Special Issue on Software Engineering Education & Training

Fostering Professionalism in Software Engineering: An Early-Exposure Approach

Valerio Terragni, *University of Auckland, Auckland, New Zealand*
Catherine Watson, *University of Auckland, Auckland, New Zealand*
Nicholas Rowe, *University of Auckland, Auckland, New Zealand*
Nasser Giacaman, *University of Auckland, Auckland, New Zealand*

Abstract—A professional software engineer needs excellent communication skills, a thorough understanding of software licensing and ethics, and the ability to follow standards in code style and version control. Albeit professionalism is a crucial aspect of Software Engineering, students are usually exposed to it later in their academic degree. We advocate for an early-exposure approach. We believe that it is important to start preparing students for the "culture" of software professionalism from early on. Our second-year Software Engineering course at the University of Auckland has been a successful application of such an early-exposure approach. In this course, we teach the fundamental aspects of software professionalism without diving into development methodologies, which are taught later in the degree. Positive student feedback confirms the value of this approach, which prepares students for long-term success and gives them a competitive edge in their careers.

Index Terms: *Software engineering education, Professionalism*

Software powers every critical aspect of our society, which makes Software Engineering a profession of crucial relevance. Educators have well-acknowledged that besides teaching students how to code, there is the equally-important learning objective of "professionalism"—something that the industry and society demands. In fact, less-technical skills such as communication and group work have recently become increasingly more important to industry [1]. Professionalism refers to the conduct, aims, or qualities that characterize or mark a profession or a professional person [2]. In the context of Software Engineering, it encompasses a set of values, behaviors, and practices that are essential for success in the industry. This includes qualities such as communication skills, ethical behavior, and a commitment to following industry standards and best practices. These activities go beyond just writing functional code. Adhering to professional standards and best practices helps prevent errors in the code, improves maintainability and readability, and promotes a culture of continuous improvement and learning. In addition, being professional can help build trust with clients and team members—leading to more successful and productive working relationships.

Software Engineering curricula often have project-based courses in which students work in teams to develop a software product from scratch—preferably involving real clients. These courses are crucial for preparing students for the industry [1], [3], and tend to be reserved for more advanced courses at the end of the degree (e.g., capstone courses). At this stage, students are better prepared to handle the technical and practical aspects of an industrial and real-world project. In such project-based courses, the "professionalism" aspects of Software Engineering cannot be ignored as they are fundamental for the successful completion of the project. Focusing only on the coding aspects will not prepare students for real industrial contexts and will likely lead to poor quality software projects, which might negatively impact the student experience.

Digital Object Identifier 10.1109/XXX.0000.0000000

THE BENEFITS OF AN EARLY-EXPOSURE APPROACH

We argue that it is potentially "too late" to face professionalism in Software Engineering in the later stages of a degree. First, project-based and industry-led courses have to cover a lot of content, such as software development processes (e.g., Scrum, DevOps) and Software Requirements Engineering. Instructors likely do not have room to adequately cover the professional aspects of Software Engineering, which will inevitably be neglected. Second, students can be overwhelmed by the new experience of working in a team and may not fully grasp the importance of professionalism in Software Engineering. Third, professionalism has to *grow*—and takes time to do so.

We advocate that Software Engineering students should face professionalism early on in their careers. We still believe that professionalism in Software Engineering is better taught with a practical (project-based) component. While moving "traditional" project-based courses partnered with industry earlier in the degree would be challenging (and potentially detrimental), we argue that it would be beneficial to have a second-year project-based course where students work in a team to implement a project from scratch. The course covers and enforces professionalism, but it is stripped from all the advanced Software Engineering concepts that only final-year students are typically able to embrace.

There are three important benefits of this *early-exposure approach*.

First, the development of professionalism in Software Engineering requires repeated exposure and practice over time, and may not be fully realized in just one capstone course. Having additional courses covering professionalism will be useful to better-face the final-year capstone projects.

Second, being consciously aware of professionalism early on will likely provide students with more opportunities to *embrace*, *implement*, and *practice* professionalism in later courses (and internships) that involve software development or team collaboration.

