to teaching students with no aptitude in mathematics formal methods of software development. To have such an approach is very important in the contemporary world where the most part of critical infrastructure uses software in some form and the software becomes extremely complex and at the same time needs higher level of reliability.

The approach presented consists of the following two parts.

1. Course contents can be modified to hide the mathematical formalism often put ahead as a main essence of formal methods under a set of commonly used terms and more usable interface of the tools with several 'magic buttons'.
2. The learning process can be made more effective by use of modern education techniques aimed to involve as much capabilities and attention of students as possible, to stimulate them to active learning and independent knowledge acquisition.

Members of RedVerst group of ISPRAS successfully apply this approach in course on formal specification of software conducted in Moscow State University. It was also used in several training course conducted for industrial software development teams to transfer UniTesK test development technology base on formal specifications (Petrenko and Omelchenko, 2003; Kuliamin et al., 2005).

REFERENCES

Abrial, J. R. (1996). *The B Book*. Cambridge University Press.

ASM Course (2001). *http://www.di.unipi.it/∼boerger/LC01.html*.

Bjorner, D. (1979). *The Vienna Development Method: Software Abstraction and Program Synthesis*. Springer-Verlag.

Borger, E. and Stark, R. (2003). *Abstract State Machines: a Method for High-Level System Design and Analysis*. Springer-Verlag.

Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *Am. Math. Monthly*, 81(6):608–612.

Fitzgerald, J. and Larsen, P. G. (1998). *Modelling Systems: Practical Tools and Techniques for Software Development*. Cambridge University Press.

Floyd, R. W. (1967). Assigning meanings to programs. In *Proc. Symposium on Applied Mathematics*, pages 19–32. American Mathematical Society.

FM Course (2008). *http://www.ispras.ru/∼RedVerst/ RedVerst/Lectures and training courses/MSU course Formal specification of software/RMain.html*.

Hartley, J. and Davies, I. K. (1978). Note-taking: A critical review. *Programmed Learning and Educational Technology*, (15):207–224.

Johnson, R. T., Johnson, D. W., and Smith, K. A. (1990). Cooperative learning: An active learning strategy for the college classroom. *Baylor Educator*, pages 11–16.

Kuliamin, V. V., Omelchenko, V. A., and Petrenko, O. L. (2005). Learning advanced software development methods: problems and solutions. *Proc. of ISPRAS*, pages 91–108.

Meredith, K., Steele, J., and Temple, C., editors (1997). *Guidebooks for Reading and Writing for Critical Thinking*. The International Reading Association.

Petrenko, O. L. and Omelchenko, V. A. (2003). Rapid training on specification based testing tools. In *Proc. SEEFM'2003*.

Rae, L., editor (2000). *Using Activities in Training and Development*. Kogan Page.

RAISE Course (2003). *http://www.iist.unu.edu/home/ Unuiist/newrh/II/2/1/1/page.html*.

RAISEGroup (1995). *The RAISE Development Method*. Prentice Hall.

Smirnov, S. A., editor (2001). *Pedagogics: Pedagogic Theories, Systems, and Technology*. Publishing Center "Academia", 4-th edition.

Tassey, G., editor (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing, Final Report*. National Institute of Standards and Technology Acquisition and Assistance Division.

VDM Course (2003). *http://coursework.cs.ncl.ac.uk/ module/2003/CSC264*.