Modernizing the Introduction to Software Engineering Course

Full Paper
SACLA 2019
© The authors/SACLA

Marko Schütz-Schmuck[1][0000−0002−0059−2726]

Department of Mathematical Sciences, University of Puerto Rico at Mayagüez, Puerto Rico
marko.schutz@upr.edu

Preprint

Abstract. We report on the modernization of an undergraduate, introductory course in software engineering that started in 2017-2018 semester 2 offered at the University of Puerto Rico, Mayagüez. We present the institutional setting, our underlying philosophy, and resources considered. We aimed at complementing informal descriptions in any phase with formal ones. We describe the revised course, discuss evaluations of the modernized course as held in two subsequent semesters, and outline options for future improvement.

Keywords: software engineering · formal methods · education

1 Introduction

"What should be taught in an introductory software engineering (I2SE) course and how should it be taught?" Or, even more generally, "What is software engineering?" and "What is engineering?". After two colleagues retired who had been teaching our I2SE course, we took the opportunity to revisit these questions and to review the I2SE course. No simple or final answer would be expected. Nonetheless, the questions lead to valuable insights and allow us to discover our tenets.

We accept that engineering combines scientific knowledge with creativity and imagination to design an artifact and to show of this design that the resulting artifact will have desired properties. To predict the properties engineering uses scientific results and mathematics. Each of the sub-disciplines of engineering uses the branches of mathematics most appropriate for the type of properties of interest in the sub-discipline. Many of the sub-disciplines use differential equations for modeling and simulation, also calculus and linear algebra are frequently used.

Branches of mathematics relevant in software engineering are logic and proofs, algebra, and discrete mathematics. Properties of software-intensive systems can then be expressed and justified using such mathematics.

We believe that descriptions, statements, etc. in an adequate formal language should **complement** (not replace) any informal descriptions of properties of a system-to-be or of its environment. Or, in the words of Mills: “Natural language is imprecise; formal language is inaccurate.” [19]. This affects all phases and/or iterations into which we might choose to subdivide the software engineering process. The use of suitable formal languages thus cuts across the topics of our course.

We also believe the little use of formal texts is an indication of the immaturity of software engineering. Considering its relative age, it is not surprising that this discipline is less mature than e.g. mechanical engineering. First uses of the term “software engineering” date back to the mid to late 1960s, so software engineering is now in its 50s. Mechanical engineering, in comparison, can be said to have started with Newton’s laws of motion - first published 1687 - making it about 330 years of age. This time has allowed the body of skills and knowledge in mechanical engineering to mature and stabilize while at the same time allowing society to form reasonable expectations towards the discipline. In contrast, the body of skills and knowledge in software engineering still seems much less stable and as society experiments with this young discipline it shows lenience towards the discipline’s failures. From the discipline’s successes and failures reasonable expectations have to form, and to the extent to which society increases its reliance on the outcomes of software engineering, the discipline will have to mature. To this end it will increasingly develop and incorporate scientific (in this case mathematical) foundations in similar ways as what can be seen in the more mature engineering disciplines.

We do not believe an introduction of formal texts to be a panacea. Conversely, we do believe that most phases of software engineering benefit from complementing informal with formal descriptions, justifications, etc. and that the mere attempt to express more formally an understanding of the domain, of requirements, … improves that understanding.

Coming back to our initial question “What should be taught in an introductory software engineering course and how should it be taught?”. As can be expected, the answer was already largely constrained. The existing programs at our university which require the I2SE course with the courses that build on it, the existing programs’ accreditation, the way the courses leading up to I2SE are taught, and the professional opinions of colleagues constrained and guided the exploration of this question. We detail the context in which we operate and the constraints arising from it in section 3. We considered related courses at other universities, a selection of textbooks, and reports on innovative ideas on introducing rigor and formal reasoning into computing curricula. Only one of the textbooks considered combined formal and informal language in the way we wanted [8,11,9]. More detail on related work is presented in section 2. None of the related courses at other institutions seemed directly usable as a model for our revised course: either for lack of consistently complementing the formal with the informal or for lack of openly accessible information about details of the course. However, from Bjørner’s textbook and his direct advice we revised

Preprint

the course and section 4 presents details on the way the course was offered in 2017-2018 semester 2 and in 2018-2019 semester 1, respectively. Students did not receive the revised course as positively as we had hoped. Section 5 gathers feedback from students' blogs and surveys of the two distinct editions held up to the time of this writing, but low student response rates to our surveys make conclusions rather difficult. For the upcoming edition we plan to make improvements in the delivery of the course. Our lessons learned and plans for the future of the course are in section 6.