Third, our early-exposure approach is designed in a way that gives students the experience of working in a team and dealing with clients without exposing students to the theory behind effective software development processes, collaboration, and communication with clients and team members. This "experience" exposes them to the real challenge of an industrial context [6], making them more critical when the theory is exposed to them later. They will be able to better contextualize, understand, and appreciate what they are being taught.

EARLY-EXPOSURE IN PRACTICE

In realizing the early-exposure approach, we designed a second-year software engineering course as a stepping-stone to traditional final-year project courses. Prior to this course, students only experienced two traditional programming courses (introduction to programming, and object-oriented programming)—without any exposure to formal Software Engineering development processes and working in teams. Figure 1 summarizes the timeline of the course in terms of classroom activities, project work, and assessment. Like other standard courses at the university, the expected workload is 10-hours per week (four hours attending classes, plus six hours of work outside class time).

Like a real-world project, there is no exam or test. There is only one semester-long "project", broken down into three deliverables (Alpha, Beta, and Final) with incremental requirements. The Alpha deliverable is an individual assignment aimed to help students develop the confidence to develop in the relevant technologies before forming teams. For the Beta and Final releases, students are randomly allocated to groups of three students (the group formations are the same for both Beta and Final). The Final release carries the most weight, and serves as the final product delivery.

Classroom activities are divided into two categories: frontal classes with occasional group exercises, and design meetings where the teachers take on the role of clients and students ask questions to explore potential design solutions for the project.

At each deliverable, functionality is assessed by the team's brief demonstration to the clients. We automatically assess code style and Git usage using tools that we developed. The next subsections discuss each of the three aspects (Project, Classroom, and Assessment) in more details.

Project

Students are allocated into teams of three "professional software engineers" working for a software company, and are tasked to work together the entire semester. The course instructor plays the role of a "client" belonging to a hypothetical organization that has contracted this software company to build a custom application. For example, in 2022, the clients represented an art school wishing to incorporate digital technologies in their offering to attract remote enrolments in their offerings. The application to implement was a drawing game, similar to *Quick, Draw!*[1]. The team's role includes

[1] https://quickdraw.withgoogle.com/

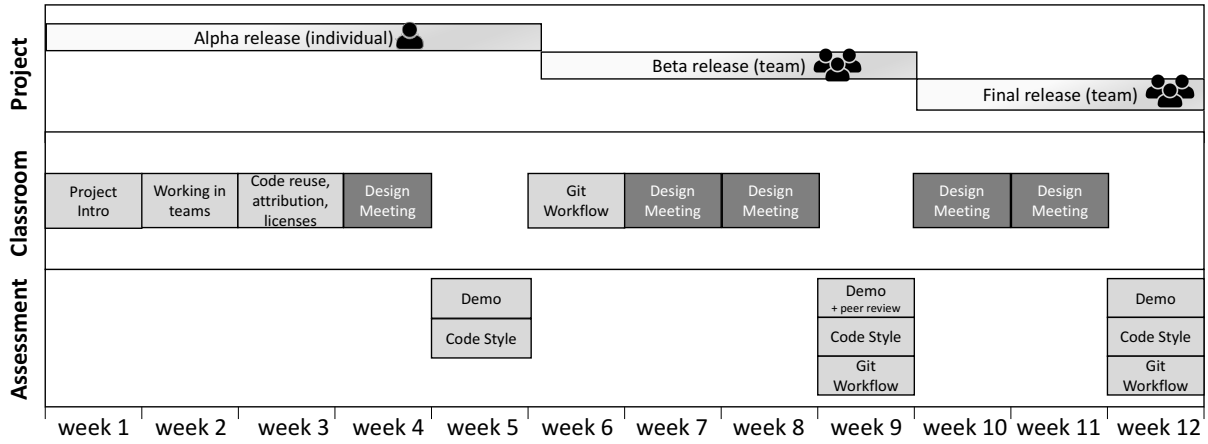

FIGURE 1: The 12-week course includes a semester-long project split into three deliverables (Alpha, Beta, Final). Design meetings transform the classroom into simulated client-engineer interactions, while assessments focus on professionalism in presenting to clients, as well as following software development protocols.

regularly facing the clients (in "design meetings" during the course-scheduled lecture times) to garner a better understanding of the requirements, while self-managing themselves in developing the application outside of class time.

