Teaching Formal Methods: An Experience Report

Mehrnoosh Askarpour[1](✉) and Marcello M. Bersani[2]

[1] Computing and Software, McMaster University, Hamilton, Canada
askarpom@mcmaster.ca
[2] DEIB, Poltecnico di Milano, Milan, Italy
marcellomaria.bersani@polimi.it

Abstract. The general attitude of students towards formal specification and verification of systems is not exactly what one could call enthusiastic. Generally, software engineering courses at universities include an introduction to specification with formal notations such as Z, Alloy, UML, etc. However, it seems that the importance of formal specification to replicate expected system behavior does not sink in as it should with the students. Moreover, other products of computer science (e.g., machine learning algorithms, robot systems deployment), rather than software, benefit from formal specification as well. This paper is a general report of our observations on teaching formal methods on undergraduate and graduate levels at Politecnico di Milano.

Keywords: Formal methods · Temporal logic · Computer science · Computer engineering · Formal specification

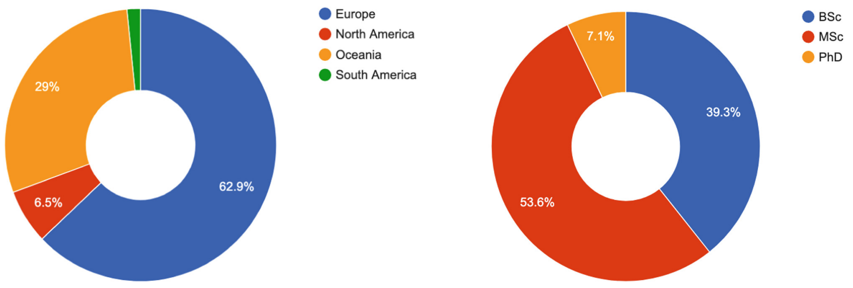
1 Introduction

Formal methods (FM) have been used in hardware and software for a wide range of applications (e.g., aerospace, transportation). Methods such as automated formal verification and model checking are used in many different research and practical areas. Hence, it is paramount for computer scientists and engineers to be aware of their benefits and apply them to problems of the real world.

Formal Methods community is very well-known, and the world congress FM is a landmark worldwide. Their affiliated Formal Methods Teaching Committee (FMTea)[1] gathered a repository of FM university courses around the world,[2] which gives a reliable estimation of the situation globally. As of April 2020, the website displays sixty-two -courses, mainly from European universities (see Fig. 1a), dispensed at the M.Sc. level (see Fig. 1b), the half of which include explicitly Logic in the list of topics of the course. Moreover, only ten of the reported courses include Temporal Logic in their program.

[1] fmeurope/teaching.
[2] fme-teaching/courses.

J.-M. Bruel et al. (Eds.): FISEE 2019, LNCS 12271, pp. 3–18, 2020.
https://doi.org/10.1007/978-3-030-57663-9_1



(a) University FM Courses Divided by Geographical Area

(b) University FM Courses Divided by Grade

Fig. 1. A visual representation of the current state of Teaching Formal Methods around the world, based on an study done by Formal Methods Teaching Committee.

To the best of our knowledge, and based on our observations at Politecnico di Milano and our discussions with our colleagues elsewhere, the participation of students in FM courses is decreasing, and their interest in doing thesis or research about FM concepts is alarmingly low. For example Politecnico di Milano has around 200 graduates every year[3] in Master of computer science and engineering of which only on average 30 students participate in our FM course—which we later report in this paper—annually.[4] We asked those 30 attendees via a questionnaire if they are interested in doing their master thesis in this area, and only about 11% of them had positive answers. Based on the same questionnaire, about 33% of students have some background or interest in the subject, which means one third of those who are interested in FM still won't pick it as their thesis topic.

Da Rosal [48] argues that the reason why students find FM courses challenging lies in their mathematical background from high school. This opinion has been raised even before [8,31,48] and highlights the need for a fundamental change in high school education and more focus on discrete and applicable mathematics, instead of pedagogical issues. Gibson [25] blames neglecting case studies and illustrative examples during teaching; Robinson [47] mentions the lack of visibility of industrial use; Mandrioli [38] blames miscommunication, and fuzzy terminology, and suggests the use of "push button" and user-friendly tools; Buote [11] claims that tools could be misleading if mathematical fundamentals and concepts are not clearly understood. Others point out student's desire of achieving quick results and bypassing important design steps [45] or their drift to virtual reality, rather than modeling [27].

Just like FM courses, any other course in computer engineering/Science has its difficulties. Students, though, do not run away from them because they view

[3] Average computed by considering the last 5 years.

