

Support for Teaching Formal Methods

Report of the ITiCSE 2000 Working Group on Formal Methods Education

Vicki L. Almstrum (co-chair)

The University of Texas at Austin, USA

almstrum@cs.utexas.edu

C. Neville Dean (co-chair)

Anglia Polytechnic University

c.n.dean@anglia.ac.uk

Don Goelman

Villanova University, USA

goelman@vill.edu

Thomas B. Hilburn

Embry-Riddle Aeronautical Univ., USA

hilburn@db.erau.edu

Jan Smith

Chalmers University of Technology

smith@cs.chalmers.se

Abstract

This report describes a growth path for the area referred to as *formal methods* within the computing education community. We define the term formal methods and situate it within our field by highlighting its role in Computing Curricula 1991, Computing Curricula 2001, and the SoftWare Engineering Body Of Knowledge (SWEBOK). The working group proposes an enhancement to an existing web resource, which is a rich collection of materials and links related to formal methods. The new resource is designed to provide a bridge between the general computing education community and the formal methods community. The goal is to allow the latter to provide useful support for the former for the ultimate benefit of all of our students. Eventually, the working group aspires to see the concepts of formal methods integrated seamlessly into the computing curriculum so that it is not necessary to separate them in our discussions.

1. Introduction

Each of the members of this working group is a strong proponent of including topics from the area called formal methods as a vital component of computing education at all levels. Specifically:

- We support the use of discrete mathematics and

formal methods in the teaching and learning of computer science and software engineering.

- We believe graduates of programs that use and emphasize formal methods will be better prepared for advanced study and research in computing and able to apply mathematical techniques to the development and verification of software artifacts (such as requirements specifications or design specifications).
- We view formal methods as the “calculus” of software engineering; without formal methods, the engineering is non-existent or suspect.

During our pre-conference preparations, the working group’s focus evolved to concentrate on providing mechanisms to allow the computing education community at large to incorporate formal methods more easily across the curriculum. This focus reflects the experiences and biases of a wide collection of concerned educators, since we invited a number of individuals with strong backgrounds in this area to serve on an advisory board. The advisory board members, who are acknowledged in Appendix A, have contributed to the work in a variety of ways, among them position papers (listed among the references, discussed in Section 4 of this paper, and cited in Appendix A), pointers to other documents, on-line and off-line discussions, and reviews of the in-progress work.

Even though formal methods are very widely used in the semiconductor industry [18]—indeed, this is the area of computing in which they have had their greatest success to date—the emphasis in this report is on software. In part this reflects the background of the committee, but it is also consistent with the greater emphasis on software in computing education.

The remainder of this report is divided into four major sections. Section 2 defines what we mean by formal methods, addresses some of the challenges and problems related to its adoption, and considers what can be done to address the communication gap between formal methodists and the computing education community at large. Section 3 defines formal methods in the context of current curriculum models and frameworks. Section 4 is an overview of resources that are currently available as sources of support for teaching formal methods. Section 5 describes the key contribution of this working group. We have set the stage for a new web-based resource that targets computing educators who would either like to begin using formal methods or who would like to expand their repertoire of use. We fit our proposal into the framework of a well-established project with broad community support, which bodes well for its long-term potential to affect the way the computing community views the prospect of teaching formal methods.

2. Formal Methods: Its Nature and Relevance

2.1 Definition

In our study and interaction about the nature of formal methods, we found various beliefs about the character and content of this area. A prevalent characterization (especially in an industrial context) is this one from Nancy Leveson [29]:

A broad view of formal methods includes all applications of (primarily) discrete mathematics to software engineering problems. This application usually involves modeling and analysis where the models and analysis procedures are derived from or defined by an underlying mathematically-precise foundation.

Robert Vienneau [42] offers this focused definition of formal methods:

A formal method in software development is a method that provides a formal language for describing a software artifact (e.g. specifications, designs, source code) such that formal proofs are possible, in principle, about properties of the artifact so expressed.

In subsequent sections, we assert the need and rationale for including formalism throughout the computer science curriculum. This motivates the more general definition that the working group adopted:

Formal methods involves the use of discrete mathematics and mathematical logic in the study and practice of computer science and software engineering.

With this definition, we intend to encapsulate within formal methods all its associated concepts and techniques, and both their informal and formal application.

Note: We also view “formal methods” as a single entity within mathematics (with related methods, techniques and applications) rather than as a set of discrete entities. Hence, we use the singular when referring to formal methods.

2.2 Foundations of Computing

From its beginning in the 1930s, computing was regarded as an abstract, mathematical science. The work of pioneers like Turing, Church, and von Neumann used mathematics to establish the essences and boundaries of the computing discipline. Although computer technology is crucial in computer science education and practice, the underpinnings *are* mathematical in nature and computing does deal with purely logical processes [31]. However, students (and some faculty) often resist the use of mathematics in the study of computing. There are many explanations for this resistance: students may lack the proper preparation or motivation; many individuals have neither an understanding of nor appreciation for the role of mathematics (formal methods) in computing [26]; and some feel intimidated by (and even fearful of) the level of mathematical knowledge and capability they believe is required.

Using the working group’s general definition from section 2.1, formal methods is found throughout the computing discipline. For example, although the connection is not always made clear to them, computing students begin by studying and using formal languages (the syntax and semantics of programming languages); programs can simply be viewed as a string of symbols from the language (similar to formulae in other mathematical systems). In Section 3, we explore how formal methods is or may be used throughout a computing curriculum.

2.3 Software Development

Although the term software engineer is becoming a popular title for software developers, there is little evidence to show that the practice of software engineering compares with the rigor and discipline that is required for practice in other engineering fields. For the software discipline, this is not just a commercial issue, but a professional and ethical problem, which would not be tolerated in other fields of science and engineering. Parnas contrasts traditional CS programs with most engineering programs, arguing that software engineering programs should follow the traditional engineering approach to professional education [35].

Quality problems arise from incomplete and imprecise requirements specifications, shoddy designs with poor documentation, and the almost sole reliance on testing for software quality assurance. Although there is increasing interest in the use of formal methods for specification and design, the concepts of mathematical modeling and rigorous verification (the staple of other engineering fields) are still used sparingly in commercial software development. This is partly revealed in a survey of software developers by Timothy Lethbridge [28], where the

developers ranked the importance of 75 knowledge areas. *Formal Specification Methods* was ranked 37th and *Predicate Logic* was ranked 39th. (Areas like *specific programming languages*, *data structures*, *software architectures* and *requirements elicitation* were ranked at the top, while areas such as *differential equations* and *VLSI* were ranked at the bottom.)