To add to the realism, the client's organization is given a name and profile describing their services and business values. The client also has an alias name to differentiate their "client–engineer" interaction from that of the "teacher–student" relationship. The benefit of the instructor playing this dual-role (of both teacher and client), as opposed to having an independent client, is that the instructor is able to better-gauge project scope to minimize feature creep that might overwhelm students—and also prioritize the pedagogical experience of professional development rather than software functionality.

The client's backstory includes a rationale for needing to recruit the software company. This includes the problems the organization is currently facing, why they need the software application developed, and the organization's bigger-picture goals (e.g., the client wants to increase engagement with their customers). By developing an insightful understanding of their client's needs, students are encouraged to demonstrate their own initiative and creativity towards their proposed solution. This teaches students the importance of *adding value with their own ideas*, as the client does not necessarily always know what they want. In particular, the project description only gives some "core" requirements regarding the main functionalities, without giving much detail on how these functionalities should be implemented. During the design meetings, students will elicit the requirements from the clients.

Classroom

Frontal classes focus on the key aspects of professionalism. As Figure 1 illustrates, we start with covering how to ask questions to clients ("Working in teams"), followed by "Code reuse, attribution, and licenses", and finally "Git Worflow".

We teach students the ethics of reusing other people's code (including third-party libraries and code snippets from Q&A sites such as Stack Overflow), and to understand software licenses. This helps students become well-versed in best practices for code reuse and can apply them effectively in their future careers.

While we did not directly teach software processes, we teach Git best-practices and workflows. In particular, we covered one of the most popular and simplest branch-based Git workflow: "*GitHub Workflow*", which students are required to follow while implementing the Beta and Final deliverables. Such workflow provides students opportunities to review each other's code, which is an essential professional skill to learn.

Design meetings are the central component of the course in training students' attitude towards professionalism, and the expectation is they are treated as meetings with real clients (rather than with their instructors). This is the only opportunity engineers have to interact and seek clarifications from the client, as clients are not available outside of these scheduled meetings (but, of course, the instructors are still available as the "teacher"). If an engineer is unable to attend, but has questions for the client, they are required to delegate these to their team members to ask on their behalf. While this initially comes across as strict, students quickly learn to appreciate the limited availability their clients will have in the real world, and to therefore value

the opportunities they have to clarify requirements.

Establishing dialogic relationships between clients and engineers, and subsequently between members of the engineering team, presents students with a complex challenge. Assessing how students have responded to the complexity of this challenge is not straightforward: an ability to engage in a collaboration requires more than just an ability to contribute ideas and complete tasks [4]. Students therefore need to value not just their own contributions, but also their ability to *consider* alternate perspectives, *enable* the articulation of potential solutions from people with differing knowledge sets, *support* a decision-making process valuing pluralism and innovation, and *foster* the socialization of ideas and sense of shared ownership amongst the collaborating participants [5].

To guide the design meeting, students are taught the *types of questions* they might ask clients.

Contextual questions involve developing a sense of the client's history and values as an organization. This includes asking around what the client is doing besides the actual project. The rationale is that engineers seek to feed such insight into the project, or expand into other projects they might end up doing with the client. **Analytical questions** involve seeking rationales and trying to get into the client's head space. With this knowledge and insightful understanding, engineers will be able to demonstrate more initiative and add value with their own ideas. **Descriptive questions** ask about the product's design itself, and what it is. It is trying to get a rich and deeper description—not just a yes/no answer. The engineers are trying to develop a more vivid picture of what clients seek to achieve within this design. **Judgemental questions** seek yes/no answers in order to establish clear boundaries to overcome ambiguity. As a final note, **Searchable questions** are questions that students should (or could) know if they do their own research about the client, or what may appear very apparent based on everything already presented. Students are informed that these sorts of questions are discouraged as they will *undermine their credibility as professional engineers*.

