Teaching of Formal Methods for Software Engineering

Maria Spichkova[1] and Anna Zamansky[2]

[1]*School of Science, RMIT University, 414-418 Swanston Street, 3001, Melbourne, Australia*

[2]*Information Systems Department, University of Haifa, Carmel Mountain, 31905, Haifa, Israel*

Keywords: Formal Modelling, Collaboration, Teaching.

Abstract: The use of Formal Methods (FMs) offers rigour and precision, while reducing ambiguity and inconsistency. The major barriers hindering the adoption of FMs in industry are the problems of understandability, comprehensibility, and scalability. To solve the understandability problem, from one side, the readability of the method have to be increased, but from another side, an appropriate teaching and learning approach have to be introduced. This paper presents an overview of existing approaches on teaching of FMs and Logic, also discusses the common issues in teaching of this subjects.

1 INTRODUCTION

For the development of embedded real-time systems in most cases experts of different disciplines have to cooperate, and for such a cooperation a specification of the developing system, i.e., precise and detailed description of its behaviour and/or structure, is important. Embedded systems are real, but their behaviours are modelled by mathematical objects, about which one can argue formally. One aim of Formal Methods (FMs) is to prove or to automatically evaluate behaviour properties of a system in a systematic way, based on a clear mathematical theory.

When dealing with formal methods, we are mainly concerned with the methods's soundness and correctness, and sometimes also its mathematical elegance, but usually do not take into account such aspects as readability, usability, or tool support. This leads to the fact that FMs are perceived by most engineers as "something that is theoretically important but practically too hard to understand and to use". (Crocker, 2006) even suggests to replace the somewhat "unattractive" name 'formal methods' with 'verified software development'. We might have here a phenomena similar to the research field of Artificial Intelligence: deeply appreciated at the beginning of Artificial Intelligence discipline, it was seen as almost useless in 70s, cf. (Crevier, 1993), but was "reborn" with the paradigms of 'expert systems' and 'intelligent agents', and 'data mining'.

Even small changes of a formal method can make it more understandable and usable for an average engineer. Moreover, human factors engineering need to be incorporated into the software development process, cf. (Spichkova et al., 2015a), but the starting point for an adaptation of FMs in practice is the education in these methods. In the last decade, a number of teaching programs has been initiated to solve this problem. For example, the *Top SE* program in Japan was introduced with the aim to produce "superarchitects" who can promote practical use of advanced, scientific methods and tools, including formal methods, for tackling problems in software engineering, cf. (Ishikawa et al., 2009)

In this paper, we discuss the common issues in teaching of FMs and logic, as well as review the various approaches for teaching Formal Methods for Software Engineering that have been proposed, and discuss how they address the above mentioned challenges. The focus of our analysis here is on *collaborative* and *communication* aspects of software development using formal methods and logical modelling.

2 TEACHING FORMAL METHODS: CHALLENGES

The discourse on what to teach in Formal Methods and how to teach it has been going on for decades. It seems to be widely agreed that Formal Methods educators face the following challenges:

- There is a great diversity in the students' background and cognitive skills due to the globalisation of higher education, which requires constant adaptation, cf. (Hoare, 2013; Feast and Bretag, 2005).

- Students have decreasing mathematical background as the curricula become more practice-oriented, leaving less room for theoretical courses, cf. (Bjørner and Havelund, 2014; Crocker, 2006; Zamansky and Farchi, 2015a).
- Students have less motivation as they are strongly focused on the *direct* relevance of what they study to their daily practice, often failing to see the link of Formal Methods to the real world, cf. (Tavolato and Vogt, 2012; Wing, 2000).

2.1 Cultural Background

As per UNESCO statistics[1], the number of students who have crossed a national border to study, or are enrolled in a distance learning programme abroad, grows around the world. To denote this group of students, UNESCO introduced a new term – *internationally mobile students* (IMSs). In 2012, at least 4 million students went abroad to study, up from 2 million in 2000, representing 1.8% of all tertiary enrolments or 2 in 100 students globally. According to this statistics, 5 destination countries hosted nearly one-half of total IMSs: the United States (hosting 18%), United Kingdom (11%), France (7%), Australia (6%), and Germany (5%).