[4] All the statistics and percentages reported throughout the paper are rounded and therefore are approximate values.

Table 1. The second column categorizes the papers in proceedings of Formal Methods in Computer Science Education 2008, Formal Methods Teaching 2004 - 2009 - 2019, based on the tools and concepts they used. The third column lists the most popular tools listed in FME Education Course Database (fme-teaching/courses).

Tools	Papers	FME course database
JML	[3,34,44]	-
Ada	[13,36]	-
Abstract state machines	[10]	-
Eiffel	[43]	-
B, Event-B	[1,14,15,26,28,34,46,46]	-
Hoare's Logic	[29,53,54]	-
Alloy	[12,41,50,51,55]	8 entries
Logic	[7,20,33,35,42,52]	-
UML	[16,24]	-
Z, Object-Z	[17,55]	-
Proof-checking	[4,9,40]	-
Model checking	[6,19,30,32]	-
Concurrency theory	[2]	-
Danfy	-	7 entries
SPIN	-	9 entries
Nusmv/Nuxmv	-	3 entries
Mini-sat	-	2 entries
Mcrl2	-	4 entries
Coq	-	7 entries

them as important and necessary for their careers. Students often think of FM courses as too theoretical and not practical. For example, when we teach them about Hoare's Logic, they straight up ask if there is any use of that at all. This shows a serious lack of prospect that will be adjusted by showing the student real-life examples of using FM in the industrial ambient or asking them to do practical projects [58]. Open-source tools such as UPPAAL [57], NuSMV [18], GreatSPN [56] are good examples of means that have industrial purposes and could be used for homework projects as well. Other tools developed for modeling and verifying complex systems, whose behavior combines discrete aspects (e.g., states and transitions) and continuous dynamics (e.g., time, continuous laws, etc.), can be found, for instance, at [23] and [21]. Section 3 explains our experience with Zot [59] on encouraging students to use formal verification for a practical project.

Table 1 envisions the current state of FM courses and their subjects, by integrating the teaching concepts extracted from teaching FM proceedings, and the most popular concepts and tools highlighted in FME course data base. As it is evident from the table, the importance of teaching temporal specification and Temporal Logic to students, seems to be left out of spotlight in FM community. On the Contrary, here at Politecnico di Milano we particularly focus on teach temporal logic in addition to other concepts. Table 2 shows the profile of this course in the last ten years.

Table 2. The profile of FM course tought at Politecnico di Milano over the last ten years.

Title	Formal methods for concurrent and real-time systems
Students level	Second year of Ms.c. in computer engineering and computer science
Average number of students	30
Hours per semester	80
Course type	Not mandatory
Required background	Theoretical computer science and mathematical logic
The syllabus	The basis of FM, Bisimulation, abstraction and refinement Concurrent, distributed and real-time systems, Computation tree logic, PetriNets, Hoare's logic, Timed automata, Temporal logic (TRIO), Case studies,
Evaluation	Studying and presenting a recently published article with a topic close the material of the course, Applying Hoare's logic on an algorithm, Modeling a toy example via TRIO and verify a predefined property with a local model-checker

The rest of this paper is structured as follows: Sect. 2 discusses the importance of teaching formal specification and verification to students, Sect. 3 reports our experience at Politecnico di Milano on project-based and lecture-based teaching, and finally Sect. 4 concludes.

2 Teaching Formal Specification: Why and When?

Language

Model based approaches are fundamental for all the academic curricula, not only those belonging to scientific faculties. The act of transferring an abstract representation of facts and their relations demands non trivial capabilities. The problem is even more complex if we consider the case where the language, that supplies all the syntactical structures capturing the facts and their relations, has not been devised yet. This scenario is, of course, far from the reality that professors or lectures face in their everyday academic teaching, which is instead characterized by a different challenge.

Conveying facts and their relations through a predefined (logical) language is the actual challenge that universities try to tackle, and that requires students to learn specific capabilities, and professors and lecturers to develop suitable teaching approaches. The first capability requires the development of specific mental process that allows a subject to encode facts into sentences of the language, by only using a limited number of basic syntactical structures, each providing predefined, and formally stated, semantics. Yielding two different logic formulae that are syntactically different but semantically equivalent is very common when the same reality is analysed by two persons. Transferring an idea into a formula or expressing solutions to problems in an algorithmic manner implicates Computational Thinking, which is envisioning the reality through simpler facts and more basic relationships. Computational thinking dates back to '50, and has been nourishing an interesting debate and research among pedagogues on whether it can be counted as a basic skill that schools should teach with no distinction of study careers and since the beginning of education. The mental path that a subject follows to analyse and decompose the reality is, of course, strongly dependent on the subject's way of thinking. Additionally, the presence of a duality, or a degree of freedom in the language is sometimes a source of problems for students that are learning logics-related subjects. For example, a subject can state the relation between A and B either by using a "future-to-past" direction, or a "past-to-future" one; that is, A will cause B in the future, or B is caused by A occurred in the past.