However, there are examples where formal methods has been used effectively to develop high-quality products. Most of these successes have been in safety critical areas and have involved the integration of formal methods with more traditional non-formal approaches [6, 17, 20, 21, 44]. For many years the use of formal methods has been mandated or encouraged in critical national security systems. With the explosive growth of electronic commerce, there is growing interest in formal methods for achieving higher confidence in other critical infrastructure systems [7, 36, 37].

A more prevalent view of formal methods is that it is intended to support the “specification and verification of hardware and software systems” [45]. Under this view, current curricula have a low level of emphasis on formal methods. This is surely due, at least in part, to a lack of interest on the part of the software industry, but much of the responsibility must be attributed to the state of conceptual curriculum design. The computing education community has adopted a curriculum strategy of dividing curricula elements into areas of “theory” and “practice”. This causes both faculty and students to view the theory of computing as separate and distinct from the practice of computing. Too often, the result is theorists who are viewed as the mathematical elite and practitioners with little respect for the applicability of formal methods to their work. This mindset inhibits the use and integration of formal methods into the software development process. Because of this (and other reasons mentioned previously), there is little guidance and support available to faculty that would like to introduce formal methods into their curricula. Also, currently available software tools are, for the most part, designed for use in research or industrial settings. We explore these issues in more depth in the following sections.

3. FM in Computing-Related Curricula

3.1 Computing Curricula '91 and 2001

Formal methods topics have had a significant role in model computer science curricula, especially in the most recent versions: Computing Curricula '91 [41] and the draft of Computing Curricula 2001 [8]. In this section we examine the role of formal methods, both explicit and implicit, in these curricula. Because Computing Curricula 2001 is still a work in progress, details will change by the time a finished product is published; our analysis is based on a version that resulted from a Steering Committee meeting in June 2000.

Computing Curricula '91 (CC91) organized the body of computer science into ten *knowledge areas*, each consisting of a set of *knowledge units*. The Computing Curricula 2001 (CC2001) committee determined that the evolution of the discipline necessitated important modifications and additions to the earlier organization. The committee therefore added four new knowledge areas and shuffled, modified, added, and deleted topics among the CC91 set of knowledge areas. Most significant for our working group topic, perhaps, is the recommendation that Discrete Structures be added as a separate area in its own right, rather than as a mathematics prerequisite (which was and is the role of *continuous* mathematics). This reflects the opinion of the CC2001 committee that many topics in the Discrete Structures knowledge area, such as basic logic and proof techniques, are so fundamental in other knowledge areas that they should be taught in a *computer science* environment with those areas in mind. Thus, many formal methods topics appear *explicitly* in the Discrete Structures knowledge area of CC2001.

Within both CC91 and CC2001, sets of topics are associated with each general knowledge area. Links among the topics and pedagogical suggestions are also being developed for CC2001, although most of those details have not been finalized at the time of this writing. In CC91, topics were cross-referenced with certain *recurring concepts*, one of which is *conceptual and formal models*. In examining the place of formal methods across the curriculum, the working group started with the units in CC91 that correlated with conceptual and formal models as a recurring concept, and located topics in CC2001 that are based on (or identical to) those units. This led to using the more recent nomenclature.

In addition to the technical approach based on the conceptual and formal models topic, the working group also took a non-technical perspective. We note that a number of regulatory bodies and established standards make a connection between systems in which high assurance is particularly important and formal methods as a means of achieving high assurance. For many years this has been the case for safety-critical systems and national security systems. It is becoming increasingly important in financial systems and systems in which privacy is required. For this reason, formal methods correlate with certain topics in the CC2001 knowledge area SP Social, Ethical, and Professional Issues.

Appendices B and C are related to this classification. Appendix B shows a top-level mapping between the two model curricula, together with the number of suggested core hours in CC2001. The mapping is not one-to-one, as some knowledge areas were added and many units were reassigned among the areas. According to Clarke [6], the core was determined as “those subjects that we would be embarrassed to have computer science majors graduate without knowing”. Appendix C lists the topics (or units)

from both model curricula that correlate with formal methods.

As a status check on the computing education community's view of the role of formal methods, the working group conducted a survey as part of our poster session during the ITiCSE conference; the results are given in Appendix D. While the number of responses was small, the results showed that the respondents do perceive a strong relationship between the model curricula and the importance of formal methods, in contrast to the "math-phobic" evidence presented by Clarke [6].

3.2 Software Engineering Body of Knowledge

A current project of the Software Engineering Coordinating Committee (SWECC) is the development of *A Guide for the Software Engineering Body of Knowledge* (SWEBOK) [11]. The SWEBOK is being developed in phases and the final version is due in 2002. The focus of SWEBOK is more specialized than the computing curricula efforts discussed in the previous subsection. One objective of the guide is to provide support for "the students learning the software engineering profession and educators and trainers engaged in defining curricula and course content". The SWEBOK divides software engineering knowledge into ten areas, with descriptions and guidance for each area. In order to enhance the SWEBOK as a tool for designing course material, programs, and accreditation criteria, the committee used Bloom's Taxonomy [4] to evaluate the required or desired capability for each of the topics included in the knowledge areas. Appendix E lists the ten SWEBOK knowledge areas and shows examples (drawn from the SWEBOK, our working group discussions, and other sources [3, 45]) of parts of formal methods that would apply in each area. The table, which is consistent both with the general computing curricula (Appendix C) and the ITiCSE survey (Appendix D) but is more detailed than both of them, suggests a framework for introducing and using formal methods in software development courses. At a minimum, the information in Appendix E can help faculty and students understand and appreciate how and where formal methods can be used. Those knowledge areas that benefit most from the use of formal methods are requirements, design, construction, and quality.

The SWEBOK does not emphasize or promote formal methods. It appears to rely more on semi-formal methods, such as using inspections and reviews throughout the development life cycle. However, the SWEBOK, with its organization and descriptions of knowledge areas, its extensive list of references, and its emphasis on analysis and model building, provides an excellent resource for curriculum and course designers trying to determine where and how to introduce formal methods.

Note: SWEBOK also includes a number of statistical techniques used in software management for risk management, project estimation and tracking, and process

and product assessment. These are not a part of formal methods.

4. The Current State of FM Resources

A variety of resources are available to support the study and application of formal methods, including software tools, books, web pages, handouts, mail lists, tutors/teaching assistants, and fellow students. In the context of this report, we restrict our consideration of resources to software tools, web-based resources, and other resources that directly support these (for example, a book on how to use a particular tool). In this section, we consider the issue of resources from three angles:

reports and papers submitted to the FM working group, including a range of statements from the FM advisory board, a resource use survey, and the FM Educational site.

4.1 Reports and Position Statements

This section highlights some key points from several documents that were submitted by advisory board and working group members. These excerpts set the stage for the remainder of this report.

