Advertising Formal Methods and Organizing their Teaching: *Yes, but …*

Dino Mandrioli[1]

[1]Dipartimento di Elettronica e Informazione, Politecnico di Milano,
P. L. Da Vinci 32, 20133, Milano, Italy
mandrioli@elet.polimi.it
www.elet.polimi.it/~mandriol

Abstract. This position paper aims to address most of the "challenges" suggested by the conference's CFP plus a few others. The style is deliberately informal and colloquial, occasionally even provocative: for every examined point some obvious agreement is given for granted but a few, more controversial, "counterpoints" are raised and hints are suggested for deeper discussion. At the end a constructive synthesis is attempted.

1. Preamble: The Essence of Formal Methods: What are they?

To optimize the organization of the teaching of formal methods (FMs) and their chances of gaining acceptance, we must first agree on some basic terminology. I very much regret that most terms are often used with fuzzy, often context-dependant, sometimes even contradictory, meanings. For instance, the term "verification" is often used as a synonym of –possibly formal– correctness proof; consequently it is *opposed* to testing. This is particularly unfortunate: first, because, in the common understanding of non-specialized people, the term "verification" is much comprehensive and includes the application of any technique aimed at guaranteeing that an artifact satisfies its requirements and goals; second, opposing different techniques with a common goal fails to show and to exploit their complementarity[1].

The term FMs itself requires some preliminary agreement on its meaning. There is now some wide consensus on claims such as

- "FMs are not mathematics but do exploit it"
- "FMs are not theoretical computer science, or theory of computation, but do exploit it"
- "FMs for Computer and Software Engineering (CSE) are rooted –mainly– in discrete mathematics whereas traditional engineering (civil, industrial, electrical, … engineering) mainly exploits continuous mathematics"

[1] This habit of using general, widely known terms, with specialized, context-dependant, meaning, occurs unfortunately in many communities: consider, e.g., the use of terms "framework" and "pattern" in the object-oriented culture.

- "Formal *Methods* should not be confused with formal *models*. Although they *use* formal models, they include much more: mainly guidelines to apply models at best to practical problems; tools supporting them, etc.

On the other hand different interpretations of the term FMs range, roughly speaking, between two extreme positions such as:

- "A FM must be fully formal", i.e., it must drive the user through the whole life cycle so that every artifact, from requirements specification to executable code, is documented through a formal syntax and semantics and every step is formally proved correct.
- Any "level of formality" is acceptable: for instance, using a formally defined graphical syntax can be considered as a FM even if a rigorous semantics for the adopted notation is lacking; also, *some* steps of the design process can follow formal guidelines but others can be carried over in a more informal way.

A typical example of such a lack of general agreement between the above positions is provided by UML, which is considered by many practicioners as a FM, whereas many theoreticians do not recognize it at all as such.

Personally, I am in favor of a fairly comprehensive and "liberal" definition of FM, as any method that in some way exploits the use of formalism. In particular, I recommend an *incremental attitude* to the application –and teaching– of FMs: moving from informal documentation to UML is an important initial step in the path that leads to a full exploitation of FMs in industrial activities; as well as, further on, enriching UML with –any kind of– semantic formalization, and augmenting refinement steps by formal correctness proofs. Such a liberal, or incremental, or "modest" attitude is also recommended in the literature as "lightweight formal methods" (see, e.g., [1], [2]).

2. Advertising and Promoting FMs: Yes, but …

Nowadays, a great amount of effort must be put in the advertising and promotion of any "product"; culture is no exception and the old times of academia compared to an "ivory tower" are perhaps buried forever. Even scientific research requires a lot of "marketing" and publishing deep technical results on major archival journals is by no way warranty of success. Thus, there is now a fairly general consensus on claims such as the following ones:

- At the root of much reluctancy against FMs there is often a "mathfobia", typical of many students.
- Preliminarly to teaching FMs we should strongly *motivate* their use by emphasizing the risks and the costs of poor quality products and the benefits that can derived by the application of FMs. This applies not only to industrial "decision makers" but even to students who are more and more reluctant to accept a course "just because it is proposed by the university".
- "Fun" in the application of FMs should be emphasized through several means (amusing examples, games and competition, user friendly tools, etc.); tedious mathematical details should be avoided and possibly hidden.
- Tools should be used to relieve the user from many, often boring, clerical details, to make the whole process more efficient, reliable and productive.