The presence of a duality, or a degree of freedom in general, in the language is sometimes a source of problems for students that are learning logics-related subjects. This freedom of choice has negative side effects leading to a phenomenon called "Supermarket dilemma". Schwart analysed this phenomenon in [49] by providing the evidence that "Autonomy and Freedom of choice are critical to our well being, and choice is critical to freedom and autonomy". For this reason, a person who stands in front of a supermarket shelf can find the choice of a product difficult, if a decision has not been already taken in her/his mind. The same feeling is experienced by students in learning logic. Despite the formal definition of a logical language, the availability of a number of distinct ways to approach the encoding of the decomposed reality they have in mind, might hamper the construction of the sentence that should be the result of a correct composition of basic syntactical structures.

Abstraction

The correct identification of the facts and relations of the reality that matter, can be attained first by answering the question "What should I say with the formula?" and, then by pulling the essential facts and relations out of the identified answer, by using a possibly iterative process that removes unessential details. Choosing the right facts and relations is another source of problems for the students who come upon the study of logic-related subjects. The lack of capacity for abstraction is the origin and the reason why students perceive the study of logic too difficult.

The causes that might lead to a scarce development of abstraction skills are far from being obvious, though they can be classified into exogenous and endogenous ones. The former are related to the environment that surrounds a subject and where the subject intertwines relationship with other persons, such as the family, friends and the attended schools. It is well known that the environment has a tangible impact on the subjects that live therein. Schools play an important role in this game, because their programs and teaching methodologies can determine the development of abstraction skills in students. The latter ones are strongly dependent on the subject's way of thinking, and can unlikely be studied by only observing the behavior of subjects.

According to Anderson et al. [37], knowledge can be classified into four distinct classes, namely, factual knowledge, conceptual knowledge, procedural knowledge and metacognitive knowledge. Factual knowledge is the first kind of knowledge that students learn at schools, and that schools must provide, and pertains to the basic evidence of a discipline. For this reason, factual knowledge always conveys the terminology and the specific details of a subject, and supplies basic building blocks to understand the higher level relationships among the objects of a given reality. Abstraction capabilities can be nourished through conceptual and procedural knowledge, that focus on the relationship among the pieces of a larger structure and how to solve problems. The pillars characterizing conceptual knowledge are: Knowledge of classifications and categories, Knowledge of principles and generalizations and Knowledge of theories, models, and structures. The lack of abstraction capabilities in students can, therefore, originate from the scarce presence of the conceptual dimension of knowledge, which is not adequately conveyed by suitable teaching program during the school.

Temporal Logic

Students often do not comprehend the meaning of the syntactical structures of the language because they do not grasp the overall context in which the language is defined. Missing the general picture is the main reason of a weak comprehension of the basic evidence conveyed through the factual knowledge.

Logical languages were conceived by humans to capture very diverse aspects of the reality. Modal logic is an example, and is one of the main families of languages that are taught during academic careers at M.Sc. of Computer Science. It includes several distinct languages that can be used to specify the "mode" (or "modality") that qualifies the validity of the sentences. Modalities can be intuitively explained by analogize them to adverbs, as they change the truth value of a sentence as adverbs refine the verbs in natural languages. For instance, the sentence "it will rain tomorrow" does not express how the rain will be and how likely the rain will occur. By using a "modality", or an adverb in this case, the information that the sentence conveys can be refined; for instance, "it will surely rain tomorrow" communicates a more precise information about the rain.

The reception of logic by Computer Science is rather recent, despite Computer Science is tightly connected with Mathematics. Amir Pnueli pioneered the use of a specific class of Modal logic, called Temporal logic, in Computer Science. Temporal logic allows for representing, and reasoning about, facts qualified in terms of time. For this reason, the language is equipped with two modalities that characterize the occurrence of a fact, that are the "eventually" and "always" modalities. The semantics of the language is based on the intuitive order of time, that allows humans to distinguish the notions of "before" and "after", that is, in particular, if an event, or a fact, holds true before or after a different one. Temporal logic includes, in turn, several families, but in its original definition, it allows one to state, for instance, that if something happens now, then in the forthcoming future something else will occur.