1) *WetStone Technologies, Formal Methods Framework, (Communicated by Randolph Johnson) [43]*

This is an extensive survey of tools used in Formal Methods. It includes a framework for classifying formal methods tools and the responses to a survey investigating the use of twelve tools. The survey addressed the nature of each tool, its availability and cost, prerequisite knowledge, and platforms. All of these tools are essentially practitioner tools and include ACL2, HOL, Larch Prover, PVS, Z/EVES, Concurrency Factory, Murphy, SVM Cadence, SPIN, NRL Protocol Analyzer, SCR, and Tatami.

2) *M. Barjaktarovic and WetStone Technologies, Report: State of the Art in Formal Methods, (Communicated by Randolph Johnson) [1]*

"Formal methods are in different stages of development, in a wide spectrum from formal languages with no tool support, to internationally standardized languages with tool support and industrial users. The field of formal methods is in a great flux and evolving rapidly, leaving research laboratories and making inroads into industrial practice."

"The major task of the formal methods community will be to provide the assistance sought. Expressed needs include: more user-friendly tools; more powerful and robust tools; more real-life applications; more infrastructure such as verified libraries; more publicity of success stories and available technologies; and more user training."

3) *Jeannette Wing, Weaving Formal Methods into the Undergraduate Computer Science Curriculum [45]*

“There is no excuse not to be using model checkers in our undergraduate courses today. With a verification tool, we can more easily teach that verification complements the testing and simulation activities of practicing hardware and software engineers. ... [I]t behooves us as educators to ensure that our students are well-versed in the state of the art verification technology.”

“Theorem provers require more expertise than we can expect our students to acquire in one semester, all the while learning other course material.”

4) Jan Smith position statement [39]

This statement outlines an approach to teaching proofs using a tool based on type theory (specifically, Alfa, developed at Chalmers University, and Coq, developed at INRIA).

“Experience of using formal proof systems in undergraduate education is limited, mostly to deductions in logic. We are proposing to present conventional mathematical subjects in a formal, machine checked, but still careful and explanatory manner. Students will be able to browse the formal definitions and proofs at a level of detail they choose, and will work their exercises using the formal proof tool.”

“We are rapidly moving towards an information society which will change both how and what we teach our students. In mathematics there is already a trend towards computations, but we expect a more radical change: a renaissance of proof in education. For the first time, the art of making proofs is on its way to become an engineering discipline because of the fundamental importance of correct hardware and software. For undergraduate students it is difficult to understand the rules by which proofs are made and even more difficult for them to develop their own proofs. The fulfillment of this project will give powerful tools to increase students’ understanding of what a proof is and how to develop proofs.”

5) Dan Craigen position statement [9]

“Our Z/EVES system has been extensively used for teaching purposes. ... Overall, Z/EVES appears to be used in a lightweight perspective for teaching. Few students/lecturers move through the adoption curve to performing complex proofs. Obviously, time and experience is an issue. However, there have been a number of undergraduate and graduate projects that have used the full capabilities of the system (with varying degrees of success). Various researchers and commercial types have pushed Z/EVES to its full extent.”

6) Kathi Fisler position statement [12]

“Formal methods education should address how to identify, develop, and prove formal statements—in particular, theorems—about programs and systems. This encompasses activities ranging from type-checking (where the theorem is stated implicitly and proved automatically) to model-checking (theorem stated explicitly and proved

automatically) to theorem-proving (theorem stated explicitly and proved manually).”

“Students at all levels should be encouraged to think about the theorems associated with computer science. In particular, a student should be able to answer the following questions:

- [1] What kinds of formal statements can be made about a system?
- [2] When is a formal statement about a system provable?
- [3] When is a formal statement about a system useful?
- [4] What resources are needed to prove a given formal statement?
- [5] What tools and techniques exist for validating formal statements about systems?”

7) Randolph Johnson position statement [25]

“I think that tool support for formal methods is essential. I have used various Z tools, most recently version 1.5 and version 2.0 of Z/EVES (see [9] for more information on Z/EVES). Among its strong points are that it runs on a variety of platforms and is free for academic use. In my experience, the biggest drawback of version 1.5, at least for student use, is that you had to know LaTeX and emacs in addition to learning formal methods and Z. Version 2.0 added a GUI and eliminated the need to know LaTeX and emacs. This is *much* better for beginners.”

“For educational use, maybe the biggest value of Z/EVES (and many other formal methods tools) is that it does syntax checking and type checking at the push of a button. Some students may never do much more than that. A nice feature of Z/EVES which I haven’t seen in other tools is that it goes beyond type checking to automatically generate domain conditions. These state that the argument to which a partial function is applied in a spec is actually in the domain of the function. Not only does it generate the conditions; it tries and often succeeds in proving them. With no effort on the part of the instructor, the tool repeatedly draws the attention of the student to an aspect of their specification which is often ignored, even by very experienced Z specifiers.”

8) Peter Gorm Larsen position statement [27]

“I believe that formal methods education should be introduced in stages and that in order to keep the interest for the students the use of tools is extremely important. Furthermore I feel that it is important to the students that the use of formal methods in themselves simply is a means to achieve better systems/software and not a goal in itself. It is my experience from some formal methods promoters that this is not sufficiently stressed. Thus it is in my opinion important to be able to envisage how one’s formal method and the associated tools could be applied in a real system/software development environment.” Larsen adds: “Tools such as VDMTools that integrate with state-of-the-art commercial tools such as Microsoft Word and Rational

Rose on platforms such as Windows give the students more appreciation for the formal methods usability.”

9) *J Strother Moore position statement* [32]

“Why teach just one tool? Why ACL2 among all the choices? ... ACL2 is the tool I know best. ... Enthusiastic teachers who deeply understand the subject matter tend to be good teachers. However, when one teaches a course based upon a particular tool, it is incumbent upon the teacher to explore the tool’s inadequacies, especially those that result from fundamental design decisions.”

“The argument for teaching just one tool is simple: a semester is not very long. If I were teaching a course on programming, I would rather the students learn one “first language” than several.”

“ACL2 is a good choice for the following reasons. ... There is now a textbook introduction... The tool is free and runs on many platforms. The tool is rugged, well-documented online, and widely used. Within the ACL2 setting there is a natural way to study some other tools... Finally, and very importantly, ACL2 is not a pedagogical toy but an advanced industrial-strength theorem prover ...”

10) *Lesley Semmens position statement* [38]

“I see the main problem in the early teaching of formal methods to be the students’ limited ability to model. The notations are not the main problem. I therefore try to build on other experience and understanding they already have. I have tried many different approaches over the ten years I have been teaching formal methods. I have done it with and without tools (fUZZ, Formaliser, ZTC). We have even built tools to translate from ERDs to Z. But, always it comes back to the students’ ability (or lack of ability) in modeling. Using tools, student beginners produce syntactically correct Z, but often it is semantic nonsense; they concentrate far too much on the syntax and coping with the idiosyncrasies of the tool.”