Students are reminded that their role as engineers is to *co-construct* with their client, and use their own initiative in helping the client with suggestions and alternatives. When they ask questions, engineers should be proactive and foresee rationale, be ready to offer options, and be ready to present the pros and cons of the possibilities they propose. *The goal here is to encourage students in taking initiative, and developing the confidence to make their own decisions along the way without expecting the client to prescribe even the smallest of requirement details.* Developing this mindset also helps students recognize the importance of *establishing credibility as capable professionals*.

Small-group *role playing* during class time has provided an engaging and interactive mechanism for students to practice asking questions—as well as formulating justifications to answer ambiguities of their own. The instructor allocates students into groups of 5–6 people. We have seen that forcing students to work in teacher-defined groups encourages socialization, and students immediately meet someone new—thereby helping promote a sense of community in the class.

Students are asked to split the group into two sub-teams of roughly 2–3 students. Each sub-team is tasked to draft up at least 3–4 questions they would like to ask the clients, as well as identifying the type of the question (e.g., analytical, descriptive). One of the sub-teams is then assigned the role of *clients*, while the other sub-team is assigned the role of *software developers*. The software developer sub-team asks the client sub-team the questions they drafted up. The rationale is to encourage students to develop confidence in formulating and asking questions. The roles of the sub-teams are then flipped around, so that both sides have the opportunity to play the role of developers asking questions, and clients answering questions.

The sub-teams reconvene into their original full group to rank the top 2–3 questions (across both sub-teams) that they feel should be moved forward to the real client (played by the instructor). The instructor then reconvenes the whole class together, and asks one group at a time to ask one of their questions while the rest of the class pays careful attention to the questions asked by other groups—and the client's responses.

The instructor's goal (through the client role) is to *encourage students* to brainstorm and propose solutions, while foreseeing issues and potential alternatives. Most students are used to teachers providing explicit and detailed instructions with clearly-defined expectations—with little scope for them to exhibit their own creativity and initiative based on their personal interpretation of open-ended and vague requirements. The instructor needs to appreciate that many students are very "fragile" at this early stage of their studies, typically nervous and fearful of being judged due to lack of experience [7]. Instructors should remain mindful of the key learning outcome (which is to develop students' skills to engage with clients and proposing ideas, rather than the functional aspects of the project), as using such encouraging language will aid in developing students' self-confidence [8].

By starting in smaller sub-teams, then moving on to the full group, and finally in front of the wider class, this provides scaffolding in developing confidence

to communicate. When students ask their prioritized questions to the (instructor) client in the open classroom, their confidence is reinforced with the reassurance that their selected question is one that the group as a whole formulated and agreed is worth asking.

The nature of this activity serves purposes beyond practising question asking. In addition to the socialization opportunity, students are often able to "answer their own questions" when in discussion with peers (either in the sub-team, or group). This also helps brainstorming alternatives to present to the client, along with the question. Students learn the importance of prioritizing their questions, and valuing their client's time by making the most of the limited time they have with the client.

Assessment

Assessment in the course focuses on two main aspects: functionality and professionalism. But the term *professionalism* is multifaceted, and requires time to grow. At this early stage, we only need to focus on *some* aspects of professionalism to start the journey of accustoming students to its importance. We assess elements of professionalism in terms of (i) how students conduct themselves with clients, and (ii) how they develop software alongside others—including code style and Git workflow. The key *learning outcomes* related to this, as linked to the SE2014 [9] curriculum guidelines, include:

- *"Learn to communicate with clients in a non-technical manner"* (PRF.com: Communication Skills, and PRF.psy.4: Interacting with Stakeholders from SE2014),
- *"Convey and develop key software application features in a way that demonstrates deep understanding for the client's mission and ambiguous requirements"* (PRF.psy.5: Dealing with Uncertainty and Ambiguity from SE2014),
- *"Apply industry best practices while contributing to a team-based software project"* (QUA.cc: Software quality concepts and culture, and PRF.psy.1: Dynamics of Working in Teams and Groups from SE2014).

Demonstrations are the primary method used to assess students. For each of the three deliverables, students present their software product to the instructors (acting as clients). While the majority of the marks reflect whether they have satisfactorily implemented the "core" requirements, some of the marks relate to "professionalism", such as the quality of the demonstration in terms of engagement, formality, and politeness. Students receive the rubric in advance so they are aware that professionalism is a marked aspect.