2 Related Work

Whether the university computing curricula need a stronger emphasis on complementing informal descriptions with formal ones is an issue of ongoing debate. Lethbridge's survey report [18] has been used to argue for reducing the formal, more rigorous, forms of description as e.g. reported in [26]. Even when accepting the need for such stronger emphasis, there are different opinions as to the way in which such stronger emphasis should be implemented in the curricula. Proposals range from elective courses on "logic for everybody" as proposed in [24] to the gamification of formal specification as proposed in e.g. [21,3] to an integration of relevant topics into the entire program of study, affecting almost every course related to computing or mathematics. Dines Bjørner proposed at least as early as 1993 [7] (and possibly earlier) this emphasis of the complementary nature of informal and formal descriptions as a cross-cutting aspect of university computing curricula in software engineering. In [12] he and Jorge Cuéllar expand substantially on those ideas, which are later incorporated into Bjørner's three-volume textbook on software engineering [8,11,9]. Similar views are presented by the ITiCSE 2000 Working Group on Formal Methods Education [14]:

> Eventually, the working group aspires to see the concepts of formal methods integrated seamlessly into the computing curriculum so that it is not necessary to separate them in our discussions.

We considered the publicly available course catalog entries and syllabi of software engineering courses at numerous institutions including but not limited to MIT, Cornell, Stanford, and McMaster. MIT does not have an explicit software engineering course, the one that comes closest would be "Software Construction" [5], although it focuses most on implementation aspects. Cornell has a "Software Engineering" course [6] which includes very little related to formal descriptions, specifications, verification, etc. Stanford does not seem to offer an introductory software engineering course and it is not clear whether/how the relevant topics are distributed over other courses. McMaster offers many courses covering software engineering [1] including (but not limited to): "Software Design I - Introduction to Software Development", "Software Design II - Large System Design", "Software Design III - Concurrent System Design", "Software Development", "Software Requirements And Security Considerations", and "Software Testing". Only short descriptions of these courses are available on McMaster's

Preprint

web pages. From the short course descriptions it is impossible to see to what extent they are concerned with the complementary nature of informal and formal descriptions.

We further considered the main undergraduate software engineering textbooks including but not limited to Sommerville [23], Pfleeger [20], Schach [22], and Ghezzi et al. [13]. All of these mention formal languages (there often called "formal methods"). None of them uses a suitable formal language in the cross-cutting and complementary way we would like. On the contrary, we find the treatment of "formal methods" delegated to a single chapter, possibly even an online supplement.

To our knowledge Bjørner's three-volume textbook [8,11,9] is the only textbook where formal text consistently complements informal text. This was the clincher to select it. In his textbook Bjørner proposes the triptych paradigm, separating domain description, requirements prescription, and software design as the phases of software development in the large. This paradigm differs from e.g. that of van Lamsweerde [17] in its identification of domain description as a phase of its own whereas van Lamsweerde considers the elicitation and elaboration of domain properties and assumptions to be part of requirements capture. Bjørner's textbooks use the formal language RSL (the RAISE Specification Language). Quoting [8]

> RSL, which we primarily use in these volumes, features both property-oriented and model-oriented means of expression, has a somewhat sophisticated object-oriented means of compositionality, and borrows from CSP [288, 289, 448, 456] to offer a means of expressing concurrency. Extensions to RSL have also been proposed, for example with timing [535], and with Duration Calculus, that is, temporal logic ideas [274].

CSP (Communicating Sequential Processes) [15] allows describing patterns of communication and synchronization among concurrently running activities.

Bjorner uses RSL and CSP throughout the entire three-volume textbook series to formally complement topics as diverse as container harbors or state machines.

3 Initial Situation

For historical reasons our institution offers several programs in computing at the undergraduate level:

program	department	faculty/college
Computer Engineering (CE)	Electrical Engineering	Engineering
Computer Science (CS)	Mathematical Sciences	Arts and Sciences
Computer Science & Engineering (CS&E)	Computer Science & Engineering	Engineering
Software Engineering (SE)	Computer Science & Engineering	Engineering

Preprint

There are 4 courses focusing on Software Engineering: “Introduction to Software Engineering”, “Software Requirements”, “Software Design”, and “Software Reliability Testing”.

