Teaching Formal Methods to Software Engineers through Collaborative Learning (Short Paper)

Livia Lestingi(✉)

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan 20133, Italy
livia.lestingi@polimi.it

Abstract. It is common knowledge among researchers in the field that teaching formal methods can prove a challenging task. This paper reports on the approach adopted for a Master's Degree course at Politecnico di Milano, Italy, as an attempt to reverse this trend by introducing collaborative learning activities. Students put concepts learned during theoretical lectures into practice through a hands-on group assignment. Each group develops the formal model of a Cyber-Physical System through the Uppaal tool, starting from a set of requirements provided by the instructor. After delivering the assignment, we invite students to fill an evaluation survey whose results suggest a very high satisfaction level towards the hybrid theoretical-practical approach of the course.

Keywords: Formal methods teaching · Collaborative learning · Software engineering education · Postgraduate education

1 Introduction

Formal methods is not what students in Computer Science are most passionate about. Over the years, several experts in the field have tried to identify the root of the problem, and the effort required to grasp the **mathematical notation** is the most commonly mentioned issue. Especially for Software Engineering education, there seems to be an ever-growing gap between the practical approach of software development and the theoretical approach of formal methods research [16]. The negative perception of mathematics is so extensive that "*mathematical anxiety*" is now a customary expression. This perspective is contradictive considering that Engineering students deal with mathematics daily and that other branches of Computer Science, such as Machine Learning, are not inferior to Formal Methods in terms of mathematical complexity but widely more popular.

Over the years, several different teaching strategies have been proposed as possible solutions to this issue. Mandrioli [15] suggests adopting an *incremental* approach and increasing the level of *user-friendliness* (for example, by favoring *state-based* notations over formulae) without forsaking the rigor of mathematical

J. F. Ferreira et al. (Eds.): FMTea 2021, LNCS 13122, pp. 75–83, 2021.
https://doi.org/10.1007/978-3-030-91550-6_6

modeling. Liu et al. [14] suggest gradually introducing students to the most complex concepts, increasing the number of exercise sessions, and helping students understand the power and effectiveness of formal techniques through short and simple examples from daily life (also previously suggested by Gibson and Mery [7]). Others propose to increase tool support during teaching activities. The following tools are notable examples: the KeY-Hoare tool [9] for teaching Hoare logic; a toolset developed by Korečko et al. [11] for teaching formal aspects of software development based on Petri nets and B-Method; a tool by Spichkova et al. [18] for model-based testing that also accounts for possible human mistakes.

This paper reports on the approach adopted for the *Formal Methods for Concurrent and Real-Time Systems*[1] course for Computer Science Master's Degree students at Politecnico di Milano. The teaching approach features the innovative element of **Collaborative Learning** [12], which allows students to work together in small groups towards a practical goal exploiting the concepts learned during the theoretical lectures. At the end of the course, we have invited students to fill an evaluation questionnaire to assess their satisfaction level. The collected results paint a very positive picture: the vast majority of students who participated in the survey reported increased **confidence** in course topics and a genuine interest in the activities undertaken while working on the project.

The paper is structured as follows: Sect. 2 presents the context of the course, i.e., the Computer Science program and the Software Engineering curriculum; Sect. 3 introduces the innovation of Collaborative Learning; Sect. 4 illustrates the educational goals, requirements and theme of the group assignment; Sect. 5 presents the results of the evaluation survey; Sect. 6 concludes.

2 Course Context and Structure

The Computer Science and Engineering M.Sc. program at Politecnico di Milano allows students to build their curriculum flexibly. Courses are grouped into ten **tracks** that students can choose from to elect their specialization. Tracks cover a wide range of Computer Science branches and are updated yearly to keep up with the latest technological trends. Formal Methods for Concurrent and Real-Time Systems is part of the *Software Engineering for Complex Systems* track. The track's goal is to train future engineers to tackle issues related to the development and deployment of **complex software systems**.