Temporal logic in Computer Science is adopted to express the desired properties of systems or to specify entirely their behavior over the time. Writing formulae that specify how a system behaves is fundamental to precisely capture the functionality that the system should exhibit, but also to verify if the designed system can actually behave as intended. For this reason, temporal logic became rapidly one of the baseline tools to perform requirement specification and analysis, during the Requirement Engineering phase of a project, and to verify certain properties of the model of a system through Formal Verification.

The notion of time in Temporal logic has been refined over the years, moving from a simple order to trees of realizable futures, or to dense time. According to Furia et al. [22] there are many issues that temporal specification faces once a system is subject to be modeled. It is important to clarify if a system is more compatible with continuous or non-continuous time models. For example, manifesting the behaviors of a model with certain characteristics is presumably simpler considering continuous time domain. It is also important to figure out if a system may function in a finite, infinite or periodic time window.

Moreover, to choose a suitable formalism for modeling purposes, one need to realise if temporal characteristics of the system concern only order of events (e.g., event A happens before event B) or also metric constraints are important (e.g., event B must happen exactly three time units before event A). One needs to choose between linear (i.e., sequences of states of the model) and branching time (i.e., trees of states of the model) pattern to better describe the system behavior. The granularity (e.g., seconds, minutes, days) and scaling with which the temporal constraints of the system are described, is also another point to be careful about. The nature of the system, in terms of determinism, non-determinism, and probabilistic, is also very crucial in picking the right modeling means and tools.

Considering all the mentioned issues, we advocate the teaching of logic and logical thinking in schools and more emphasis on temporal logic in M.Sc. of Engineering curricula, as it is of utmost importance to the future computer scientists and engineers. We argue that the fundamentals of formal methods need to start being thought at the initial years of university. This is in fact how universities around the world approach to building the logic and discrete mathematical background of students. However, more specific material such as formal modeling of systems, model checking, and formal verification are concepts to be thought during Masters, when students have already passed programming courses and developed an understanding of requirements specification (e.g., by using Alloy, UML diagrams).

3 Experience Report at Politecnico di Milano

In this section, we report our experience of teaching a FM course at our university. The course, as shown in Table 2, has been going on for more than ten years and follows a standard structure. The first part of this section explains our project-based teaching method of temporal logic, and the second part reports our observations from teaching Hoare's Logic.

Project-Based Teaching of Temporal Logic

A part of the evaluation is usually done by a modeling project. In other words, students are required to write a specification of a (complex) system by using temporal logic formulae. This task is demanding but essential to make students aware of the potential of logic and to show how logic goes beyond the abstract examples that are commonly adopted for teaching it. Using logic to specify and verify properties of realistic scenarios is therefore fundamental to motivate Computer Engineering students, as the exercise links abstract notions with tangible and realistic applications that they might encounter in their professional life.

We provide students with description of a safety-critical system. Using the last years's project[5] as an illustrative example, the students were supposed to formalize a scenario in which a human and a robot collaborate in an industrial setting; robot moves workpieces around in a the workcell to suitable places (position p1 and p2) for human to manipulate (at p1) and inspect (at p2) them. Students had to model the dynamics between human and robot and temporal back and forth between the two in a realistic and correct manner. Once they do that, they need to come up with a simple safety property such as "human and robot should never be closer that a certain distance, while robot is moving" and verify if the model they defined satisfy this property or not.

[5] polimi/fm2019.

The formula below is an example of what students wrote as safety property. It states that always[6] no same position for human and robot exists when the robot is moving.

$$\begin{aligned}&\mathsf{Alw}(\neg\exists x \in \mathit{Positions} : \mathit{RobotPosition} == x \wedge \\ &\mathit{HumanPosition} == x \wedge \mathit{RobotMoving})\end{aligned} \tag{1}$$

where *RobotPosition* and *HumanPosition* are variables with a limited domain expressing the position of the elements, and *RobotMoving* is a predicate indicating robot is changing its position.

$$\begin{aligned}&\mathit{RobotMoving} \Leftrightarrow \exists x, y \in \mathit{Positions} : \\ &(x \neq y \wedge \mathit{RobotPosition} == x \wedge \mathsf{Past}\,(\mathit{RobotPosition} == y, 1))\end{aligned} \tag{2}$$

Students were supposed to use TRIO [22], a metric temporal logic, to build their model, and Zot [59], a bounded satisfiability checker implemented and maintained at Politecnico di Milano, to verify their specified property. Following the "push button tools" indication earlier, the Zot tool is easily accessible through a Docker image and easy to use and readable instruction guides.

Our main observations during supervising the students for their projects follows below.