There are two offers of a course called “Introduction to Software Engineering”: one is by the Department of Mathematical Sciences, the other by the (recently founded) Department of Computer Science and Engineering. The two courses are considered equivalent with respect to students’ program requirements. They only differ in the faculty assigned to the course and, as a consequence, in the textbook used, the homework assignments and other such aspects as they vary from one faculty member to another.

Students in all of the above programs need to take I2SE, but only students in SE need to take all the remaining courses. For students in other programs they are electives. For the students in SE and in CE the I2SE course is a pre-requisite for taking their capstone project course.

The programs are ABET accredited except for the CS program, which is working towards accreditation. The I2SE ABET accredited syllabus allocates times for covering topics as in table 1

Table 1. time allocated to topics

topic	contact hours
Introduction to the course	1
The Software Lifecycle	3
Estimation: Cost, effort and agenda	3
Planning and tracking	3
Risk analysis and management	2
User Interface design	1
UML language	4
Requirements analysis and specification	5
Design principles and concepts, system design testing	6
Software testing	4
Exams, discussion sessions, and presentations	13
Total hours: (equivalent to contact period)	45

Preprint

When two colleagues who had been involved in teaching I2SE retired in short succession and a new colleague started teaching the course we considered this to be a good opportunity to take a fresh look at how I2SE is taught.

4 Revised Course

Diverse influences contributed to the shape of the revised course. We contacted Dines Bjørner about the use of his textbook in our course, he was immediately available and we are very grateful for his many helpful comments, suggestions, and his general support.

We initially selected material to cover in consultation with the author. The course contributes to ABET-accredited programs and we needed the revised course to follow the existing ABET-accredited syllabus. For some students in programs with a capstone-course requirement I2SE would be the only course explicitly exposing them to software engineering topics. In light of these requirements we re-balanced the time spent on covering some of the material, added material on time management, scheduling, planning and estimation, on risk management, and on user interfaces. These concerns together with the philosophy presented in section 1, the choice of textbook, and discussions with colleagues shaped the revised course. It was first held in the 2017-2018 spring semester in 3 separate small sections with 14, 17, and 19 students, respectively and then held a second time in the 2018-2019 fall semester in one larger section of 53 students. In both cases we used Moodle [2] as the course's learning management system.

The details changed a little between the 2 semesters due to the difference in section size and since we made some adjustments based on the experience from the first offering.

In our first offering we did not include time management. Also, Algebra, Mathematical Logic, and CSP Channels were discussed right after the chapter on The Triptych SE Paradigm instead of a little bit later in the course. We initially thought students would benefit from the relatively early exposure to these topics, but from personal conversation we concluded that placing them later in the course would benefit the students.

The topics of the course are now covered by the resources in table 2.

The course emphasizes application by giving the students homework for every week. We time the homework so that students have to independently study the material in order to complete the homework. The goal is to make learning more problem-based. Homework was due on each Friday before class.

For the duration of the semester, students work within a broad application domain from one of the 15 domains outlined in the textbook - assigned using the student's ID modulus 15. We took homework exercises from the textbook and most exercises referred to the student's domain. We chose the 15 domains from the textbook since they have already been substantially elaborated and they are each broad enough to allow students to find a distinct niche within the domain.

Preprint

4.1 2017-2018 semester 2

Since we had 3 small sections for the first offering and no assistance, we decided on the following way of homework assessment:

- Friday's sessions were for homework discussion
- students individually presented one exercise at a time
- the work was discussed with the other students
- students' turn followed the class list
- if a student forfeited her/his turn, it was the next student's turn to present the current exercise
- we managed an average of 5 presentations per Friday session of 50 min