“I am not totally against tools. With [students in their] final year, who have more understanding of what they are doing (and more ability in modeling), I use tools, such as fUZZ and ZTC if time permits. In any case I encourage them to try out the tools, in the same way as they might any other software engineering tool. This seems to work, they are not trying to learn two things at once and appreciate the help the tool gives them.”

11) *Jonathan Bowen position statement* [5]

This position statement specifically addresses the issue of web-based teaching of the formal notation Z. The author describes his experiences of delivering an FM course in this manner, including the use of books and software tools (LaTeX, ZTC and ZANS).

“No one book was followed exactly. It was recommended that a Z textbook be obtained by students and used to complement the course unit with additional reading outside lectures.”

“Student understanding of producing a Z specification increased significantly after the practical sessions. Many seemed to appreciate using tools far more than just pencil and paper. However, a danger with the use of an animator is the possibility of confusion between a (possibly non-executable) formal specification and an executable program. A few always seem to stubbornly fail to recognize the difference even after this is emphasized repeatedly in lectures.”

“In the experience of the author, using tools in supporting formal methods course units helps the students in their understanding and increases their appreciation of the usefulness (or at least decreases their negativity) of formal methods.

“By insisting that students type-check their Z specifications (using the ZTC tool on the course unit described here) and check for explicitness (or otherwise) at least using the related ZANS animation tool, many errors in Z specifications can be discovered and eliminated by the students themselves, sometimes with no help from demonstrators in the case of bright students. This allows demonstrators (and markers) to concentrate on the more interesting and difficult aspects of formulating a Z specification that require human inspection.

“Web support for formal methods and other course units is a useful adjunct. A benefit is the accessibility of material by students, the staff involved, other colleagues, internal and external examiners, etc. It also helps in the maintenance of course unit material as a unit develops since this can be easily added and information can quickly be corrected in the case of errors. However, it is recommended that all essential material is still given to students in paper form, even if it is available on the web, since students will tend to print this anyway, which is still relatively expensive compared to photocopying. Of course the web resource can contain considerable extra supplementary material if desired at very little cost once it is installed.”

Themes in the Excerpts

We found several themes emerging from these excerpts:

- 1) The need for more robust and user-friendly tools.
- 2) There is a wide spectrum of tools available (perhaps too wide for educational purposes).
- 3) The value of including industrial strength tools in the curriculum, particularly with regard to model checking.
- 4) Students need more educationally oriented tools to help learn the concepts of theorem proving rather than being overwhelmed by the power of industrial strength tools.
- 5) Ancillary issues (such as the operating system a tool runs under, and the need to learn other software tools, such as LaTeX and emacs) can have major adverse effects on the students’ learning experience.

- 6) Tools can be extremely important in keeping the interest of students and enhancing the learning experience.
- 7) Tools are not necessarily a good idea at the introductory stages; indeed they may hinder the learning, or even mislead the student into a mindset where formal methods are seen as yet another programming approach!
- 8) Web support is useful, if only as a means of providing access to the (latest) handout materials.

4.2 Uses of FM Resources

A questionnaire circulated before the ITiCSE conference investigated resource use among members of the Formal Methods focus group of the (UK-based) Learning and Teaching Support Group for Computer Science and Information Technology. Participants in this mailing list discuss the teaching of formal methods in the United Kingdom. The response rate was about 50% and provided useful information about the situation in the UK. (Note that FMs teaching has a longer history than in the US.) Appendix F lists the questions and overviews the responses. The following lists some key points from the results:

- The respondents use a wide variety of tools, mostly industrial strength.
- Z is very popular in the UK but other FM approaches are also used.
- Very few of the tools include features designed to help students develop their FM skills.
- Based on follow-up communications with the respondents, it seems that many tools (especially the Z tools) are somewhat problematic in use: some work only on a Unix platform; several require the use of LaTeX or troff; and others have integrated editors that students find difficult to use. For example, Formaliser uses a structure editor, which students find difficult to use, although it could be argued that thinking more structurally is a valuable skill (in which case Formaliser can be seen as a tool having an educational function).
- There seems to be a dearth of good backup materials (tutorials, handouts, books), although at least one respondent has produced good material using LaTeX and CadiZ. An exception to this seems to be Toolbox Lite (for VDM); this is an educational version of the Toolbox tool and comes packaged with a textbook on VDM [13].
- There is very little use of web-based resources. There are however a wide variety of textbooks available and in use. About a quarter of respondents indicated that their courses tend to be “traditional”, that is, book-based with hand-worked exercises.
- Opinion seems to vary on what constitutes a formal method and a formal methods tool. For example, are

declarative languages like Prolog and ML to be considered as FMs or FM tools?

Generally, the state of the art in resources for teaching formal methods seems to be a maze that the novice must try to navigate: “which FM shall I teach?”; “which tool shall I use?”; “what materials are available?”. The temptation may be to start with the available resources and to choose (perhaps somewhat randomly) those that seem most convenient or that are used on apparently similar courses. But the working group and advisory board members believe that a better approach would be to first identify student needs and then decide how best to facilitate student learning. There must be some means of helping both the novice and the more experienced educator to decide what resources (in the general sense outlined above) may aid the achievement of these aims and objectives.

In his invited ITiCSE talk, Needham [34] talked about the conflict between teaching basic CS concepts and the need for students to have experience building large-scale systems. Although Needham’s remarks focused on tools in general, they are equally applicable to the specific area of formal methods. He suggested that, at an introductory level, industrial-strength tools are too much for students and hinder the students’ understanding of concepts; that such tools are appropriate and desirable in final-year group projects; and that, when used, the tools should be fully understood by the educators. This has been a recurring theme in our investigations. It is clear that there is a need for more thought and study about providing resources to aid learning and that there must be suitable materials, such as tutorials and books, to support the resources.

In point 9 in the previous subsection, Moore makes a good start at describing what attributes a good tool should have if it is to be used for educational purposes. For the teacher, the problem is to define clearly what the aims and objectives are at each stage in the students’ learning and to relate these to the resources available.

4.3 FM Educational Site

As seems to be the case in essentially every imaginable area, the World Wide Web offers an ever-expanding array of resources in the area of formal methods. The definitive gateway to information about formal methods is the World Library for Formal Methods [46], maintained by Jonathan Bowen. This site serves as a clearinghouse to a wealth of information about all aspects of formal methods: research and practice, theory and application.