- As a particular case joining the two above points, so called "push button" tools such as those based on model-checking are strongly recommended since, in principle, they allow the user to be concerned exclusively with the *writing* of properties to be analyzed, leaving all the burden of their verification to automatic tools.

However, most of the above claims hide some subtle traps that could lead to even counterproductive actions. Thus, in their application, one should also keep in mind the following "counterpoints":

- Do not "oversell" FMs; avoid "miracle promises" such as "FMs help producing bug-free software"; "FMs make testing useless"; etc. Such claims can be easily verified as false or at best as overstated and consequently produce the opposite result[2].
- Tools should not be advertised as a "panacea": even outside the FM realm many failures happened due to the fact that managers erroneously hoped that just buying state-of-the-art tools guarantees innovation and improvement in the production process. Also, in some cases, too early distribution of prototype tools could produce a global rejection of the underlying method only because of the poor quality of, say, tool's interface.
- In particular "push-button" itself maybe an example of overselling: in fact, most often such tools are based on brute force algorithms that "do not scale up", i.e., whose complexity becomes soon intractable with the increase of problem size; thus, in order to obtain practical results, users must indeed apply some intellectual skill.
- Not only the contents and the style of the teaching, but even the advertising arguments should be carefully tailored to the particular audience. There are major differences not only between university students and industrial practicioners, between engineers and managers, but even between graduate and undergraduate students, between young and experienced engineers (the former are usually more fresh-minded and open to novelties; the latter only accept minor changes to their current habits); between software engineers and application domain experts (both should be acquainted with FMs but in different ways), etc.
- A particular class of "students" who are often even more reluctant to change their habits than "official students" is the class of … teachers, both in the high schools (often the deprecated mathfobia is rooted in bad teaching of mathematical bases at junior schools) and even in universities (where far too often professors of Department X ignore and/or disparage the discipline of Department Y. FMs teachers are not absent from this class …).
- If some "thresholds of commitment and skill" are not guaranteed it is better to downgrade the objective or even to give it up at all. This general claim has several particular instantiations. For instance:
  - If within an industrial environment there is not enough interest and resource commitment in the training of FMs (typically: short term delivery deadlines repeatedly overwhelm time scheduled for training sessions) further insisting may become counterproductive.

[2] Some classic references about "selling and overselling FMs" are [3], [4], [5].

- Students' mathfobia can and should be fought with "fun" and other tools but not up to the point of hiding the fact that some mathematical skill is a necessary prerequisite for successful application of FMs. Even without going to extreme positions such as Dijkstra's [6], FMs teaching should avoid oversimplified examples that hide the technical difficulties of intricate cases[3].

- (With main reference to the case of teaching to industrial people). In general, "teaching" does not consist exclusively in explaining a topic; a formidable teaching aid is "working together". Building joint teams of application experts and FM experts often produces the best results. This practice should not be applied only during the training activity: in some cases the level of expertise that is needed is such that temporarily "hiring" specialized consultants is more effective than insisting in teaching highly sophisticated technology to not-sufficiently-motivated-or-skilled people. For instance, in several cases of industrial environments, application domain experts could and should be involved in the production of specification documents, but the application of powerful but difficult formal verification techniques such as theorem proving should be left to FMs experts.

3. So what? *Integrating* the teaching of FMs within engineering curricula

Let me now address the issue of organizing the teaching of FMs. The foundations over which I build my proposal are the following (as usual, some of them are widely shared, others are perhaps more controversial):

- FMs are a very general engineering principle; they have always been a major tool to achieve rigor in analysis and design (one can be rigorous without being formal, but this is usually more difficult). FMs *in general* should be well mastered within any field of science and engineering.
- FMs are, however, "context-dependant": traditional science and engineering (physics, biology, mechanical, industrial, civil engineering, …) have from a long time their own well-established FMs. They are mostly rooted in continuous mathematics. There is no doubt that computer science and engineering have developed their own FMs and that they are –much more, *but not exclusively*– rooted in discrete mathematics and in mathematical logics. It is also a(n unpleasant) fact that FMs are much less exploited within Computer Science and Engineering (CSE) than in other older fields.
- As an obvious consequence, the teaching too of FMs must be somewhat context-dependant: it certainly depends on the specific application field; but it must also

[3] As a "counter-counter-point" the above argument should not be intended as a generic blame of so-called "toy problems" as opposed to "real-life" problems. In my opinion, *well-designed toy problems* are often even a better teaching aid than real-life projects since they help focusing attention on –few, selected– critical aspects, whereas real projects often bury subtle points under a mass of clerical details.

depend on the environment within which it occurs: teaching FMs within university curricula may be quite different than teaching them in an industrial environment, perhaps in a few intensive weeks with highly specialized goals and focus.