The course is structured to teach students how to exploit formal methods throughout the **software development** process. The relevance of this practice has already been acknowledged over the years [10] and is now gaining more popularity as the demand for dependable software increases. A recent survey by Gleirscher and Marmsoler [8] highlights a non-negligible usage of Formal Methods in some areas, such as transportation and critical infrastructures. Promoting education on these techniques might help break the vicious cycle formed over the years [19]. Specifically, FM cannot spread in industry if employees (i.e., former students) do not possess sufficient knowledge on the topic, while students are

[1] Full information about the course can be found at: https://bit.ly/3gLXOdR.

not motivated to study it if (among the other reasons) expertise on FM is not required to work in industry. As a matter of fact, although more than 400 students enrol in the CS M.Sc. program every year, only about 40 of them chooses the FM course for their curriculum (specifically, 39 for A.Y. 2020/2021).

The selection of topics for the course (fully reported by Askarpour and Bersani [3]) and the adopted teaching approach are a tentative compromise between the two conflicting tendencies in FM teaching: deep focus on **theoretical background** and mathematical formalism versus the **learning by doing** approach. Concerns that Computer Science education is exceedingly distancing itself from abstract theoretical concepts started to emerge twenty years ago [20]. Over the years, this dichotomy has sparked a debate on whether this ultimately results in less-prepared computer scientists [15] or boosts their chances to solve real-life problems successfully [4]. In the following, we present the strategy adopted to tackle this challenging issue.

3 Introducing Collaborative Learning

The initial assumption is that avoiding the theoretical side of FM topics is not a viable option. Indeed, only teaching students how to use verification tools without proper knowledge of the underlying formalisms defeats the purpose of an academic formal methods course. On the other hand, the lack of *confidence* caused by the often elaborate mathematical notation requires attention and explaining theory through small examples only partially solves the problem [14].

The adopted strategy features two alternative ways for students to pass the course. They can either do an **oral examination** on course topics, which counts for 100% of the final grade or: 1) select and **present** a **FM tool**[2] in front of the classroom, counting for 60% of the final grade; 2) work on the **group assignment**, counting for 40% of the final grade. Both presentation and project have to be carried out in groups of 2–4 people. Students have about two months to work on the project before the final deadline. As of this year, 75% of the classroom has chosen the second alternative (tool presentation and group assignment).

The innovative measure is the introduction of **Collaborative Learning** (CL). CL is "*an educational approach to teaching and learning that involves groups of learners working together to solve a problem, complete a task, or create a product.*"[12] Students work on the same task which is entirely carried out using only one tool making it impossible to delegate fully independent subtasks to single group members; thus, students have to rely on one another to achieve the goal promoting **interdependence**. Students cannot complete the task autonomously, but they have to **interact** and challenge each other's ideas. The course targets students aged (on average) 23-24 who have completed at least three years of academic education; thus, they naturally display a good level of individual **accountability**. Group discussion and collaborative thinking help students develop skills such as conflict management and leadership. Finally,

[2] Eligible tools include: JBMC, CBMC, Prism, TLA+, COSMOS, SPIN, NuSMV.

groups are encouraged to monitor their progress with respect to the delivery deadline and periodically contact the instructor to receive feedback.

At the time of project assignment, Italy was not in full lockdown due to the COVID-19 pandemic. Although we do not possess accurate data due to privacy concerns, it is safe to estimate–based on how many people physically attended lectures[3]–that the vast majority of them were not residing permanently in Milan during project development. Nevertheless, the project outcome is entirely in software form and the university provides all students with the tools necessary to communicate remotely. Therefore, the collaborative learning strategy has not been affected by COVID-19 limitations at its core.

4 Group Assignment: Goals and Structure

This section reports on the group assignment's educational goals and how it is structured to meet these requirements. Afterward, we report the specific theme and model requirements for this year.

4.1 Educational Goals

The group assignment fits in with the Software Engineering profile of the students attending this course since its educational **goals** are:

G1: developing the **modeling** skill, i.e., how to translate informal requirements (expressed in natural language) into the formalism of choice;
G2: amplifying **critical thinking**, in terms of analytical experimental data evaluation to gain insights into the system performance;
G3: improving the capability of **expressing** oneself clearly and convincingly in written form and using accurate scientific language.