Internationalization of the higher education has created the so-called *borderless university*, which provides better opportunities for learning and increases the human and social sustainability, cf. (Woodcraft, 2012; Vallance et al., 2011; Penzenstadler et al., 2012). An obstacle to successful transnational teaching and learning could be the diversity in cultural and technical/educational backgrounds of teachers and students (as well as among the students). For example, the learning style and perception of the plagiarism problems is different by students coming from Asian and European countries, cf. (Zobel and Hamilton, 2002). This diversity has to be taken into account while teaching and assessing the students.

A partial solution to this problem might be introduction of *active and inductive* learning in Software Engineering education process (Sedelmaier and Landes, 2015). In the traditional *deductive* teaching, the lecturer introduces general theoretical principles and mathematical models, proceeds with examples on the applications of these principles and models, and concludes with practical exercises. An alternative to the deductive teaching is an inductive teaching approach that includes a range of instructional methods, such as inquiry learning, problem-based learning, project-based learning, case-based teaching, discovery learning, and just-in-time teaching, cf. (Prince and Felder, 2006) for more details.

A recent work (Alharthi et al., 2015; Spichkova and Schmidt, 2015) presents an approach on requirements specification and analysis for eLearning systems and for the geographically distributed software and systems. The eLearning systems are usually developed to use within different organisations or even different countries. The organisation/country-specific requirements can differ in each particular case because of technical, cultural, or legal diversity. The challenge is to deal with this diversity in a systematic way, avoiding contradictions and non-compliance.

2.2 Mathematical Background

The fundamental role of Logic and FMs is recognised in the ACM CS (Sahami et al., 2011) and SE (LeBlanc et al., 2006) undergraduate curriculum guidelines.

The pioneer of SE, David Parnas, believes that a solid understanding of logic is essential for a software engineer: *"Professional engineers can often be distinguished from other designers by the engineers ability to use mathematical models to describe and analyze their products."*, cf. (Parnas, 1993). Nevertheless, he also suggested that software engineers have to have engineers as role model, not philosophers or logicians, to have more practical view on mathematics, FMs and their application within software development, cf. (Parnas, 2010).

In his works, Parnas has noted the problem of understandability of formal specifications more than 20 years ago. To make the formal specifications more attractive for practitioners, Parnas suggested to make the formal expressions and system specifications shorter, changing their size and perceived complexity. In his recent works (Parnas, 2011; Jin and Parnas, 2010) on precise documentation and using formal methods for these purposes, Parnas highlighted that using tabular expression (tables) can increase the readability of formal methods. As per (Parnas, 2010), *"One of the most important roles that mathematics could play in software development would be to provide precise, provably complete, easy-to-use, testable documents. The popular formal methods have not been designed with use in documentation as the main goal."*

[1]http://www.uis.unesco.org/Education/Pages/international-student-flow-viz.aspx

In (Parnas, 2010), Parnas mentioned three alarming gaps, developed within last 50 years:

- the gap between formal methods research and practical software development is much larger today;
- the gap between software development and older engineering disciplines: where engineering programs teach how to apply mathematics to the engineering fields, most computer science departments teach abstract mathematics without learning them how to to apply;
- gap between computer science and classical mathematics.

From our point of view, closing these gaps should start on the level of university and higher school teaching.

There is also increasing concern that computer science and information technology curricula are drifting away from fundamental commitment to theoretical and mathematical ideas (Tucker et al., 2001). As (Bjørner and Havelund, 2014) points out, *"Todays masters in computing science and software engineering are not as well educated as were those of 30 years ago"*. Mandrioli (2015) further comments on a *"general tendency towards soften the teaching of engineering principles; this requires a certain amount of heroism to convince students that not everything can be obtained without effort"*, cf. (Mandrioli, 2015).