In this paper my attention is mainly centered on university curricula[4].

- However, I consider particularly unfortunate the present state of the art of the organization of university curricula, where, at every level, specialization far overwhelms generality and cross-fertilization among different disciplines. For instance, despite the fact that computer-based applications are mostly part of heterogeneous systems (plant control systems, banks, embedded systems, etc.) not only Computer Engineering (CE) curricula are quite distinct from other engineering curricula, but we have major differences between Computer Science (CS), CE, Software Engineering (SE), etc.

  FMs teaching, unfortunately, is no exception to such an overspecialization: in some cases there have been even proposals of FMs curricula *per se*, forgetting that FMs are a *means* for rigorous and high quality design, *not a goal*; also, many FMs courses focus on single, often fashionable, methods (e.g. model-checking, or theorem proving) failing to show commonalities in their goals and complementarities in their approaches.

In conclusion, I believe that engineering curricula should first emphasize the general usefulness and the common principles of FMs *per se*; a strong interdisciplinary background should also be shared by almost all engineering curricula: CSE majors should know enough of the FMs of traditional engineering (e.g., models for electric circuits) *and conversely* (many non-computer-rooted engineering curricula wrongly consider computer science just as a tool for numerical computations, access to Internet, etc, ignoring its fundamental richness of concepts and principles.)

Only later, the context-dependant part of FMs should be tailored towards the specific needs of the application field.

To state it in another way, the teaching of FMs should be well-integrated in any engineering –and not only software engineering– curriculum and cannot be addressed by itself.

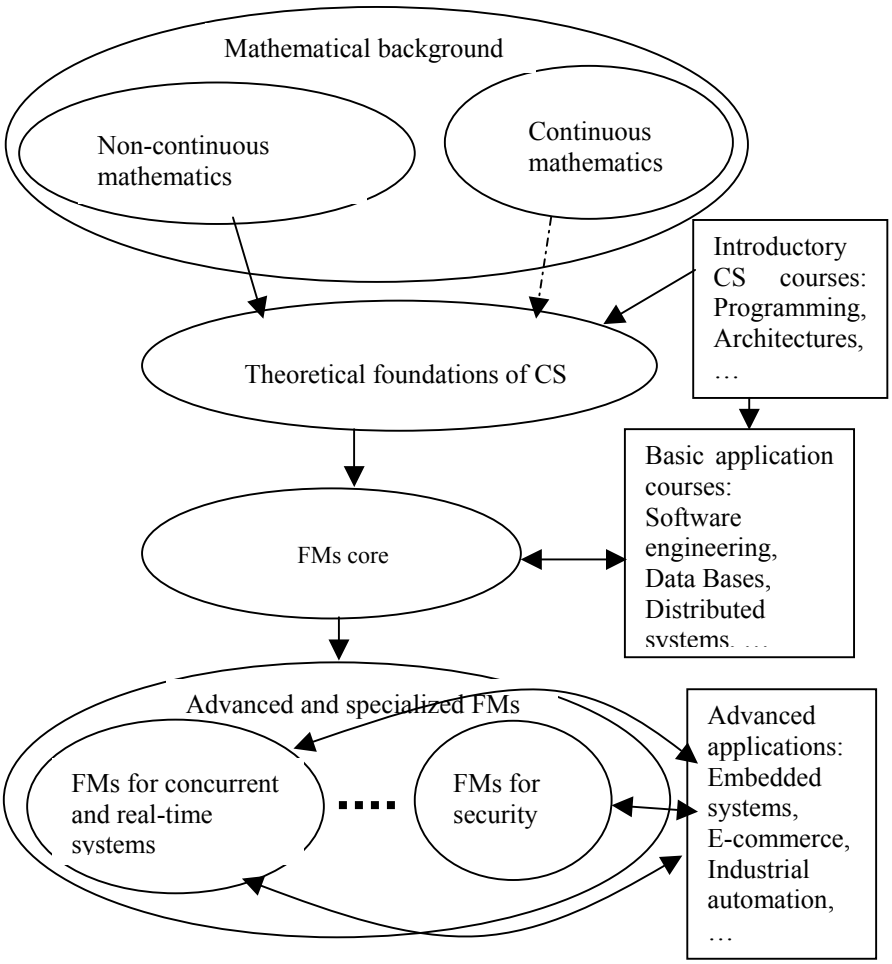
Next, Section 3.1 suggests an example of how the teaching of FMs could be "plugged" into a curriculum for CSE majors, which is the main focus of this paper. To complete the picture, Section 3.2 also provides a few hints on the teaching of (CS) FMs within non-computer engineering curricula.