Up until 1998, there was little to help guide the educator through the World Library’s maze of materials on formal methods. This situation was addressed as one outcome from the 21st Century Engineering Consortium Workshop, held in Melbourne, Florida, in March 1998. The workshop’s principal concern was to promote formal methods education in computer science and computer engineering programs. Toward this end, the organizers gathered leading practitioners, experienced academics, and

government advocates interested in educational issues relevant to formal methods development. One outcome from the workshop was a report [24] that describes the state of formal methods education in academia and practice and makes recommendations for education and training. The other key outcome of this workshop was the Formal Methods Education site [14], which was created and has been maintained by Kathi Fisler. This web site is essentially an information hub, with the general philosophy of providing a collecting point for materials related to teaching formal methods. The site includes sections about course pages, tools, reading materials, instructional materials, benchmarks and examples related to formal methods, and position announcements. The Formal Methods Education site has become the educational node within the World Library for FM and is the foundation from which the resource we propose in the next section will emerge.

5. A Resource to Support FM Teaching

While the Formal Methods Education site presented in section 4.3 is a rich and varied resource, at this time it is primarily useful for educators who are already teaching formal methods and want to find materials for their existing courses. The wealth of information available at the site is somewhat overwhelming; it would be a daunting task for someone who wishes to begin teaching formal methods to navigate all of the information in order to come up with the “right” materials for a particular situation.

This working group proposes a new area for inclusion in the Formal Methods Educational site, with the goal of creating a set of pages to ease the continued integration of formal methods into existing and future courses. This resource will be a useful “entry point” for computing educators who are new to teaching formal methods or for those who would like to move along to the “next level” of teaching formal methods. It can also prove to be a useful tool to support virtual university studies in this area.

The remainder of this section describes our proposal for the organization of this sub-site and some of its features. The actual implementation will be an on-going process, which will require cooperation by a number of individuals and institutions. We discuss our plan of action where such a plan exists.

5.1 Proposed Content

In this section, we present the areas we propose for the new sub-site. Much of the required material already exists in various locations on the Web, such as this report, the report from the 21st Century Engineering Consortium Workshop [24], the Formal Methods Education site [14], the World Library for Formal Methods [46], and the site for Formal Methods Europe [15]. The challenge is to provide links in a way that makes it all easily accessible.

Introduction: What is Formal Methods? This section will be a brief summary (such as the one in the introduction

to this paper) that will point to more detailed information such as that found in the World Library for Formal Methods [46] or on the site for Formal Methods Europe [15].

FAQs We will seed the FAQ with some standard questions and answers. In order to make the FAQ section dynamic, we will include the means for submitting new questions and answers (with intervention by the current moderator before they become part of the site).

FM Motivation This section will be designed to assist educators who need facts and statements to convince colleagues or students about the merits of including formal methods topics in the curriculum. Subsections we anticipate include:

- Advantages, problems, and challenges.
- Statistics and evaluations (such as those in a survey by alumni of an introductory mathematics course at SUNY Stony Brook [40]).
- Success stories where formal methods have been used in real, large-scale projects.
- Standards in which the use of formal methods is mandated or encouraged, especially in the safety-critical and security areas.
- Template letters and flyers that can be downloaded, adapted, and printed.

Discrete mathematics Because discrete mathematics is the foundation for formal methods, it is natural to highlight this area. Some examples of existing resources include:

- The approach to teaching discrete mathematics presented in David Gries’s position statement [19].
- Neville Dean’s paper from ZUM ‘95 [10], which discusses how one can teach the discrete mathematics foundations of Z .
- The pages of the working group on Integrating Mathematical Reasoning into Computer Science Curricula [23], which has as its charter to raise awareness of mathematical reasoning in CS and of ways of teaching it.
- The LINK software system [30], which is aimed at educational applications of discrete mathematics at levels ranging from junior high to research. It includes a 50-page tutorial that is designed to be read with no prerequisite.

Formal Methods across the curriculum This section will provide the specific information that computing educators can use as they incorporate formal methods into a particular course or an entire curriculum. Sections can include:

- Current computing curricula guidance
- Courses matrix and FM components (drawn from appendices B and C)
- Guidelines for designing an FM course

- Modules that can be incorporated into other courses (reflecting the results from projects such as the one recently awarded to Rice University [2] and the training materials listed on this NASA site [33])

Approaches to evaluating the use of Formal Methods in computing-related curricula Such evaluations can be done after a course, after graduation, or after students have started to work. A positive evaluation may be the best motivation for encouraging others to try including formal methods modules in their courses. Evaluation is also a necessary basis for improvement of a curriculum. For an example evaluation, see the Stony Brook survey [40].

5.2 Resource Navigation Assistant

As a key enhancement of the FM Educational web site, we propose the addition of a resource navigation assistant (RNA). We feel such a capability would be invaluable in surveying the maze of resources. We visualize the RNA as more than a search engine; it would be built around a searchable network of interrelated resources that would allow cross-referencing among courses and curriculum modules, tutorials, software tools, textbooks, and relevant case studies, examples, and benchmarks. Eventually, this could become an interactive tool that guides the user (rather than requiring the user to ask the “right” questions). Because the FM Educational web site will evolve as people submit additional relevant material, the RNA itself will need ongoing maintenance and evaluation.

5.3 Educational Tools Wish List

Advances in the teaching of formal methods, for whatever reason, will be fostered by more coordination among teachers, with help from tool developers. Currently, most tools used to support the teaching of formal methods were developed for practitioners, rather than for educators or first-time users. What changes are needed in order to make such tools more useful in teaching and learning formal methods? In order to encourage the development of educationally desirable tools, we propose creating a wish list of important features and trade-offs. For example, such a tool should help highlight the underlying principles, should enhance the learning, should have a good learning curve trade-off (that is, the time invested in learning the tool should not outweigh the ultimate benefit of using the tool), and should have an intuitive and accessible user interface.

As the wish list develops, we suggest that a sub-section should be added with assessments of FM tools according to the criteria in the wish list. Ideally, such assessments should be based on actual experience using the tools in educational settings.

While the wish list will support the efforts we recommend for creating a resource for novice users, it is not intended for their direct use. As a result, the wish list should reside in a separate subsection of the FM education web site. The wish list should also inform the development of the resource navigation assistant we described in Section 5.2.

5.4 Logistics for Creating the New Resource

The initial plan for implementing the proposed web pages is to extend the existing FM Educational site [14]. The webmaster of that site is committed to refining and improving the site and has been a very active member of the advisory board. We visualize the development of the new area as an evolutionary process, which will respond to suggestions and newly emerging needs. The logistics for bringing the site into use include encouraging contributions and publicizing the site. As one aspect of publicizing the FM Educational resource, the working group proposes a simple flyer that can be readily available on the web site. This will allow FM proponents to print out the flyer and have it available on the handout table during appropriate conferences. We will also devise a plan for distributing information about the resource via relevant mailing lists.