- It is only by doing the project that students perceive the concept of exhaustiveness of model checking which is its main difference with simulation. They learn how important it is to define constraints that make the model outputs realistic, but at the same time, do not limit the model to propagate only certain outputs (e.g., those that are most probably predictable by students while they imagine the model outputs) and avoid biased results.
- The concept of guaranteeing a property was also better conveyed to students by doing the project. For example, a common mistake by many students was to verify property P by model M with checking formula $M \wedge P$. They are happy when the tool satisfies the formula and pops out a trace of M in which P holds. This again shows the lack of proper understanding of model checking as a concept.
- Again on property verification, it is hard for students to grasp its motivation on a simple toy example; usually they ask "but why not to add the property directly in the model instead of verifying if the model satisfies it or not"? Working on the project allows students to clearly distinguish the model of the system from the specification that renders a specific requirement stated during requirement analysis, and that should be verified.
- Students usually better comprehend automata-based models, such as Timed Automata [5], rather than plain logic formulae, and prefer to practice on graphical modeling tools instead of writing formulae with a text editor. This is not a problem per se, as we could use off-the-shelf tools such as UPPAAL.

[6] A TRIO operator formalized as $\mathsf{Alw}\,(\phi) \Leftrightarrow \forall t(\mathsf{Dist}\,(\phi, t))$ which means ϕ occurs d time units in the future, where $\mathsf{Dist}\,(\phi, d)$ holds at time t if, and only if, ϕ holds at time $t + d$.

Lecture-Based Teaching of Hoare's Logic

Another part of the evaluation of the students is the application of Hoare's logic to a sample algorithm. We teach the use of Hoare's logic by applying it on a simple loop-less algorithm, given proper pre and postconditions. The students should learn how to come up with (i) a suitable invariant to prove the partial correctness of the algorithm, and (ii) a variant to prove the termination of the algorithm. After explaining the basics of Hoare, we move on to explain how we can unroll the loops and analyse all the iterations with deductive reasoning. We then see the same thing for conditional clauses. Students then are asked to do the same thing on a new algorithm.

For example, consider the bubble sort algorithm below. Its pre-condition is that array a had no repetition ($D(a)$), and its postcondition is that array a is sorted ($ORD(a)$), has no repetition and has all, and only the elements that it had before the execution of the algorithm. The latter is formalized by $P(a, b)$ that holds if there is an array b which had all and only the elements of a. The

Algorithm 1. The bubble sort algorithm used as an example to teach correctness proof.

```
1:  ${n \geq 0 \wedge P(a,b) \wedge D(a) = pre}$
2:  i := n-1;
3:  while $i > 0$ do
4:      $j := 0$;
5:      while $j < i$ do
6:          if $a[j] > a[j+1]$ then
7:              $temp := a[j+1]; a[j+1] := a[j]; a[j] := temp$;
8:          end if
9:          $j++$;
10:     end while
11:     $i--$;
12: end while
13: ${P(a,b) \wedge D(a) \wedge ORD(a) = post}$
```

steps of correctness proof is the following:

There are two while loops in the algorithm, thus students need to come up with two invariants, that are Inv_1 for the outer while and Inv_2 for the inner loop. Here we do not go trough the whole proof which is available in Mandrioli et al. [39] and discuss the parts student find more challenging which are defining Inv_1 and Inv_2, and analysing the effect of $temp := a[j+1]; a[j+1] := a[j]; a[j] := temp$; at step three which requires to be unrolled properly. We suggest students to ask themselves two questions in order to discover the proper invariant: (i) what keeps the loop going on? and (ii) what does each iteration do? The answers to these two questions for the outer loop are (i) while goes on for $0 < i < n$, (ii) it decreases i at each iteration so at final iteration $i = 0$ and rearranges the array at each iteration so that every element after position i is larger than every

Algorithm 2. The steps of correctness proof for algorithm 1.

```
1: $\{pre\}\ i := n-1; \{Inv_1\}$
2: $\{Inv_1 \wedge i > 0\}\ j := 0; \{Inv_2\}$
3: $\{Inv_2 \wedge j < i \wedge a[j] > a[j+1]\}\ temp := a[j+1]; a[j+1] := a[j]; a[j] := temp; j++; \{Inv2\}$
4: $\{Inv_2 \wedge j < i \wedge a[j] \le a[j+1]\}\ j++; \{Inv_2\}$
5: $\{Inv_2 \wedge j \ge i\}\ i--; \{Inv_1\}$
6: $\{Inv_1 \wedge i \le 0\} \Rightarrow \{post\}$
```

element before it. We also make students note that the precondition constraints still hold at line 3. Hence, Inv_1 would be:

$$Inv_1 = \begin{bmatrix} (0 < i < n \vee i == 0) \wedge \\ \forall z (i < z < n \Rightarrow \forall m (0 \le m \le i \Rightarrow a[m] < a[z])) \wedge \\ P(a,b) \wedge D(a) \wedge ORD(a,i) \end{bmatrix} \tag{3}$$

The first line cold be rewritten as $(0 \le i < n)$.