The problems are even more acute in universities of applied sciences (Tavolato and Vogt, 2012) and in the Information Systems discipline (Zamansky and Farchi, 2015a). The lack of empirical evidence that mathematical background is directly relevant for practitioners makes it even more difficult to convince decision makers that this situation must be handled. A notable example of such evidence is the Beseme project (Page, 2003): in a three-year study, empirical data on the achievements of two student populations was collected: those who studied discrete mathematics (including logic) through examples focused on reasoning about software, and those who studied the same subject illustrated with more traditional examples. An analysis of the data revealed significant differences in the programming effectiveness of these two populations in favour of the former.

In 2004, Symposium on Teaching Formal Methods was to explore the failures and successes of formal methods education, cf. (Dean and (editors), 2004). Now another decade is gone, but we are facing very similar problems: understandability and readability of FMs.

2.3 Lack of Motivation

Currently, FMs have very limited use in industrial software development process, which is a significant hurdle in making the based on FM courses attractive to the students. Woodcock at al. present survey of industrial use, comparing the situation in 2009 with the most significant previous surveys, and discuss the issues surrounding the industrial adoption of formal methods, cf. (Woodcock et al., 2009). Thus, we have a vicious cycle: To embed FMs into the software development lifecycle on industrial level, the FMs and the corresponding mathematical background have to be a part of the university curriculum. But the students are not motivated to learn FMs until they are not largely adopted by industry.

As FMs require a mathematical background and abstract thinking skills, many students have negative perceptions and even fear of courses that require dealing with complex mathematical notations. This is strongly related to the phenomenon of *mathematical anxiety*, cf. (Wang et al., 2014; Sherman and Wither, 2003). The term *mathematical anxiety* was introduced in 1972 by Richardson and Suinn as *"feelings of tension and anxiety that interfere with the manipulation of numbers and the solving of mathematical problems in a wide variety of ordinary life and academic situations,"* cf. (Richardson and Suinn, 1972). As stressed by Wang et al., mathematical anxiety has attracted recent attention because of its damaging psychological effects and potential associations with mathematical problem solving and achievement. Students of the Software engineering, Computer Science, and IT disciplines often prefer to attend the courses that do not require such a background.

3 APPROACHES FOR TEACHING OF FORMAL METHODS

Teaching Software Engineering courses is a difficult task, as it requires imparting abstract reasoning skills necessary for problem solving (Sprankle and Hubbard, 2011). One of the solutions to this problem would be embedding in the Software Engineering curriculum such subjects as logic and formal specification, because these subjects can provide a good starting point for exploring concrete ways in which we abstract thinking can be taught (Zamansky and Farchi, 2015a).

The approaches we discuss in Section 3.2 mostly focus on overcoming issues that comes from the diversity in mathematical background (or even lack of a solid mathematical background) as well as from the lack of motivation to study formal concepts. We do not attempt to cover the issues that comes from the diversity in the learning style and perception of the plagiarism problems, because these issues are not FMs specific.

3.1 Teaching Strategies

As pointed out by Ferreira et al., the success of teaching process depends on the amount of self-learning and self-discovery that is left for the students: *"If the teacher discloses all the information needed to solve a problem, students act only as spectators and become discouraged; if the teacher leaves all the work to the students, they may find the problem too difficult and become discouraged too. It is thus important to find a balance between these two extremes."*, cf. (Ferreira et al., 2009). MathIs project, presented by Ferreira et al., aimed to reinvigorate secondary-school mathematics by exploiting insights of the dynamics of algorithmic problem solving.

As suggested in (Wang and Yilmaz, 2006), the approaches in integrating FMs into software engineering curriculum can be divided into three main categories:

(1) to avoid FMs,

(2) to devote a specific course with emphasis on formal verification of source code;

(3) to redesign the entire program so that formal methods are integrated throughout the curriculum.