6. Conclusions

This report proposes a web-based resource related to teaching formal methods. The intended consumers of the new resource are computing educators who wish to begin teaching formal methods as well as educators already teaching formal methods who wish to move to the “next” level. Rather than duplicating information that already exists, the intention is to create a resource that simplifies the task of effectively accessing useful and usable information in this area. The working group’s web site [16] and the existing FM Educational site [14] will be ongoing sources of information about these efforts. A key part of the future work will be continued evaluation and feedback.

By the time this report appears, the Formal Methods Education site [14] should include the first basic version of this new resource. We anticipate that the site will ease the goal of expanding and enhancing the use of formal methods throughout the computing curriculum. We encourage others to join us and to contribute to this area, which is critical to the improvement of computing education and the preparation of our students for research and software engineering practice.

Acknowledgement

The working group extends a special note of thanks to the following advisory board members, whose comments and suggestions on preliminary versions of the report have improved the content and presentation significantly: Kathi Fisler, Randy Johnson, Susan Gerhart, Ebba Thora Hvannberg, Peter Gorm Larsen, and Dan Craigen.

References

- [1] Barjaktarovic, M., and WetStone Technologies, Inc. *The State-of-the-Art in Formal Methods*, January 1998, available <http://www.cs.utexas.edu/~csed/FM/docs/StateFM.pdf>

- [2] Barland, I, Felleisen, M., Fislser, K, Kolatis, P., and Vardi, M. *Joint position statement: Integrating Logic into the Computer Science Curriculum*. Adapted from a grant proposal to develop materials, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/iticse-fislervardi.pdf>
- [3] Bjørner, D., and Cueller, J., Software Engineering Education: Roles for Formal Specification and Design Calculi, *Annals of Software Engineering*, April 1999.
- [4] Bloom, B. S. (Ed.) *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. New York; Toronto: Longmans, Green, 1956.
- [5] Bowen, J. P., *Experience Teaching Z with Tool and Web Support*, July 2000, available <http://www.cs.utexas.edu/users/csed/FM/docs/bowen.pdf> (FM Advisory Board contribution).
- [6] Clarke, E.M., et al., Formal Methods: State of Art and Future Directions, *ACM Computing Surveys* 28(4), December 1996.
- [7] The Common Criteria for Information Security Evaluation (CC) version 2.1 / ISO IS 1540, available <http://csrc.nist.gov/cc/>
- [8] Computing Curricula 2001, DRAFT (March 6, 2000), The Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery, March 2000, available <http://computer.org/education/cc2001/>
- [9] Craigen, D., Position Statement, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/craigen.html> (FM Advisory Board contribution).
- [10] Dean, N. Mental Models of Z: I -- Sets and Logic. In Bowen J and Hinchey M (Eds.), *ZUM'95: The Z Formal Specification Notation*, LNCS 967, 1995, pp. 498--507.
- [11] Dupuis, R., et. al. A Guide to the Software Engineering Body of Knowledge, Version 0.7, available <http://www.swebok.org>.
- [12] Fislser, K., Position Statement, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/fislser.html> (FM Advisory Board contribution).
- [13] Fitzgerald, J. and Larsen, P. G. *Modelling Systems: Practical Tools and Techniques for Software Development*. Cambridge University Press, 1998.
- [14] Formal Methods Educational Site, <http://www.cs.indiana.edu/formal-methods-education/>.
- [15] Formal Methods Europe, available <http://www.fmeurope.org/>
- [16] Formal Methods Working Group, ITiCSE 2000, available <http://www.cs.utexas.edu/~csed/formal-methods/>.
- [17] Gerhart, S., Craigen, D., and Ralston, T., Experience with Formal Methods in Critical Systems, *IEEE Software*, January 1994, pp. 21-28.
- [18] Gopalakrishnan, G. and Windley, P. (Eds.) *Proceedings 2nd International Conference on Formal Methods in Computer-Aided Design, FMCAD'98*, Palo Alto, CA, November 1998, LNCS 1522, Springer Verlag, 1998.
- [19] Gries, K., Position Statement, July 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/gries.html> (FM Advisory Board contribution).
- [20] Hall, A., Using Formal Methods to Develop an ATC Information System, *IEEE Computer*, March 1996.
- [21] Hinchey, M.G. and Bowen, J.P., *Applications of Formal Methods*, Prentice-Hall, 1995.
- [22] Hvannberg, E. T., Position Statement, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/hvannberg.html> (FM Advisory Board contribution).
- [23] Integrating Mathematical Reasoning into Computer Science Curricula, available <http://www.cs.geneseo.edu/~baldwin/math-thinking/>
- [24] Johnson, S. D., Alexander, W. P., Chin, S. K., and Gopalakrishnan, G. (Eds.) *Report on the 21st Century Engineering Consortium Workshop: a forum on formal methods education*, March 1998, available <http://www.cs.indiana.edu/formal-methods-education/xxiec/report.html>
- [25] Johnson, R., Position Statement, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/johnson.html> (FM Advisory Board contribution).
- [26] Kelemen, C., Tucker, A., Henderson, P., Bruce, K., and Astrachan, O. Has Our Curriculum Become Math-Phobic? (an American Perspective), *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, July 2000, pp. 132-135.
- [27] Larsen, P. G., Position Statement, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/larsen.html> (FM Advisory Board contribution).
- [28] Lethbridge, T.C. What Knowledge is Important to a Software Engineer *IEEE Computer*, May 2000.
- [29] Leveson, N. G., Guest Editor's Introduction: Formal Methods in Software Engineering, *IEEE Transactions in Software Engineering*, September 1990.
- [30] LINK: A Software System for Discrete Mathematics, version: 1.3, 02/23/98, available <http://zhivago.elon.edu/~berryj/LINK.html>.
- [31] Mills, H. *Software Productivity*, Dorset, 1988.
- [32] Moore, J S., Position Statement, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/moore.pdf> (FM Advisory Board contribution).
- [33] NASA Jet Propulsion Laboratories, Formal Methods Training and Courseware, April 2000, available http://eis.jpl.nasa.gov/quality/Formal_Methods/training.html
- [34] Needham, R. Invited Talk, ITiCSE 2000, Helsinki Finland.
- [35] Parnas, D. L., Software Engineering Programs are not Computer Science Programs, *IEEE Software*, November/December 1999.
- [36] Critical Foundations: Protecting America's Infrastructures, The Report of the President's