In addition to feedback from the clients, each team is required to attend the Beta presentation of another team, where each student must provide independent feedback to the demonstrating team. Guidelines are provided on how to give constructive and actionable feedback, and students are graded on the quality of their feedback to peers. This feedback before the Final release provides an opportunity for teams to further improve their final product.

Code style and Git workflows are evaluated using tools that automatically analyze source code and Git logs to report any violations. To assess code style, we use our code style grading tool—GradeStyle [10]—which follows Google's widely adopted code style conventions and other popular code conventions. To assess the correct usage of the GitHub Workflow and Git etiquette, such as meaningful commit messages, we rely on a tool developed by our honours-year students.

It is important to note that in this course, we aim for complete shared code ownership. All marks for the Beta and Final deliverables are collective marks. This approach fosters a collaborative environment where students can share ideas, knowledge, and skills. Shared code ownership helps create a sense of ownership and responsibility among team members, which often leads to increased motivation. However, to recognize cases in which students did not contribute equally, we analyze Git logs, and we require students to complete two confidential reviews about their teammates at the end of Beta and Final submissions.

EVALUATION

To evaluate the impact of our approach, we present feedback gathered from students, to understand their perceptions of the value and usefulness of the course. An anonymous university-administered survey was released to students at the end of the semester, following a standard template for all courses across the university. Of the 132 enrolled students invited, 72 students (54.5%) completed the survey. The standard template is composed of ten 5-point Likert-scale questions, followed by three open-ended questions.

The final Likert-scale question on the survey is used as the university's key indicator of overall student satisfaction (*"Overall, I was satisfied with the quality of the course"*). Here, the course scored 4.72 (out of 5.0), while the engineering courses averaged 4.11, and the university courses averaged 4.15. All other metrics were scored similarly. The course scores ranged between 4.48–4.72, while the university-wide scores ranged between 3.89–4.32).

The students' responses to the open-ended ques-

tions helps illustrate the student experience. When asked *"what aspects of the course were most helpful for learning"*, students particularly valued the real-world simulation and having a full-semester project:

> *"I honestly loved the layout of this course. I thought it was a great idea to have one project spanning the whole semester as it gave us a real insight into what it would be like in industry. I really enjoyed the real life simulation of this course. It was a great way to learn how to interact and work with clients for our career."*

A common positive theme in the survey results was the opportunity to collaborate in a team environment, practice asking questions to clients, and the expectation of students to act professionally during classroom-based design meetings:

> *"Since we aren't used to working in groups for coding assignments, it was good that we had 'buffer' design meeting sessions that allowed us to create and ask questions with other people. I also like the structure of design meetings, the simulation of a professional environment is real enough and also engaging."*

When asked to reflect on what they found *"the most challenging about learning in this course"*, students commented on the challenge that comes with freedom to be creative—but they still appreciated the value of this freedom:

> *"Bit jarring to switch from an academic 'must do everything as the lecturer says' mindset to a developer 'how to implement this in my way' mindset, but it was a good challenge."*

> *"I found it challenging to implement some aspects of the app, since this course gives you lots of freedom and doesn't give direct instructions on how to do things. But that is the whole point of this course. It is challenging but rewarding."*

This feedback demonstrates students' acknowledgement of the challenges that come with being a professional software engineer. The freedom of creativity and committing to self-justified design decisions can be nerve wracking for many students, especially when grades are at stake. Students' realization that clients do not have all the answers prepares them for a career where they must exhibit initiative and credibility by co-constructing solutions with clients. Experiencing and practising dialogic relationships, with clients and teammates, alleviates concerns that there must be one correct solution. Creativity, and freedom to explore, are essential in order to solve complex problems [11].

Students will continue to practice these skills in subsequent courses during their degree, as the curriculum incorporates a lot of team-based project work. One of the much-demanded professionalism gap is that of developing students' awareness of business context [1], which is something not currently covered in our course. Incorporating this will enhance the comprehensive understanding of professionalism—complementing the course's current emphasis of communication, teamwork, and stakeholder interaction.

CONCLUSION

Fostering students to become professional software engineers presents a complex challenge for students. They are typically not faced with these challenges until later in their studies, where they have developed a much stronger technical understanding of software development—often in senior courses collaborating with industry.