However, the work of Wang and Yilmaz does not mention another category, which can be very promising for integrating FMs into software engineering curriculum: to introduce a specific course that

- covers basics of logic and FMs,
- does not require a deep knowledge in mathematics, as only the core aspects of the FMs will be introduced,
- uses visualisation and ramification strategies to make the material more understandable and less "boring".

One examples of courses from the above category is the course *Applied Logic in Engineering*, or a "logic for everybody" course (Spichkova, 2016). Other examples include the *Logic and FM* course designed for Information Systems students (Zamansky and Farchi, 2015b), and a series of courses specifically adapted to the needs of university of applied sciences are described in (Tavolato and Vogt, 2012). Recently courses in the spirit of "computational thinking for everybody" envisioned by Wing (Wing, 2006) have begun to be offered at various departments, e.g., IS103 Computational Thinking course at the Singapore Management University and the COMP101 Computational Thinking and Design course at the University of Maryland.

Another way to attract students while teaching FMs was presented by Curzon and McOwan: Within the engagement project *cs4fn*, Computer Science for Fun[2], they taught logic and computing concepts using magic tricks, cf. (Curzon and McOwan, 2013).

There are also recent approaches on embedding the e-learning and blended learning strategies in teaching of mathematics and logic, cf. (Pokorny, 2012).

3.2 Visualisation and Tool Support

Visualisation tools using a notional machine have been used since the early 1970s to promote understanding of programming constructs (Mayer, 1975; Mayer, 1981).

The cognitive load can be reduced through visualisation of the learning tasks (Pane and Myers, 1996; Powers et al., 2007). Visualisation can help correct misconceptions as they commonly occur when outcomes are not readily visible, cf. (Sirkiä and Sorva, 2012). Advanced tasks could be designed to facilitate critical thinking by rewarding optimal or near optimal solutions as they cause students to reflect on their strategies. Moreover, publishing optimal benchmark figures for different configuration may promote greater interaction about possible strategies thus leading to a form of social constructivism that promotes higher levels of learning (Kozulin et al., 2003).

Vosinakis et al. introduced the MeLoISE platform (Meaningful Logical Interpretations of Simulated Environments) for teaching Logic Programming, cf. (Vosinakis et al., 2014). The platform developers focused on visualisation aspects, to allow the students experience a collaborative visual interface to the Prolog programming language.

AutoFocus[3] tool, cf. (Hölzl and Feilkas, 2010; Spichkova et al., 2012), was developed as a scientific prototype for formal modelling of distributed, timed, reactive systems. The implemented modelling language was based on a graphical notation that can be used to teach basic modelling constraints and the usability aspects (Spichkova et al., 2013).

Korecko et al. developed a toolset for support of teaching formal aspects of software development, cf. (Korecko et al., 2014). The toolset focuses on Petri nets and B-Method and visual representation of a train schedule example. This approach can stimulate abstract reasoning skills, but besides abstract rea-

[2]http://www.cs4fn.org/magic/

soning skills, students require a so-called *computational thinking* (Wing, 2006), a mental activity in formulating a problem to select a computational solution. The key attributes of computational thinking, as introduced by Wing, are (*i*) reformulating a complicated problem into a problem (or set of problems) which is already known and which we are able to solve; (*ii*) using abstraction and decomposition when analysing and decomposing a large complicated task or designing a complex system; and (*iii*) choosing an appropriate representation for the problem and/or modelling the relevant aspects of a problem to make it analysable.

A tool for *human-centred* model-based testing, which takes into account the human tester's possible mistakes and supports revision and refinement, was preposed in (Spichkova et al., 2015b). We believe that this tool may contribute to teaching of formal aspects of modelling and testing.

The KeY-Hoare tool, cf. (Bubel and Hähnle, 2008), was also developed to teach Hoare Logic. KeY-Hoare is based on a variant of Hoare logic with explicit state updates which allows one to reason about correctness of a program by means of symbolic forward execution.