3.1 A FM Track within CSE Curricula

Figure 1 provides a synthetic view of the way the teaching of FMs should be integrated within a CSE curriculum. Then, a few explanations and comments are given for some distinguishing elements. Notice that Figure 1 does *not* display the *full structure* of an ideal CSE curriculum: it only deals with the integration of FMs

[4] Some personal experiences and lessons learned in the introduction of FMs-based practices in industrial environments are reported in [7] and [8].

teaching within it; essential –in my opinion– topics such as basics of physics, chemistry, industrial and civil engineering, economics, etc. are omitted.



Legend: single-headed arrows denote a precedence relation; double-headed arrows denote mutual dependencies and benefits; dashed arrows denote weak precedence (general culture, beneficial but not mandatory).

Figure 1 A synthetic view of the relations between FMs and other typical topics of CSE.

3.1.1 Mathematical Background

Every scientist and engineer should have a *strong background* both in continuous and non-continuous (this term is more comprehensive than "discrete") mathematics. Strong background does not mean "many topics" but essential topics well-rooted and

well understood. I insist that even CSE students must have such a background on mathematical analysis and calculus: the concept of continuity is fundamental for our community too! Certainly, the impact of non-continuous mathematics is more direct. It should include: elementary algebra and set theory; basics of mathematical logics (propositional and predicate calculus); a little of combinatorics. All in all mathematical background should require *at least* 5 one-semester courses (for CSE majors: 3 courses on non-continuous and 2 on continuous mathematics. A *very minimum* could be 2+2).

Remarks

There is a tendency, mainly imported from the US, to give little and fairly superficial mathematical background at the undergraduate level; later on, graduate and more talented students –it is claimed– will be able to go deeper into mathematical concepts. I am against this approach, as far as it concerns the *mathematical foundations*: foundations must be understood in depth from the beginning to help understand even trivial applications (e.g., the execution of a machine instruction); advanced mathematical topics can and should be taught at the graduate level as well as advanced applications.

Another major hole in the normal way of teaching mathematics is the lack of training in *building deductive proofs*. Often many –maybe complex– proofs are given but the students only have to learn and repeat them –and, sadly, they do so by heart, without even trying to *understand* them; instead, little or nothing is done to increase their skill to develop their own proofs. This unfortunate circumstance is probably the main reason why formal correctness proofs are considered as the most inapplicable formal technique.

3.1.2 Basics of Theoretical Computer Science

Notice that I distinguish between *theoretical foundations* of CS (TFCS) and –basic– FMs: both contribute to the theoretical core of a CSE curriculum, but they are two different things: *TFCS is about models and their properties, FMs are already a first* ***application*** *thereof to practical problems*.

TFCS is often identified with Automata and Formal Language Theory. I disagree, although certainly such topics are important TFCS. My favorite –one-semester– course on TFCS includes:

- **Models for CS**: *simple* automata and their *basic* properties; simple grammars; (usually I cover much less on these topics than traditional textbooks); use of mathematical formulas (essentially first-order formulas) to formalize simple systems (e.g., formal languages, but also every-day-life objects: railroad crossing systems, elevators, to mention "classical examples"; it is nice and useful to exploit simple examples of hybrid systems, to give the message that often for some system components continuous models are suitable, whereas for others discrete models fit better).
  The fundamental skill of this part is the ability to formalize reality much more than deducing sophisticated mathematical properties from other mathematical properties.