The centrality of managing models for Computer Science (goal **G1**) is widely acknowledged [5]. Besides requirement abstraction, students must perceive that a model can never perfectly match reality. Therefore, they must quickly learn how to balance complexity. System's behavior should be verified or simulated but purely reporting the results without analyzing them defeats the purpose of an engineering degree [6]. Students should proactively experiment with different system configurations and assess how this impacts the performance (goal **G2**). Finally, a study has revealed how non-technical skills, such as *self-expression* (goal **G3**), are highly requested by job applications, but they are also one of the most common gaps in a fresh engineering graduate's skillset [17].

Each **deliverable** required by the assignment fulfills one of the set-out goals:

D1: the developed **formal model** meeting the initial set of requirements;
D2: **verification results** and multiple ($\geq$2) **model configurations**;

[3] An internal survey shows that physical attendance rate at its peak was only 25%, whereas 75% of the students attended remotely.

D3: a **written report** describing model, experimental results, design choices, and the reasoning that led the team to choose one alternative over the others.

In the following, the specific project theme is presented with technical aspects such as the selected formalism and verification tool.

4.2 Project Content: Model-Checking for Warehouse Robotics

This year's (A.Y. 2020/2021) project theme is **warehouse robotics** management. Automated warehouses significantly spread over the last few years thanks to the introduction of mobile robots. These wheeled platforms can take charge of several tasks, most importantly *picking and delivering* operations. Items are stored in racks (i.e., *pods*) that robots can lift and transport to a *delivery point*. Human operators are usually in charge of manual edge tasks in this setting, such as picking the specific item from the pod. Students are required to develop a formal model (deliverable **D1**) of the following entities:[4] a) the warehouse **layout**: b) the **robots**; c) the **tasks**; d) the **human** operator. Specifications intentionally leave room for interpretation. For example, each team is free to choose whether the layout should be a standalone automaton or hard-coded into the model (i.e., as a two-dimensional array). The goal is to push each team to make **design choices** while drafting their model and provide reasonable justifications.

For the past ten years, the project focused on temporal logic [3], while, as of the last two editions of the course, the formalism of choice is **Timed Automata** (TA) [2]. The system under analysis dictates this choice since its behavior mainly hinges on timely synchronization among the different elements. Several features also naturally lend themselves to be expressed as *clock constraints*. For example, robots move every K time units, a new task spawns every T time units, and the operator takes time H to pick the item, where $\mathsf{K}, \mathsf{T}, \mathsf{H}$ are constant parameters.

Subsequently, students have to verify through **model-checking** a critical property (deliverable **D2**). The mandatory property is: "*it never happens that the number of tasks in queue exceeds the maximum queue size.*" This property subsumes that the chosen system configuration (e.g., number of robots, robot speed, tasks spawn rate, etc.) allows robots to complete tasks quickly enough to avoid *task overflow*. The property must be expressed in TCTL (Timed Computation Tree Logic). For illustrative purposes, a possible formulation is shown in Eq. 1, where parameter $\mathsf{MAX\_T}$ corresponds to the queue size and the number of tasks currently stored in the queue is captured by variable n_{tasks}.

$$\forall \, \Box \, (n_{\text{tasks}} \leq \mathsf{MAX\_T}) \tag{1}$$

Both the modeling and verification tasks of the assignment must be entirely carried out through the **Uppaal** tool [13]. As mentioned in Sect. 2, the course program includes a whole session dedicated to a hands-on demo of the tool.

[4] The full set of requirements is available at: https://bit.ly/3mEJbgc.

Starting this year, we have included the option to add **stochastic** features to the developed model that count as extra points in the final evaluation. These optional model features capture the **uncertainty** (refined by probability distributions) of the system's behavior. The introduction of probability distributions makes the automaton network no longer eligible for exhaustive model-checking but fit for **Statistical Model Checking** (SMC) [1]. To this end, students attend two additional lectures on the fundamentals of SMC and the Uppaal SMC extension. If they choose to pursue the stochastic path, they must verify through SMC the *probability* of property in Eq. 1 holding within a time-bound τ, whose formulation in PCTL logic is given in Eq. 2.

$$\mathcal{P}_{\leq\tau}(\square\ n_{\text{tasks}} \leq \mathsf{MAX\_T}) \tag{2}$$

Despite the extra effort, 55% of the teams have chosen to develop the stochastic features. Although they may have been motivated by the chance of getting a higher grade, this shows genuine interest on their side towards the project topics. The evaluation survey results presented in Sect. 5 confirm this intuition.