Sznuk and Schubert developed a tool for teaching Hoare Logic, HAHA (Hoare Advanced Homework Assistant), cf. (Sznuk and Schubert, 2014). To estimate the impact that introduction of a tool has on the educational process, they used statistical methods of quantitative psychology (Trierweiler and Stricker, 1998). In contrast to KeY-Hoare, HAHA is based on classical Hoare logic. Classical Hoare logic requires backwards reasoning, which can be argued to be less natural and harder to learn.

Another tools used in educations were Why3 (Filliâtre and Paskevich, 2013), Dafny (Leino, 2010), as well as proof assistants Isabelle(Nipkow et al., 2002), and Coq proof (Henz and Hobor, 2011). Why3 is a tool for deductive program verification based on the WhyML language, which is an intermediate language in verifiers for Java, C, and Ada programming languages. Dafny can be used to verify functional correctness and termination of sequential, imperative programs.

4 CONCLUSIONS

Despite the impressive volume of work on teaching FM , this field still lacks systematisation; it is easy for educators to get lost in the "jungle" of the proposed methods and tools. This position paper presents our ongoing work in providing a "jungle map" to teaching FM, i.e., systematically reviewing the variety of approaches to teaching FM, taking into account the aims of the course, its target audience, its respective mathematical background and motivation.

ACKNOWLEDGEMENTS

The second author was supported by The Israel Science Foundation under grant agreement no. 817/15.

REFERENCES

Alharthi, A. D., Spichkova, M., and Hamilton, M. (2015). Requirements engineering aspects of elearning systems. In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, pages 132–133. ACM.

Bjørner, D. and Havelund, K. (2014). 40 years of formal methods. In *FM 2014: Formal Methods*, pages 42–61. Springer.

Bubel, R. and Hähnle, R. (2008). A hoare-style calculus with explicit state updates. In Instenes, Z., editor, *Proc. Formal Methods in Computer Science Education (FORMED)*, Electronic Notes in Theoretical Computer Science, pages 49–60. Elsevier.

Crevier, D. (1993). *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, Inc., New York, NY, USA.

Crocker, D. (2006). Teaching formal methods with perfect developer. *Teaching Formal Methods: Practice and Experience, Electronic Workshops in Computing*.

Curzon, P. and McOwan, P. W. (2013). Teaching formal methods using magic tricks. In *Fun with Formal Methods: Workshop at the 25th International Conference on Computer Aided Verification*.

Dean, C. N. and (editors), R. T. B., editors (2004). *Teaching Formal Methods: CoLogNET/FME Symposium, TFM 2004*. LNCS. Springer-Verlag.

Feast, V. and Bretag, T. (2005). Responding to crises in transnational education: new challenges for higher education. *Higher Education Research & Development*, 24(1):63–78.

Ferreira, J. a. F., Mendes, A., Backhouse, R., and Barbosa, L. S. (2009). Which mathematics for the information society? In *Proceedings of the 2Nd International Conference on Teaching Formal Methods*, TFM '09, pages 39–56, Berlin, Heidelberg. Springer-Verlag.

Filliâtre, J.-C. and Paskevich, A. (2013). Why3 — where programs meet provers. In Felleisen, M. and Gardner, P., editors, *Programming Languages and Systems: 22nd European Symposium on Programming*, pages 125–128. Springer Berlin Heidelberg, Berlin, Heidelberg.

Henz, M. and Hobor, A. (2011). Teaching experience: Logic and formal methods with coq. In Jouannaud, J.-P. and Shao, Z., editors, *Certified Programs and*

Proofs: First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings, pages 199–215. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hoare, L. (2013). Swimming in the deep end: transnational teaching as culture learning? *Higher Education Research & Development*, 32(4):561–574.

Hölzl, F. and Feilkas, M. (2010). Autofocus 3: a scientific tool prototype for model-based development of component-based, reactive, distributed systems. In *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, MBEERTS'10, pages 317–322.