- **Basics of computability**. Despite a few "revolutionary claims" I still believe that the halting problem plays a fundamental role in this topic. It must serve, however, the purpose of going deep into "what can be done and what cannot be done by a computer". In first courses in CS and programming I always get questions whose answer is "there is no algorithm to build algorithms to solve a given problem"; but I also add "I will be able to explain you better this claim in the TFCS course."
- **Basics of complexity theory**. This topic is fairly controversial, at least in Italy: our students usually attend courses on algorithms and data structures (where, typically, they learn tree-managing, sorting, …) *before* TFCS. Thus, the goal of resuming the complexity issue here is not to teach them to understand whether an algorithm is $O(n^2)$ or $O(n.\log(n))$; rather, it is to teach them to understand when a logarithmic cost criterion is better than a uniform cost criterion and why in some cases the "poor Turing machine" is a better complexity model than the powerful Java Virtual Machine or than counting the statement execution in a C program.

Dave Parnas [9] seems to be in agreement with the above view.

3.1.3 Core FMs topics

Not surprisingly, "core FMs" coverage should go somewhat in parallel with the basics of design courses such as Software engineering, Hardware design, Operating systems, etc.

Here is a personal proposal for its structure:

- **FMs for system specification**
  This should exploit the knowledge of basic models such as automata and logic formulas to come up with formalization and analysis of real systems. It should include some examples of requirements elicitation.
  Two important remarks are in order:
  - Accent should be on *methods* rather than on *languages*: e.g. languages such as Z or VDM can certainly be used as vectors to illustrate the methods, but methods should be the real focus just as in programming courses the accent should be –but often is *not*– on programming principles, not on C rather than Pascal or Java.
  - I emphasize the term *system* specification as *opposed to software* specification, the latter being a particular case of the former.
- **FMs for design**
  Here clearly, the main keyword is *refinement*. Again several linguistic choices are possible (e.g., B) but accent should be on methods.
- **FMs for system verification**
  Various verification methods should be reviewed and formally treated:
  - "Traditional" formal correctness proofs certainly deserve attention. But FMs for verification are certainly *not only proofs* (this is a common and still hard to fight misunderstanding).
  - Model checking –of course!– should be presented as a major "winner" among FMs.

- FMs do support also verification techniques traditionally considered as empirical and opposed to FMs: FMs for the derivation, evaluation, … of testing is a main argument on this respect.

Remarks

- Often "practical courses" such as Software engineering do cover part of the above topics, mainly when the teachers have some "sympathy" with FMs. As an obvious consequence a problem arises of coordination and borderline when plugging a FM track within a CSE (and not only) curriculum. For instance, about system specification, a SE course could introduce to UML and to its use, and a well coordinated course on FMs (see the first horizontal arrow in Figure 1) should mention motivation, problems, and approaches to make a UML specification fully or partially formal.
- As it happens with most "life-cycles" the above structure reflects different phases of system development that in practice should not be necessarily applied in a "waterfall" style. For instance, some amount of verification (V&V) must be applied during requirements elicitation. Of course it is up to the teacher to decide the best organization of the topics.

In my opinion the topics addressed in this section should approximately constitute the part of a FMs track plugged into an undergraduate curriculum.

Of course, –we certainly agree within the FM community– such a track (not my own proposal but *any* track in FMs) should not be an elective track for a minority of theoretically oriented students but should be a core part of the whole CSE curriculum.

The next section outlines, instead, more advanced topics in FMs, that should probably be covered at a graduate level.

3.1.4 Advanced FMs Topics for Specialized Applications

There are, of course, several advanced topics in FMs, usually associated in a natural way with emerging or specialized application fields. In this paper, it is perhaps not necessary to go deep into such an issue.

Typical examples of such topics are:

- Models and methods for concurrent and/or distributed and/or real-time systems
- Models and methods for Artificial Intelligence
- Models and methods for security
- …

What is most important here –nowadays even more than in the past– is a strong emphasis on critical evaluation, comparison, and *integration* between different methods, which reflect different requirements of more and more integrated applications. A fashionable example of these days is how fault tolerance, real-time, security, all concur to make a system *dependable* as a whole.

In this case too, strong coordination between FMs courses and corresponding applicative courses is demanded. Even more, in such specialized fields one could merge into a single course the treatment of the formal model and its exploitation to

the application field; although with this approach there could be the risk of (over)specialization, thus missing important integration and cross-fertilization chances.