Table 2. resources used to cover course topics

contact hours	chapter(s)	chapter title(s)
1	[9] Ch. 1	The Triptych SE Paradigm
2	[16,25]	Time Management, Planning, Scheduling, Tracking
2	[9] Ch. 2	Documents
2	[9] Ch. 5	Phenomena and Concepts
1	[8] Ch. 8	Algebras
1	[8] Ch. 9	Mathematical Logic
2	[8] Ch. 21	CSP Channels
3	[9] Ch. 8	Overview of Domain Engineering
	[9] Ch. 11	Domain Facets
	[9] Ch. 16	Domain Engineering Process Model
2	[11] Ch. 10	Modularisation (Objects)
2	[11] Ch. 11	Automata and Machines
1	[11] Ch. 12	Petri Nets
2	[11] Ch. 13	Message Sequence Charts
1	[11] Ch. 14	Statecharts
1	[9] Ch. 17	Overview of Requirements Engineering
2	[9] Ch. 19	Requirements Facets
	[9] Ch. 24	Requirements Engineering Process Model
1	[9] Ch. 25	Hardware/Software Codesign
	[9] Ch. 26	Software Architecture Design
	[9] Ch. 30	Computing Systems Design Process Model
	[9] Ch. 31	The Triptych Development Process Model
1	[9] Ch. 32	Finale
1	[17] Ch. 3.2	Risk Analysis
1		User Interface Design
29		

Preprint

– the average of a student's best 4 presentations formed her/his homework grade

4.2 2018-2019 semester 1

The second offering was also without assistance. The mode of homework presentation used in the previous semester was not practical for a single group of 53 students. We decided to use peer assessment:

– the homework grade was split in two grades: one for the homework a student submitted and another for homework a student assessed of other students
– for every one of the 15 homework assignments we prepared a catalog of (on average) 15 assessment aspects
– assessment aspects had weights from 1-10
– students assessed the aspects in the range of 1-100
– Friday's sessions - after students' submission - were used to discuss example solutions, to clarify remaining doubts on the assessment aspects, and to lead into the assessment in general

- for every homework submission the submitting student assessed 5 randomly assigned peer submissions
- every student's assessment had a weight of 1
- for every homework we randomly assigned 6 student submissions to be assessed by the professor using the same assessment aspects
- the professor's assessments had a weight of 6
- we used the grade calculation in the Moodle workshop activity [4] for the grade on the assessments

5 Reception

5.1 From Students' Blogs

One of the students' (graded) course work activities is to regularly write a reflective learning journal (blog). We encourage the students to write about any aspect of the course, its content, the presentation, their experiences, suggestions, etc. We can expect that the fact that this is graded and not anonymous will change the way the students write and also what they write. On the other hand, we get a very high response rate. Students stated that they:

- previously "had a concept of Software Engineering being pure programming"
- were surprised to see an immediate application "of my knowledge from the class" on an intern job
- disliked the peer assessment since "some students do their job in the assessment and give constructive criticism while other just gave random grades"
- found the amount of homework too much
- "still do not understand RSL or CSP and really hate it"
- "believe the weekly homeworks and assessments help A LOT for this course, not only because it forces you to study the material every week, but you get to see what your classmates did (anonymously, of course) which opens your mind to other possibilities in solving these problems"
- liked "The weekly homeworks [..] because they keep you involved in the class even when you are not attending lectures"
- are "still hating this book"
- "found it[the textbook] somewhat strange and needed time getting used to. In the end, the book was actually good, it provided a lot of examples so it made up for any shortfalls."
- "don't mind the long exams as they provide you with lots of opportunities to get a decent grade"
- considered "state charts and sequence charts are one of the most important parts of a system in the design process [..] because last year I was working with a company as a COOP student and [..] the majority of documentation [of the system] were state charts and sequence charts modeling the behavior of the system"
- found that "time log and analysis of the management of time was impressive"

Preprint

– had heard about "new methods and tools that are been used by the industry as agile methodologies, but none of them were discussed in the course."
– "really felt like [they were] learning something useful: Documents, Domain Facets, Petri Nets, Statecharts and Sequence Charts, Requirements Facets and Software Design (as well as the time management week) were all great weeks for this class."
– found "Learning about how to describe the domain and write a good requirements document really shaped the way that I think about tackling anything that requires the modelling of systems."
– thought "The project management sections were highlights and definitely deserve to have more attention."
– "Petri Nets are actually fun though."

5.2 Student Surveys

Preprint

1. 2017-2018 semester 1 We conducted a survey towards the semester end. Students anonymously rated the following aspects concerning the professor on a scale from 1 indicating "never" to 7 indicating "frequently":

   (a) Indicates where the class is going, (b) Explains material clearly, (c) Indicates important points to remember, (d) Shows genuine interest in students, (e) Effectively directs and stimulates discussion, (f) Provides helpful comments on homework, (g) Is tolerant of different opinions expressed in class, (h) Is available outside of class, (i) Explains thinking behind statements, (j) Effectively encourages students to ask questions and give answers, (k) Adjusts pace of class to the students' level of understanding, (l) Seems well-prepared, (m) Stimulates interest in material, (n) Treats students with respect, and (o) Is effective, overall, in helping me learn.