For the outer loop the answers are (i) the loops goes on for $0 \le j < i$ while Inv_1 holds, (ii) it increases j at each iteration so at final iteration $j = i$ and it places the largest element between position 0 and j at position j.

$$Inv_2 = \begin{bmatrix} Inv_1 \wedge (0 \le j \le i) \wedge \\ \forall z (0 \le z < j \Rightarrow a[j] > a[z]) \end{bmatrix} \tag{4}$$

After defining Inv_1 and Inv_2, we guide the students through the deduction they need to make step by step starting from precondition down to postconditions, as described in Algorithm 2. The most challenging step of Algorithm 2 for students is step three, where they have to analyse the manipulation of an array. We try to make it easy to understand as along the following lines. Starting from the constraint on line three of Algorithm 2 and having Inv_2 figured out, we need to replace j in Inv_2 with $j+1$ (assuming that backwards replacement has been already explained to students with easier examples). Then, we need to study how to apply backwards replacement with $a[j] := temp$. We ask the students to imagine a new and old version for the array a, which correspond to before and after execution of $a[j] := temp$. This would lead to the conclusion that a_{new} is the same as a_{old} except at position j where $a_{new}[j] = temp$. Next, we analyse $a[j+1] := a[j]$. Here again a_{new} is the same as a_{old}, except for positions j and $j+1$ where $a_{new}[j] = temp$ and $a_{new}[j+1] = a_{old}[j]$. Finally, $temp := a[j+1]$ ends to a_{new} be the same as a_{old} except for $a_{new}[j] = a_{old}[j+1]$ and $a_{new}[j+1] = a_{old}[j]$. This in other words mean that the analysed three commands swaps element j with $j+1$ and the rest of a has remained the same.

Our main observations on teaching Hoare follows:

- It is usually very difficult for students to guess an invariant. They usually move along the proof with a wrong invariant and surprisingly get to the end

(of course by mistake). It shows that student still have issues recognizing the edge cases in an algorithm.
- It is particularly difficult for students to deal with algorithms that work with arrays.
- Backwards replacement and deductions from constraints is not always easy for students, and even in best cases one could find errors in their proofs which goes back to their logic background, and imprecision.

4 Conclusions

In this paper we surveyed the current status of teaching formal methods at university and discovered a lack of effort in encouraging students to learn temporal modeling and specification. We analyzed several factors that might hamper an effective learning of logic and advocated the need for more effort in teaching logical thinking from the early stage of students careers. We argued that students lack a correct prospect on the practice of formal specification and verification which could be treated by project-based teaching. We then presented a report on our experience at Politecnico di Milano on teaching temporal and Hoare's logic. We used a project-based teaching method for the first one and a lecture-based for the other. In order to make a comparison between the two methods we asked student's feedback through a questionnaire. From 30 students attending the latest round of the course, of which only about 33% claimed to have some knowledge about FM, approximately 45% reported that the course made them more interested in the topic, only 11% consider to do their master thesis in this area, and 45% evaluated the project as the most interesting and useful part of the course. We received several comments from the students stating that the project helped them to practice temporal logic that otherwise would seem too theoretical and impractical. Additionally, we draw the following few observations:

- Teaching model checking is more productive by practicing it with a realistic system and off-the-shelf tools; Project-based teaching helps students practicing concepts such as exhaustiveness, property verification, and providing guarantee that a certain situation would never happen.
- The tool we provide the students with is well documented and has many available examples. However, not all off-the-shelf tools, which could potentially be very good alternatives to Zot, are well documented and easy for students. We have to consider that students have a limited time for the project and the effort for using the tool and modeling the scenario should be proportional to the credits of the course. Therefore, our options for the tool(s) we suggest to students are limited.
- Logic formulae scares students. It is helpful to use automata notions as the first step of approaching formalization of systems.
- In order to teach either of theorem proving or model-checking, students need a strong initial motivating introduction that demonstrates the practical use of these techniques.

– The questionnaire asked students to pick their favourite part of the course. As we said earlier 45% voted to the project, the other 33% picked the theoretical part of the course on temporal logic (which was necessary for the project as well), and 22% found the student presentations more interesting. That leaves 0% in favor of Hoare's logic! We think that the unpopularity of the Hoare's logic among students is due to the lack of case studies to justify its usefulness in the industry, and little possibility of assigning students with a feasible and meaningful project about it.