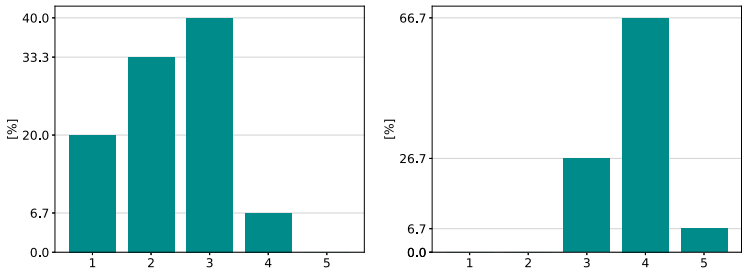
5 Evaluation Survey Results

We have invited students to fill an online evaluation survey[5] to assess their satisfaction level for the course and its effectiveness. Despite it being optional, 64% of all students who participated in the group assignment filled out the survey, whose results we comment in detail in the following. In some cases, results are compared with the ones previously presented by Askarpour and Bersani [3] to assess the evolution of the course with respect to its previous editions. About 75% of all students attended the course during their 1^{st} M.Sc. year (fourth year of academic education according to the Italian system). For 6 students out of 10, this was an **optional** course, which is a reassuring indication given the historical low attendance that affects this course.

Concerning the students' attitude and expectations towards learning formal methods before attending the course, only 20% of them state that they had prior experience with formal methods. Moreover, Fig. 1a shows the students' self-assessed level of **confidence** for these topics *before* attending this course which amounts to an average of 2.33/5. Although this may be a physiological consequence of a student's lack of knowledge in a specific area before receiving education, we can consider the increase of confidence shown in Fig. 1b as a valuable achievement. The reported confidence level after attending the course is, indeed, 3.8/5 and, most importantly, no student chose a value lower than 3, which hints at a homogeneous improvement for the whole classroom. The reported reasons of low confidence unsurprisingly mention mathematical notation and lack of prior expertise as the main sources.

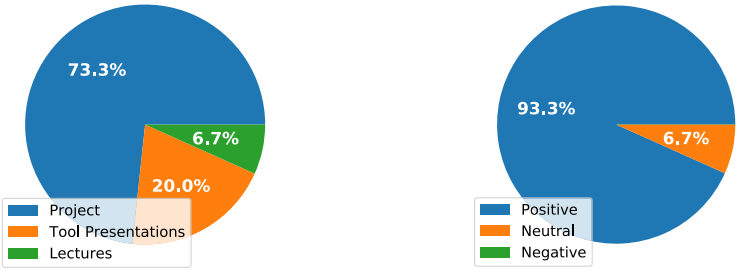
The questionnaire features questions specifically targeting the group assignment effectiveness. The results shown in Fig. 2 provide evidence that this is a

[5] Interested readers find the full set of questions at: https://bit.ly/3gAiHbS.



(a) Before attending the course (b) After attending the course.

Fig. 1. Students self-assessed level of confidence ($[1-5]$) on course topics.



(a) Favourite part of the course. (b) Impact on interest on course topics.

Fig. 2. Students replies targeting group assignment effectiveness.

successful strategy. Almost 75% of the students chose the project as the most appreciated part during the course. Moreover, the vast majority (93.3%) stated that working on the project **increased** their **interest** in course topics and, according to these results, no one lost interest because of the project. To complement the data in Fig. 2, 93.3% of the respondents also stated that the course stimulated their curiosity towards FM, and 4 people out of 10 said that they would consider a FM-related project for their Master Thesis (compared to the 1/10 ratio from two years ago [3]). Finally, the question about whether they would recommend the course to other students received an average score of 4.0/5.0, which is in line with the average of Politecnico courses (3.2/4.0).