3.2 On Possible FMs Tracks within non-CSE Curricula

This paper focuses on integrating a FM track within a CSE curriculum. However, I also insist that a suitable track devoted to "CSE FMs", i.e., methods rooted into non-continuous mathematics and devoted to computer-based applications, should be included in other science and engineering fields to complement "their" traditional FMs. The teaching of the various FMs should be somewhat symmetric: on the one side, it is important that CSE majors are exposed to some continuous mathematics and related methods; on the other side the normal CS culture that is provided to physicists, industrial and civil engineers, etc. should not be restricted to describe computers as *tools* for enhancing their job but should include fundamental *concepts* helping to *model and analyze* systems as a whole: after all most systems are hybrid: thus their global understanding requires the managing of both families of models and methods from both communities. Only later on, in the lower and more specialized phases of the design every engineer will go deeper into his own specialized formalism.

As an example, the track described above for CSE majors could be adapted (restricted) to non-majors in the following way:

- One course only for non-continuous mathematics (summarizing the main topics: some predicate calculus and a little of combinatorics and set theory).
- One course presenting a selection of FMs particularly well-suited for the main science or engineering field. Such elements could also be included within a global CS course that integrates informal and formal methods: for instance, a software engineering course for mechanical engineers could include the important issue of specifications and could present some UML possibly integrated with a little formalization: an historical fact that strongly supports such an "open attitude" towards formalisms is that Parnas' SCR has been used to do some specification analysis in cooperation with pilots [10].

4 Conclusions

There is much agreement that teaching is a crucial factor to increase acceptance of FMs in the practice of industrial projects. In this position paper I reviewed, reinforced, and complemented a few suggested clues to improve the state of the art. I also argued in favor of some guidelines that are not often agreed upon, or even adversed, in many university curricula. In essence, I argue for a strongly interdisciplinary approach in the engineering fields as opposed to highly specialized curricula, and for a FMs *track* that is strongly integrated within engineering –and not only engineering– curricula (as opposed to curricula that have FMs as their main topic, and even in the title); that exploits discrete as well as continuous mathematics;

that emphasizes the application of formal models as opposed to teaching one or more specific formalism.

5 References

[1] Easterbrook, S.; Lutz, R.; Covington, R.; Kelly, J.; Ampo, Y.; Hamilton, D.. "Experiences using lightweight formal methods for requirements modeling". *IEEE Transactions on Software Engineering,* Vol. 24, no. 1, pp. 4-14, 1998

[2] Saiedian, H., Bowen, J. P., Butler, R. W., Dill, D. L., Glass, R. L., Gries, D., Hall, A., Hinchey, M. G., Holloway, C. M., Jackson, D., Jones, C. B., Lutz, M. J., Parnas, D. L., Rushby, J., Wing, J. and Zave, P.. "An Invitation to Formal Methods". *IEEE Computer*, Vol. 29, no.4, pp. 16-30, 1996.

[3] Hall A. "Seven Myths of Formal Methods", *IEEE Software*, Vol. 7, no. 5, pp. 11-19, September 1990.

[4] Bowen J., Hinchey M., "Seven More Myths of Formal Methods", *IEEE Software*, Vol. 12, no. 4, July 1995

[5] Bowen J., Hinchey M., "Ten Commandments of Formal Methods", *IEEE Computer*, Vol., 28, no. 4, pp. 56-63, April 1995.

[6] Dijkstra. E.W., "On the cruelty of really teaching computing science". Communications of the ACM, Vol. 32, no.12, pp. 1398-1404, 1989.

[7] Ciapessoni E., Coen-Porisini A., Crivelli E., Mandrioli D., Mirandola P., Morzenti A., "From formal models to formally-based methods: an industrial experience", *ACM Transactions on Software Engineering and Methodologies*, Vol. 8. no 1, pp.79-113, January 1999.

[8] Cigoli S., Leblanc P., Malaponti S., Mandrioli D., Mazzucchelli M., Morzenti A., Spoletini P.: "An Experiment in Applying UML2.0 to the Development of an Industrial Critical Application", Proceedings of the UML'03 workshop on Critical Systems Development with UML, San Francisco, CA, October 21 2003.

[9] . Parnas D.L, "Software engineering programmes are not Computer science Programmes", CRL Report 361, McMaster University, Ontario, 1998.

[10] Parnas D.L.,. Heninger K,. Kallander K., Shore J., "Software Requirements for the A-7E Aircraft", Naval Research Laboratory, Report no. 3876, November 1978.

6 Acronyms

CE: Computer Engineering
CS: Computer Science
CSE: Computer Science and Engineering
FM: Formal Method
SE: Software engineering
TFCS: Theoretical Foundations of Computer Science
V&V: Verification and validation