   Only 11 of the students from all 3 courses together responded. Figure 1 summarizes the results.

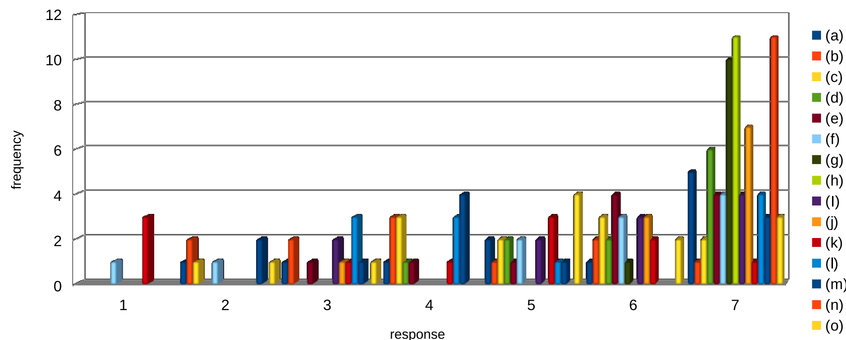

Fig. 1. Responses 1-7 to questions (a)-(o) about the professor 2017-2018-S2

Figure 2 summarizes results of students' assessment of the course on a scale of 1 indicating "no or very little" to 7 indicating "yes or very much" using the aspects:

(a) Would you likely recommend this course to a friend or fellow student?, (b) Did the content that was delivered and the organization of the course match what you were promised in the syllabus?, (c) How much new information and knowledge did you receive in the course?, and (d) How actionable do you think the information is that you received in the course?

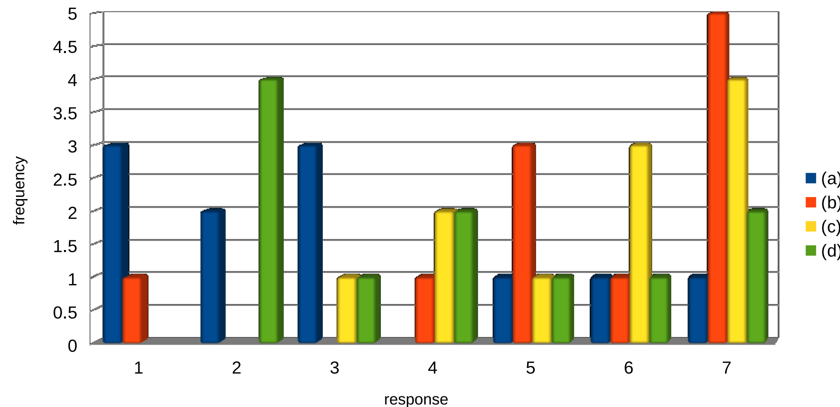

Fig. 2. Responses 1-7 to questions (a)-(d) about the course 2017-2018-S2

Prep

Finally, we asked them to provide free feedback on:

(a) What do you like best about this course?, (b) What would you like to change about this course?, (c) What do you think is this instructor's greatest strength?, and (d) What suggestions would you give to improve this instructor's teaching?

and to indicate "Approximately how many class meetings have you missed (including excused absence)?"

Respondents liked the weekly homework, the clarity of presentation, feeling they became better organized in problem solving in general, and an improvement in their presentation skills. The weekly homework with the student presentations was mentioned most.

Respondents would like to change the textbook, the grading of the homework (expressing that not presenting when they had done the homework felt like a waste of time), add more coverage of RSL.

Respondents suggested changing the homework evaluation (again expressing that not presenting when they had done the homework felt like a waste of time), changing the textbook, shortening the exams and switching to a different formal language than RSL. Changing the homework evaluation was mentioned most.

2. 2018-2019 semester 2 We conducted a similar survey after the second offering of the course. The response rate was lower than the first semester. This is likely due to the fact that a student created his own survey during the semester. While we do have the results of this survey also, the survey was structured very differently to ours which makes results hard to compare.
   Only 6 of the 53 students responded to our survey with results shown in figures 3 and 4.