Acknowledgements. The credit of the statistics reported on our course goes to its official responsibles, previously prof. emeritus Dino Mandrioli and currently prof. Pierluigi San Pietro.

References

1. Abrial, J.R.: Teaching formal methods: an experience with event-B (invited speaker's extended abstract). In: Formal Methods in Computer Science Education, p. 1 (2008)
2. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Teaching concurrency: theory in practice. In: Proceedings of the Inernational Conference on TFM, pp. 158–175 (2009)
3. Ahrendt, W., Bubel, R., Hähnle, R.: Integrated and tool-supported teaching of testing, debugging, and verification. In: Proceedings of the International Conference on TFM, pp. 125–143 (2009)
4. Almeida, A.A., Rocha-Oliveira, A.C., Ramos, T.M.F., de Moura, F.L.C., Ayala-Rincón, M.: The computational relevance of formal logic through formal proofs. In: Dongol, B., Petre, L., Smith, G. (eds.) FMTea 2019. LNCS, vol. 11758, pp. 81–96. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32441-4_6
5. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994)
6. Artho, C., Taguchi, K., Tahara, Y., Honiden, S., Tanabe, Y.: Teaching software model checking. In: Workshop on Formal Methods in Computer Science Education, pp. 171–179 (2008)
7. Back, R.J., Mannila, L., Peltomaki, M., Sibelius, P.: Structured derivations: a logic based approach to teaching mathematics. In: FORMED 2008: Formal Methods in Computer Science Education (2008)
8. Back, R.J., Von Wright, J., et al.: Structured derivations: a method for doing high-school mathematics carefully. In: Turku Centre for Computer Science (1999)
9. Bohórquez, J., Rocha, C.: Assisted calculational proofs and proof checking based on partial orders. In: Formal Methods in Computer Science Education, p. 37 (2008)
10. Börger, E.: A practice-oriented course on the principles of computation, programming, and system design and analysis. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 65–84. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_5
11. Boute, R.: Teaching and practicing computer science at the university level. ACM SIGCSE Bull. **41**(2), 24–30 (2009)
12. Boyatt, R., Sinclair, J.: Experiences of teaching a lightweight formal method. In: Proceedings of Formal Methods in Computer Science Education (2008)

13. Carro, M., Mariño, J., Herranz, Á., Moreno-Navarro, J.J.: Teaching how to derive correct concurrent programs from state-based specifications and code patterns. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 85–106. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_6
14. Cataño, N.: Teaching formal methods: Lessons learnt from using event-B. In: Dongol, B., Petre, L., Smith, G. (eds.) FMTea 2019. LNCS, vol. 11758, pp. 212–227. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32441-4_14
15. Dadeau, F., Tissot, R.: Teaching model-based testing with Leirios test generator (2008)
16. Davies, J., Simpson, A., Martin, A.: Teaching formal methods in context. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 185–202. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_12
17. Duke, R., Miller, T., Strooper, P.: Integrating formal specification and software verification and validation. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 124–139. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_8
18. FBK-IRST, group at Carnegie Mellon University, T.M.C., the Mechanized Reasoning Group at University of Genova, at University of Trento, T.M.R.G.: NuSMV (2015). http://nusmv.fbk.eu/
19. Fernández-Iglesias, M.J., Llamas-Nistal, M.: An undergraduate course on protocol engineering – how to teach formal methods without scaring students. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 153–165. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_10
20. Ferreira, J.F., Mendes, A., Backhouse, R., Barbosa, L.S.: Which mathematics for the information society? In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 39–56. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_4
21. Filippidis, I.: A catalog of tools for verification and synthesis. github.com/fm-tools
22. Furia, C.A., Mandrioli, D., Morzenti, A., Rossi, M.: Modeling time in computing: a taxonomy and a comparative survey. ACM Comput. Surv. **42**(2), 6:1–6:59 (2010)
23. Garavel, H., Jorgensen, M.: A catalog of tools for the quantitative zoo. http://cadp.inria.fr/faq.html
24. Gibson, J.P., Lallet, E., Raffy, J.L.: How do i know if my design is correct. In: Formal Methods in Computer Science Education, pp. 61–70 (2008)
25. Gibson, P., Méry, D.: Teaching formal methods: lessons to learn. In: 2nd Irish Workshop on Formal Methods, vol. 2, pp. 1–13 (1998)
26. Guyomard, M.: Eb: A constructive approach for the teaching of data structures. In: Formal Methods in Computer Science Education, p. 25 (2008)
27. Habrias, H.: Teaching specifications, hands on. In: Formal Methods in Computer Science Education, pp. 5–15 (2008)
28. Habrias, H., Faucou, S.: Linking paradigms, semi-formal and formal notations. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 166–184. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_11
29. Hähnle, R., Bubel, R.: A Hoare-style calculus with explicit state updates. In: Formal Methods in Computer Science Education, pp. 49–60 (2008)
30. Hallerstede, S., Leuschel, M.: How to explain mistakes. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 105–124. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_8
31. Hartel, P.H., van Es, B., Tromp, D.: Basic proof skills of computer science students. In: Hartel, P.H., Plasmeijer, R. (eds.) FPLE 1995. LNCS, vol. 1022, pp. 269–283. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60675-0_50