Despite the favorable scenario, these results are not exempt from validity threats. Although most students enrolled in the course filled the survey, the original classroom size was meager, leading to only 25 replies. Furthermore, the survey carried out by Askarpour and Bersani in 2019 did not include specific questions about the project [3]. Therefore, we can only assess its effectiveness based on a single-year investigation. We will undoubtedly iterate the analysis for upcoming editions of the course to monitor future progress.

6 Conclusion

This paper reports on the strategy adopted at Politecnico di Milano for teaching Formal Methods to Computer Science students, specifically for the Software Engineering curriculum. The approach hinges on Collaborative Learning through the assignment of a group project. The answers given by students through an evaluation questionnaire provide evidence that the approach succeeds in building their confidence and stimulating their interest in course topics.

Acknowledgments. The credit for the course structure and syllabus goes to the official professors in charge, previously Prof. Dino Mandrioli and currently Prof. Pierluigi San Pietro.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. (TOMACS) **28**(1), 1–39 (2018)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Askarpour, M., Bersani, M.M.: Teaching formal methods: an experience report. In: Bruel, J.-M., Capozucca, A., Mazzara, M., Meyer, B., Naumchev, A., Sadovykh, A. (eds.) FISEE 2019. LNCS, vol. 12271, pp. 3–18. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57663-9_1
4. Bareiss, R., Griss, M.: A story-centered, learn-by-doing approach to software engineering education. In: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, pp. 221–225 (2008)
5. Desel, J.: Teaching system modeling, simulation and validation. In: Winter Simulation Conference Proceedings, vol. 2, pp. 1669–1675. IEEE (2000)
6. Ghezzi, C., Mandrioli, D.: The challenges of software engineering education. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, pp. 115–127. Springer, Heidelberg (2006). https://doi.org/10.1007/11949374_8
7. Gibson, P., Méry, D.: Teaching formal methods: lessons to learn. In: 2nd Irish Workshop on Formal Methods 2, pp. 1–13 (1998)
8. Gleirscher, M., Marmsoler, D.: Formal methods in dependable systems engineering: a survey of professionals from Europe and North America. Empir. Softw. Eng. **25**(6), 4473–4546 (2020). https://doi.org/10.1007/s10664-020-09836-5
9. Hähnle, R., Bubel, R.: A hoare-style calculus with explicit state updates. Formal Methods in Computer Science Education, pp. 49–60 (2008)
10. Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J.P., Margaria, T.: Software engineering and formal methods. Commun. ACM **51**(9), 54–59 (2008)
11. Korečko, Š, Sorád, J., Dudláková, Z., Sobota, B.: A toolset for support of teaching formal software development. In: Giannakopoulou, D., Salaün, G. (eds.) SEFM 2014. LNCS, vol. 8702, pp. 278–283. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10431-7_21
12. Laal, M., Laal, M.: Collaborative learning: what is it? Procedia-Soc. Behav. Sci. **31**, 491–495 (2012)
13. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell, vol. 1, pp. 134–152. Springer-Verlag (1997)

14. Liu, S., Takahashi, K., Hayashi, T., Nakayama, T.: Teaching formal methods in the context of software engineering. ACM SIGCSE Bull. **41**(2), 17–23 (2009)
15. Mandrioli, D.: On the heroism of really pursuing formal methods. In: FME Workshop on Formal Methods in Software Engineering, pp. 1–5. IEEE (2015)
16. Parnas, D.L.: Really rethinking formal methods. Computer **43**(1), 28–34 (2010)
17. Parts, V., Teichmann, M., Rüütmann, T.: Would engineers need non-technical skills or non-technical competences or both? (2013)
18. Spichkova, M., Liu, H., Laali, M., Schmidt, H.W.: Human factors in software reliability engineering. arXiv preprint arXiv:1503.03584 (2015)
19. Spichkova, M., Zamansky, A.: Teaching of formal methods for software engineering. In: ENASE, pp. 370–376 (2016)
20. Tucker, A.B., Kelemen, C.F., Bruce, K.B.: Our curriculum has become mathphobic! In: Technical Symposium on Computer Science Education, pp. 243–247 (2001)