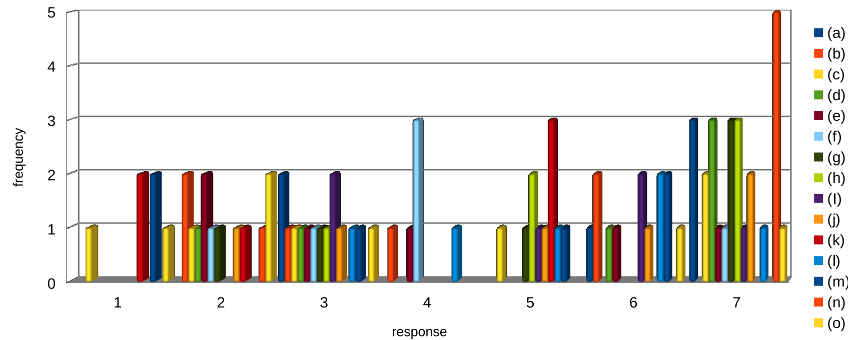

Fig. 3. Responses 1-7 to questions (a)-(o) about the professor 2018-2019-S1

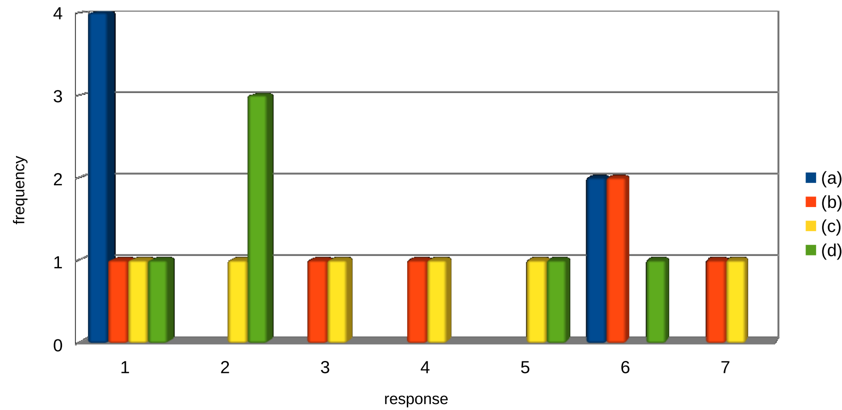

Fig. 4. Responses 1-7 to questions (a)-(d) about the course 2018-2019-S1

On the free feedback respondents liked the practicality of the topics, their exposure to project planning, that “it teaches you good problem solving

skills", and their experience in peer assessment. They disliked the slides from the textbook and would like to see the use of RSL removed or at least to spend more time introducing it. They would also like to change the domain they were assigned for the semester to be more like a project they pursue throughout the semester.

6 Lessons Learned & Future Work

The blogs (high participation, not anonymous) and the surveys (low participation, anonymous) by themselves give very incomplete pictures, even when combined we have to be careful not to follow a few opinions just because they are voiced loudly.

Respondents did not find the course's message to be very actionable. For the future, we hope to improve upon this by more strongly tying the homework exercises together into a single project. While the textbook already goes a long way on this, we feel that students perception of the exercises and the perceived actionability of their experience will improve with this change.

Respondents disliked the high workload of the homework. We hope that this perception changes at least in part when we integrate (most of) the current individual homework exercises into one semester-long project.

No respondent expressed a like for RSL, some expressed dislike, others did not mention it. We need to evaluate how to best improve this. Options include spending more time introducing RSL at the start of the course. This would clearly help students with reading and writing in that language. It would not address students' concern that there is too little additional material (tutorials, blogs, …) available and/or no visible use of RSL in industry. Another option would be to switch to a different formal language, e.g. TLA+/Pluscal. While this would give the students many more secondary sources of material to study, it would also require a thorough evaluation of the differences e.g. in the representation of internal vs. external nondeterminism, in the treatment of concurrency and synchronization, as well as a redevelopment in TLA+/Pluscal of a substantial portion of the examples used in the textbook.

Dislike of the slides accompanying the textbook. An obvious option would be to create slides in a style similar to the slides on time management, scheduling, tracking, and estimation, which the respondents preferred. We also consider transitioning more to a flipped classroom, i.e. successively replacing slide-based lectures with in-class demonstrations of solving example problems.