Ishikawa, F., Taguchi, K., Yoshioka, N., and Honiden, S. (2009). What top-level software engineers tackle after learning formal methods: Experiences from the top se project. In *Proceedings of the 2Nd International Conference on Teaching Formal Methods*, TFM '09, pages 57–71. Springer-Verlag.

Jin, Y. and Parnas, D. L. (2010). Defining the meaning of tabular mathematical expressions. *Sci. Comput. Program.*, 75(11):980–1000.

Korecko, S., Sorad, J., Dudlakova, Z., and Sobota, B. (2014). A toolset for support of teaching formal software development. In Giannakopoulou, D. and Salaun, G., editors, *Software Engineering and Formal Methods*, volume 8702 of *LNCS*, pages 278–283. Springer.

Kozulin, A., Gindis, B., Ageyev, V. S., and Miller, S. M. (2003). *Vygotsky's Educational Theory In Cultural Context*. Cambridge University Press.

LeBlanc, R. J., Sobel, A., Diaz-Herrera, J. L., Hilburn, T. B., et al. (2006). *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. IEEE Computer Society.

Leino, K. R. M. (2010). Dafny: An automatic program verifier for functional correctness. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, pages 348–370, Berlin, Heidelberg. Springer-Verlag.

Mandrioli, D. (2015). On the heroism of really pursuing formal methods. In *Formal Methods in Software Engineering (FormaliSE), 2015 IEEE/ACM 3rd FME Workshop on*, pages 1–5. IEEE.

Mayer, R. E. (1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. *Journal of Educational Psychology*, 67:725–734.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *ACM Comput. Surv.*, 13(1):121–141.

Nipkow, T., Paulson, L. C., and Wenzel, M. (2002). *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer.

Page, R. L. (2003). Software is discrete mathematics. *ACM SIGPLAN Notices*, 38(9):79–86.

Pane, J. and Myers, B. (1996). Usability issues in the design of novice programming systems. *School of Computer Science Technical Report CMU-CS-96-132*.

Parnas, D. (1993). Predicate logic for software engineering. *Software Engineering, IEEE Transactions on*, 19(9):856–862.

Parnas, D. L. (2010). Really rethinking 'formal methods'. *Computer*, 43(1):28–34.

Parnas, D. L. (2011). Precise documentation: The key to better software. In Nanz, S., editor, *The Future of Software Engineering*, pages 125–148. Springer Berlin Heidelberg, Berlin, Heidelberg.

Penzenstadler, B., Bauer, V., Calero, C., and Franch, X. (2012). Sustainability in software engineering: A systematic literature review. In *Evaluation Assessment in Software Engineering (EASE 2012), 16th International Conference on*, pages 32–41.

Pokorny, M. (2012). Efficiency of blended learning in teaching logic, sets and binary relations. In *2012 IEEE 10th International Conference on Emerging eLearning Technologies Applications (ICETA)*, pages 301–305.

Powers, K., Ecott, S., and Hirshfield, L. M. (2007). Through the Looking Glass: Teaching CS0 with Alice. *SIGCSE Bull.*, 39(1):213–217.

Prince, M. J. and Felder, R. M. (2006). Inductive teaching and learning methods: Definitions, comparisons, and research bases. *Journal of Engineering Education*, 95:123–138.

Richardson, F. C. and Suinn, R. M. (1972). The mathematics anxiety rating scale: psychometric data. *Journal of counseling Psychology*, 19(6):551.

Sahami, M., Guzdial, M., McGettrick, A., and Roach, S. (2011). Setting the stage for computing curricula 2013: computer science–report from the acm/ieee-cs joint task force. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 161–162. ACM.

Sedelmaier, Y. and Landes, D. (2015). Active and inductive learning in software engineering education. In *Proceedings of the 37th International Conference on Software Engineering*, volume 2 of *ICSE*, pages 418–427. IEEE Press.