- Commission on Critical Infrastructure Protection, October 1997, available <http://www.info-sec.com/pccip/web/index.html>.
- [37] Schneider, F. B. (Ed.) *Trust in Cyberspace*, Committee on Information Systems Trustworthiness, National Research Council, 1999, available <http://books.nap.edu/catalog/6161.html>.
- [38] Semmens, L., Position Statement, July 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/semmens.html> (FM Advisory Board contribution).
- [39] Smith, J. A Proposal for Computer Assisted Proof in Mathematics Education, June 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/smith.html> (FM Advisory Board contribution).
- [40] Stony Brook Computer Science alumni, "Foundations of CS I" Stony Brook Alumni Survey, Spring 1999, available <http://www.sinc.sunysb.edu/cse113/survey/>.
- [41] Tucker, A. B. (Ed.). *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force*. IEEE Computer Society Press, 1991, available <http://www.acm.org/education/curricula.html#CRVI>.
- [42] Vienneau, R., *A Review of Formal Methods*, Kaman Sciences Corporation, 1993.
- [43] WetStone Technologies, Inc., *Formal Methods Framework, final month status report*, Contract # F30602-99-C-0166, October 26, 1999, available <http://www.cs.utexas.edu/~csed/FM/docs/FMFramework.pdf>.
- [44] Wing, J., Woodcock, J., and Davies, J. (Eds.) FM'99 – Formal Methods: World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999, Proceedings, Vol. I, LNCS 1708, Vol. II, LNCS 1709, Springer-Verlag, September 1999.
- [45] Wing, J., Weaving Formal Methods into the Undergraduate Computer Science Curriculum, Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology (AMAST) 2000, Education Day, Iowa City, Iowa, US, May 20-27, 2000, available <http://www.cs.utexas.edu/~csed/FM/docs/Wing-abstract.pdf>
- [46] World Library for Formal Methods, available <http://archive.comlab.ox.ac.uk/formal-methods.html>

Appendix A: Advisory Board Members and Materials Contributed

- **Jonathan Bowen**, South Bank University, London, UK
 - Position Statement [5]
- **Dan Craigen**, ORA Canada, Ottawa, Ontario, Canada
 - Position Statement [9]
- **Kathi Fisler**, Worcester Polytechnic Institute, Worcester, MA USA
 - Position Statement [12]
 - Joint position statement: *Integrating Logic into the Computer Science Curriculum* [2]
- **Susan Gerhart**, Embry-Riddle Aeronautical University, Prescott AZ USA
- **David Gries**, University of Georgia, Athens GA USA
 - Position Statement [19]
- **Ebba Thora Hvannberg**, University of Iceland, Reykjavik, Iceland
 - Position Statement [22]
- **Randolph Johnson**, National Security Agency, Fort Meade MD USA
 - *The State-of-the-Art in Formal Methods*, January 1998 [1]
 - *Formal Methods Framework, final month status report* [43]
 - Position Statement [25]
- **Peter Gorm Larsen**, IFAD A/S, Odense, Denmark
 - Position Statement [27]
- **J Strother Moore**, University of Texas at Austin, Austin, TX USA
 - Position Statement [32]
- **Lesley Semmens**, Leeds Metropolitan University, Leeds, UK
 - Position Statement [38]
- **Moshe Vardi**, Rice University, Houston, TX USA
 - Joint position statement: *Integrating Logic into the Computer Science Curriculum*. [2]
- **Jeannette M. Wing**, Carnegie Mellon University, Pittsburgh, PA USA
 - *Weaving Formal Methods into the Undergraduate Computer Science Curriculum* [45]
- **J. C. P. Woodcock**, Oxford University Software Engineering Centre, Oxford, UK

Appendix B: Top-level Mapping between Knowledge Areas in Computing Curricula 2001 [8] and Computing Curricula 91 [41]

CC 2001 Knowledge Area and Number of Core Hours	CC 91 Knowledge Area
DS: Discrete Structures (40)	None (<i>refer to discussion in Section 3</i>)
PF: Programming Fundamentals (59)	AL: Algorithms and Data Structures
AL: Algorithms and Complexity (31)	AL: Algorithms and Data Structures
PL: Programming Languages (6)	PL: Programming Languages
AR: Architecture (36)	AR: Architecture
OS: Operating Systems (14)	OS: Operating Systems
HC: Human-Computer Interaction (6)	HU: Human-Computer Communication
GR: Graphics and Visualization (5)	HU: Human-Computer Communication
IS: Intelligent Systems (10)	AI: Artificial Intelligence and Robotics
IM: Information Management (10)	DB: Database and Information Retrieval
NC: Net-Centric Computing (15)	OS: Operating Systems
SE: Software Engineering (35)	SE: Software Methodology and Engineering
CN: Computational Science (0)	NU: Numeric and Symbolic Computing
SP: Social and Professional Issues (16)	SP: Social, Ethical and Professional Issues

Appendix C: Specific Topics Correlated with Formal Methods in Computing Curricula 2001 [8] and Computing Curricula 91 [41]

CC 2001 Topic	CC 91 Unit
DS1: Functions, relations, and sets	None (<i>refer to discussion in Section 3</i>)
DS2: Basic logic	None
DS3: Proof techniques	None
DS4: Basics of counting	None
DS5: Graphs and trees	None
PF1: Fundamental programming constructs	None
PF2: Algorithms & problem solving	AL8: Problem-solving strategies
PF3: Object-oriented programming	PL11: Programming paradigms
PF4: Fundamental data structures	AL1: Basic data structures
PF5: Recursion	AL3: Recursive algorithms
AL5: Basic computability theory	AL7: Computability and undecidability
AL6: The complexity classes P and NP	AL5: Complexity classes
AL7: Automata theory	PL7: Finite-state automata and regular expressions PL8: Context-free grammars and pushdown automata
PL1: History of programming languages	PL1: History and overview of programming languages
PL3: Virtual machines	PL2: Virtual machines
PL4: Introduction to language translation	PL9: Language translation systems
PL5: Language translation systems	PL9: Language translation systems
PL6: Type systems	PL3: Representation of data types
PL7: Models of execution control	PL4: Sequence control
PL8: Declaration, modularity, and storage management	PL6: Run-time storage system
PL9: Programming language semantics	PL10: Programming language semantics
PL10: Programming paradigms	PL11: Programming paradigms
AR1: Digital logic and digital systems	AR1: Digital logic
AR6: CPU implementation	AR1: Digital logic
OS2: Concurrency	OS3: Process coordination and synchronization
OS3: Scheduling and dispatch	OS4: Scheduling and dispatch
OS6: Security and protection	OS8: Security and protection
OS8: Real-time systems	OS10: Distributed and real-time systems
HC2: Modeling the user	HU1: User interfaces
GR3: Modeling	HU2: Computer graphics
IS2: Search and optimization methods	AI2: Problems, state spaces, and search strategies
IS3: Knowledge representation and reasoning	AI2: Problems, state spaces, and search strategies