7 Conclusion

We have shared our experience revising the I2SE course to reflect - what we feel - is a more modern approach rooted in the philosophy that formal texts and informal texts complement one another. We explored existing teaching materials and courses offered elsewhere on whether they were compatible with our philosophy. Except for the textbook we chose, none of the others seemed compatible

Preprint

with these views. Our initial situation imposed diverse constraints on the course revision. We had to adjust some of our initial choices in order to satisfy these constraints. The course now differs from traditional I2SE courses in its use of suitable formalism to complement most of the topics of traditional I2SE courses. Reception is still far from what we aim for. In future offerings of the course will improve based on the feedback we received from our past students.

References

1. Course listing. `https://www.eng.mcmaster.ca/cas/programs/course-listing`, last accessed 2019/05/13
2. Moodle. `https://moodle.org/`, last accessed 2019/05/13
3. Verigames. `http://verigames.com/about-us.html`, last accessed 2019/05/13
4. Grade for assessment. `https://docs.moodle.org/36/en/Using_Workshop#Grade_for_assessment` (May 2017), last accessed 2019/03/05
5. Software construction. `http://web.mit.edu/6.031` (2019), last accessed 2019/05/13
6. Arms, W.Y.: Software Engineering. `http://www.cs.cornell.edu/courses/cs5150/2019sp/lectures.html`, last accessed 2019/05/13
7. Bjørner, D.: University Curricula in Software Technology. In: B.Z. Barta, S.H., Cox, K. (eds.) Software Engineering Education. pp. 5–16. [], Elsevier (1993). https://doi.org/10.1016/c2009-0-10293-3
8. Bjørner, D.: Abstraction and Modelling. In: Software Engineering [10]
9. Bjørner, D.: Domains, Requirements, and Software Design. In: Software Engineering [10]. https://doi.org/10.1007/3-540-33653-2
10. Bjørner, D.: Software Engineering. Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin (2006)
11. Bjørner, D.: Specification of Systems and Languages. In: Software Engineering [10]
12. Bjørner, D., Cuéllar, J.R.: Software Engineering Education: Rôles of Formal Specification and Design Calculi. Annals of Software Engineering **6**(1/4), 365–409 (1998). https://doi.org/10.1023/a:1018969717835
13. Carlo Ghezzi, Mehdi Jazayeri, D.M.: Fundamentals of Software Engineering. Pearson, Prentice Hall, 2nd edn. (2003)
14. Goelman, D., Hilburn, T.B., Smith, J.: Support for Teaching Formal Methods Report of the ITiCSE 2000 Working Group on Formal Methods Education (2000), `http://www.cs.utexas.edu/users/csed/FM/work/final-v5-7.pdf`, last accessed 2019/05/14
15. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
16. Humphrey, W.: Introduction to the Personal Software Process. Addison-Wesley, Reading, Mass (1997)
17. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley, Chichester, England Hoboken, NJ (2009)
18. Lethbridge, T.: What Knowledge is Important to a Software Professional? Computer **33**(5), 44–50 (2000). https://doi.org/10.1109/2.841783, last accessed 2019/05/14
19. Mills, B.: Practical Formal Software Engineering: Wanting the Software You Get. Cambridge University Press, Leiden (2009)

Preprint

20. Pfleeger, S., Atlee, J.: Software Engineering: Theory and Practice. Pearson Prentice Hall (2006)
21. Prasetya, I.S.W.B., Leek, C.Q.H.D., Melkonian, O., Tusscher, J.t., Bergen, J.v., Everink, J.M., Klis, T.v.d., Kostic, P., Meijerink, R., Oosenbrug, R., Oostveen, J.J., Pol, T.v.d., Vries, M.d., Zon, W.M.v.: Having Fun in Learning Formal Specifications. CoRR (2019), http://arxiv.org/abs/1903.00334v1
22. Schach, S.: Object-Oriented and Classical Software Engineering. McGraw-Hill, New York (2011)
23. Sommerville, I.: Software Engineering. International Computer Science Series, Pearson (2011)
24. Spichkova, M.: "Boring Formal Methods" or "Sherlock Holmes Deduction Methods"? CoRR (2016), http://arxiv.org/abs/1612.01682v1
25. Spolsky, J.: Evidence based scheduling. https://www.joelonsoftware.com/2007/10/26/evidence-based-scheduling/ (2007), last accessed 2019/05/03
26. Zamansky, A., Farchi, E.: Exploring the Role of Logic and Formal Methods in Information Systems Education, pp. 68–74. Software Engineering and Formal Methods, Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-49224-6_7

Preprint