32. Jard, C.: Teaching distributed algorithms using spin. In: Formal Methods in Computer Science Education, p. 101 (2008)
33. Kofroň, J., Parízek, P., Šerý, O.: On teaching formal methods: behavior models and code analysis. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 144–157. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_10
34. Kramer, J.: Abstraction and modelling: A complementary partnership. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 1–1. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_1
35. Lau, K.: A beginner's course on reasoning about imperative programs. In: Proceedings of CoLogNET/FME Symposium on TFM, pp. 1–16 (2004)
36. Lau, K.-K.: A beginner's course on reasoning about imperative programs. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 1–16. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_1
37. Anderson, L.W., Krathwohl, D.R., Bloom, B.S.: A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives (2001)
38. Mandrioli, D.: Advertising formal methods and organizing their teaching: *Yes, but*. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 214–224. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_14
39. Mandrioli, D., Ghezzi, C.: Theoretical Foundations of Computer Science. John Wiley & Sons, New York (1987)
40. Naumowicz, A.: Teaching how to write a proof. In: Formal Methods in Computer Science Education, p. 91 (2008)
41. Noble, J., Pearce, D.J., Groves, L.: Introducing alloy in a software modelling course. In: Formal Methods in Computer Science Education, p. 81 (2008)
42. Ölveczky, P.C.: Teaching formal methods based on rewriting logic and maude. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 20–38. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_3
43. Paige, R.F., Ostroff, J.S.: Specification-driven design with eiffel and agents for teaching lightweight formal methods. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 107–123. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_7
44. Poll, E.: Teaching program specification and verification using JML and ESC/Java2. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 92–104. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_7
45. Reed, J.N., Sinclair, J.E.: Motivating study of formal methods in the classroom. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 32–46. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_3
46. Robinson, K.: Embedding formal development in software engineering. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 203–213. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_13
47. Robinson, K.: Reflecting on the future: Objectives, strategies and experiences. In: Formal Methods in Computer Science Education, p. 15 (2008)
48. da Rosa, S.: Designing algorithms in high school mathematics. In: Dean, C.N., Boute, R.T. (eds.) TFM 2004. LNCS, vol. 3294, pp. 17–31. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30472-2_2
49. Schwartz, B.: The Paradox of Choice (2004)
50. Simonot, M., Homps, M., Bonnot, P.: Teaching abstraction in mathematics and computer science (2012)

51. Simonot, M., Homps, M., Bonnot, P.: Teaching abstraction in mathematics and computer science - A computer-supported approach with alloy. In: Proceedings of the 4th International Conference on Computer Supported Education, vol. 2, pp. 239–245 (2012)
52. Spichkova, M.: "Boring formal methods" or "Sherlock Holmes deduction methods"? In: Milazzo, P., Varró, D., Wimmer, M. (eds.) STAF 2016. LNCS, vol. 9946, pp. 242–252. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50230-4_18
53. Spichkova, M., Zamansky, A.: Teaching of formal methods for software engineering. In: ENASE, pp. 370–376 (2016)
54. Sznuk, T., Schubert, A.: Tool support for teaching Hoare logic. In: Giannakopoulou, D., Salaün, G. (eds.) SEFM 2014. LNCS, vol. 8702, pp. 332–346. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10431-7_27
55. Tarkan, S., Sazawal, V.: Chief chefs of Z to alloy: using a kitchen example to teach alloy with Z. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 72–91. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_6
56. University of Torino: GreatSPN: Graphical editor and analyzer for timed and stochastic petri nets (2001). http://www.di.unito.it/greatspn/index.html
57. Department of Information Technology at Uppsala University, Sweden, the Department of Computer Science at Aalborg University in Denmark: Uppaal (2008). http://www.uppaal.org/
58. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: Practice and experience. ACM Comput. Surv. **41**(4) (2009)
59. Zot: A bounded satisfiability checker (2012). github.com/fm-polimi/zot