Sherman, B. F. and Wither, D. P. (2003). Mathematics anxiety and mathematics achievement. *Mathematics Education Research Journal*, 15(2):138–150.

Sirkiä, T. and Sorva, J. (2012). Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises. In *Proceedings of the 12th International Conference on Computing Education Research*, pages 19–28. ACM.

Spichkova, M. (2016). Applied logic in engineering. CoRR.

Spichkova, M., Hölzl, F., and Trachtenherz, D. (2012). Verified System Development with the AutoFocus Tool Chain. In *2nd Workshop on Formal Methods in the Development of Software*, WS-FMDS.

Spichkova, M., Liu, H., Laali, M., and Schmidt, H. W. (2015a). Human factors in software reliability engineering. *Workshop on Applications of Human Error Research to Improve Software Engineering (WAHESE2015).*

Spichkova, M. and Schmidt, H. (2015). Requirements engineering aspects of a geographically distributed architecture. *10th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2015)*.

Spichkova, M., Zamansky, A., and Farchi, E. (2015b). Towards a human-centred approach in modelling and testing of cyber-physical systems. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 847–851.

Spichkova, M., Zhu, X., and Mou, D. (2013). Do we really need to write documentation for a system? In *International Conference on Model-Driven Engineering and Software Development (MODELSWARD'13)*.

Sprankle, M. and Hubbard, J. (2011). *Problem Solving & Programming Concepts*. Pearson Education.

Sznuk, T. and Schubert, A. (2014). Tool support for teaching hoare logic. In Giannakopoulou, D. and Salaün, G., editors, *Software Engineering and Formal Methods: 12th International Conference, SEFM 2014 Proceedings*, pages 332–346. Springer International Publishing, Cham.

Tavolato, P. and Vogt, F. (2012). Integrating formal methods into computer science curricula at a university of applied sciences. In *TLA+ Workshop at the 18th International Symposium on Formal Methods*.

Trierweiler, S. and Stricker, G. (1998). *The Scientific Practice of Professional Psychology*. Springer.

Tucker, A. B., Kelemen, C. F., and Bruce, K. B. (2001). Our curriculum has become math-phobic! *ACM Sigcse Bulletin*, 33(1):243–247.

Vallance, S., Perkins, H. C., and Dixon, J. E. (2011). What is social sustainability? a clarification of concepts. *Geoforum*, 42(3):342 – 348. Themed Issue: Subaltern Geopolitics.

Vosinakis, S., Koutsabasis, P., and Anastassakis, G. (2014). A platform for teaching logic programming using virtual worlds. In *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT)*, pages 657–661.

Wang, S. and Yilmaz, L. (2006). A strategy and tool support to motivate the study of formal methods in undergraduate software design and modeling courses*. *International Journal Of Engineering Education*, 22(2).

Wang, Z., Hart, S. A., Kovas, Y., Lukowski, S., Soden, B., Thompson, L. A., Plomin, R., McLoughlin, G., Bartlett, C. W., Lyons, I. M., and Petrill, S. A. (2014). Who is afraid of math? two sources of genetic variance for mathematical anxiety. *Journal of Child Psychology and Psychiatry*, 55(9):1056–1064.

Wing, J. M. (2000). Weaving formal methods into the undergraduate curriculum. In *Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology*, pages 2–7.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36.

Woodcraft, S. (2012). Social sustainability and new communities: Moving from concept to practice in the UK. *Procedia - Social and Behavioral Sciences*, 68:29 – 42.

Zamansky, A. and Farchi, E. (2015a). Exploring the role of logic and formal methods in information systems education. In *2nd Human-Oriented Formal Methods workshop (HOFM 2015)*.

Zamansky, A. and Farchi, E. (2015b). Teaching logic to information systems students: challenges and opportunities. In *Proceedings of the 4th International Conference on Tools for Teaching Logic (TTL)*.

Zobel, J. and Hamilton, M. (2002). Managing student plagiarism in large academic departments. *Australian University Review*, 45(2):23 – 30.