Appendix continues on the next page

**Appendix C, continued: Specific Topics Correlated with
Formal Methods in CC 2001 and CC 91**

CC 2001 Topic	CC 91 Unit
IS6: Machine learning	AI2: Problems, state spaces, and search strategies
IS7: Natural language processing	AI1: History and applications of artificial intelligence
IS10: Knowledge-Based systems	AI1: History and applications of artificial intelligence
IM2: Data modeling and the relational model	DB2: The relational data model
IM3: Database query languages	DB2: The relational data model
IM4: Relational database design	DB2: The relational data model
IM5: Transaction processing	DB1: Overview, models, and applications of database systems
IM6: Distributed databases	DB1: Overview, models, and applications of database systems
IM7: Advanced relational database design	DB2: The relational data model
NC2: Communication and networking	OS9: Communications and networking
NC7: Distributed systems	OS10: Distributed and real-time systems
SE2: Software requirements and specifications	SE3: Software requirements and specifications
SE3: Software design and implementation	SE4: Software design and implementation
SE4: Verification and validation	SE5: Verification and validation
SE5: Software tools and environments	SE2: The software development process
CN1: Numerical analysis	NU2: Iterative approximation methods

Appendix D: Responses to Conference Survey on Importance of Formal Methods

The working group polled its colleagues at the ITiCSE 2000 conference as to how they would assess the importance of formal methods in the different knowledge areas. Participants could view a poster with the current state of the areas and their topics in CC 2001 (similar to Appendix B), with those topics that correlated to formal methods highlighted. The participants were also able to read the working definition of formal methods discussed in Section 2, as well as a wide variety of definitions drawn from the literature. In order to complete the survey, participants were asked to assign a numerical ranking to each knowledge area using the following scale:

- 1– *It would be a stretch to use formal methods in teaching this area.*
- 2 – *This area could be taught using formal methods.*
- 3 – *This area should be taught using formal methods.*
- 4 – *This area can't be taught without using formal methods.*

Thus, a rating of 2.5 for a knowledge area could be construed as a neutral stance about the applicability of

formal methods. The number of respondents was approximately ten.

The results of the survey showed that the respondents feel there is a high degree of correlation between the model curricula and the importance of formal methods. In particular, the attendees polled corroborated the importance of formal methods in the Software Engineering area with a median of 3.8/4.0. In fact, the median rating for Computational Science was 3.9. This supported the correlation posited in CC91, in spite of the fact that many formal methodists do not usually have continuous mathematics in mind when they refer to formal methods. Finally, while the number of respondents is not significant, it is interesting to note the relatively low rating assigned to the Information Management knowledge area. This would seem to indicate that many faculty either are unaware of, or do not assign strong weight to, the central role played by mathematical logic in the relational calculus formal query language.

CC 2001 Knowledge Area	Median Rating in Formal Methods Survey (4-point scale as given above)
Discrete Structures	3.8
Programming Fundamentals	2.9
Algorithms and Complexity	3.7
Programming Languages	2.9
Architecture	2.5
Operating Systems	2.9
Human-Computer Interaction	1.4
Graphics and Visualization	3.2
Intelligent Systems	3.2
Information Management	2.0
Net-Centric Computing	2.7
Software Engineering	3.8
Computational Science	3.9

Appendix E: SWEBOK and Formal Methods

SWEBOK Knowledge Area	Applicable Formal Methods
Software requirements	formal domain modeling formal requirements specification analysis diagrams - data/control flow, entity-relationship, object diagrams
Software design	formal design specification design diagrams - structure charts, object diagrams, state diagrams program design languages
Software construction	algorithm/complexity analysis data structures detailed design formalisms (e.g., pre/post conditions, invariants, design tables and diagrams) program syntax and semantics
Software testing	program flow diagrams, case construction, specification decomposition into cases, reliability and coverage arguments
Software maintenance	(any FM used in initial development)
Software configuration management	configuration process diagrams
Software engineering management	task schedule diagrams (PERT, Gantt)
Software engineering tools and methods	FM tools -analysis and design tools, compilers, type/domain checkers, animators, model checkers, theorem provers, test case generators
Software engineering process	formal process modeling
Software quality	formal analysis and verification - symbolic execution, state machines, model checking, theorem proving, "proof by team checking"

Appendix F: Responses to FM Tool Survey

<p>1. What software have you used in teaching formal methods?</p> <p><i>(Numbers in parentheses indicate the number of responses for each tool; all others represent a single response.)</i></p> <ul style="list-style-type: none"> • Z (including Z Specific Formaliser (3), CadiZ (3), Fuzz (2), Z/EVES, ZTC, and Z Browser) • B (B Tool(2)) • SDL (Telelogic TAU) • Lotos (SEDOS, TOPO, LOLA) • PVS • VDM (ToolBox Lite) • CSP (Kramer & McGee (KM) model checker; FDR model checker) • Design CPN (Concurrency Workbench) • ML • No software used (5)
<p>2. How have you used the software in your teaching?</p> <p>a) taught its use as a tool for doing formal methods in the case of: Z Specific Formaliser (3), CadiZ(2), Fuzz(2), Z/EVES, TOPO, LOLA, PVS, Tool, KM model checker, Concurrency Workbench, ZTC</p> <p>b) used it as a tool for teaching/learning the concepts of FM in the case of: Z Specific Formaliser (2), CadiZ, Telelogic TAU, SEDOS, TOPO, Fuzz, Z/EVES, PVS, ToolBox Lite, B Tool, Concurrency Workbench, FDR model checker, Fuzz, Z Browser</p>
<p>3. At what levels have you used the software?</p> <p>a) undergraduate courses: Z Specific Formaliser (2) , CadiZ, Telelogic TAU, SEDOS, TOPO, ToolBox Lite, B Tool, KM model checker, Concurrency Workbench, Fuzz, ZTC</p> <p>b) postgraduate courses (i.e. after the undergraduate degree): B Tool, Telelogic TAU, SEDOS, TOPO, ToolBox Lite (M.Sc. Conversion), Concurrency Workbench, FDR model checker, ZTC</p>
<p>4. What resources have you used in support?</p> <ul style="list-style-type: none"> • textbooks: <ul style="list-style-type: none"> • Turner, "Using Formal Description Techniques" • Jacky, "The Way of Z" • Potter, Sinclair & Till (2), "Introduction to ... Z." • Lightfoot, "Z" • Spivey, "Z Reference Manual", • Woodcock and Davies, "Using Z" • Currie, "The Essence of Z" • Diller (2), "Z" • Ellsberger <i>et al.</i> "SDL" • Fitzgerald & Larsen, "Modelling Systems" • Unknown (3) • on-line tutorials/manuals: Fuzz, TOPO, LOLA, Z/EVES, PVS; Own materials, CadiZ; No such material used (3); own handouts (3) • other web based resources: Z, CadiZ • mailing lists: Z/EVES, PVS