We propose the early-exposure approach, as a way to foster this development earlier in their degree to encourage students to appreciate the challenges and expectations they will face as professional software engineers. We implement our approach as a semester-long project, with instructors simulating the role of clients. Classroom activities provide opportunities for students to explore design alternatives that demonstrate initiative and credibility as engineers.

Our experience has demonstrated second-year Software Engineering students' readiness to embrace the challenges of this pedagogical shift—despite only having learned fundamental programming concepts. We believe this shift will bring profound benefits to students' careers. In particular, scaffolding students' exposure to a simulated professional environment early on enables more opportunities for them to contemplate and practice the professional mindset over a larger portion of their studies, as they transition from students to professional software engineers.

REFERENCES

1. V. Garousi, G. Giray, E. Tuzun, C. Catal, and M. Felderer, 'Closing the Gap Between Software Engineering Education and Industrial Needs', IEEE Software, vol. 37, no. 2, pp. 68–77, 2020.
2. D. P. Hammer, B. A. Berger, R. S. Beardsley, M. R. Easton, and Others, 'Student Professionalism', Am J Pharm Educ, vol. 67, no. 3, p. 96, 2003.
3. C. Ghezzi and D. Mandrioli, 'The Challenges of Software Engineering Education', in Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, USA, 2005, pp. 637–638.
4. P. Griffin and E. Care, Assessment and Teaching of 21st Century Skills: Methods and Approach. Springer, 2014.
5. N. Rowe, R. Martin, R. Buck, and A. Mabingo, 'Teaching Collaborative Dexterity in Higher Education: Threshold Concepts for Educators', Higher Education Research & Development, vol. 40, no. 7, pp. 1515–1529, 2021.
6. O. Cico, L. Jaccheri, A. Nguyen-Duc, and H. Zhang, 'Exploring the Intersection Between Software Industry and Software Engineering Education - A Systematic Mapping of Software Engineering Trends', Journal of Systems and Software, vol. 172, p. 110736, 2021.
7. M. Norman and T. Hyland, 'The Role of Confidence in Lifelong Learning', Educational Studies, vol. 29, no. 2–3, pp. 261–272, 2003.
8. O. Akbari and J. Sahibzada, 'Students' Self-Confidence and its Impacts on their Learning Process', American International Journal of Social Science Research, vol. 5, no. 1, pp. 1–15, 2020.
9. M. Ardis, 'Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering'. IEEE Computer Society and ACM, 2015.
10. C. Iddon, N. Giacaman, and V. Terragni, 'GradeStyle: GitHub-Integrated and Automated Assessment of Java Code Style', in IEEE/ACM International Conference on Software Engineering (SEET track), 2023.
11. W. Groeneveld, J. Vennekens, and K. Aerts, 'Identifying Non-Technical Skill Gaps in Software Engineering Education: What Experts Expect But Students Don't Learn', ACM Trans. Comput. Educ., vol. 22, no. 1, Oct. 2021.

Valerio Terragni is a Lecturer in Software Engineering at the University of Auckland, Auckland, New Zealand. His current research interests include software testing, and program analysis. Valerio received the Ph.D. degree in Computer Science from The Hong Kong University of Science and Technology. Contact him at v.terragni@auckland.ac.nz.

Catherine Watson is an Associate Professor at the University of Auckland, Auckland, New Zealand. Her current research interests include speech production, speech synthesis, and acoustic phonetics. Catherine received the Ph.D. degree in Engineering from the University of Canterbury. Contact her at c.watson@auckland.ac.nz.

Nicholas Rowe is a Professor at the University of Auckland, Auckland, New Zealand. His current research interests include dance, collaboration, and education in diverse cultural contexts. Nicholas received the Ph.D. degree in Dance Studies from the University of Kent. Contact him at n.rowe@auckland.ac.nz.

Nasser Giacaman is a Senior Lecturer and Director of Software Engineering at the University of Auckland, Auckland, New Zealand. His current research interests include educational technologies, computing education, and immersive technologies. Nasser received the Ph.D. degree in Engineering from the University of Auckland. Contact him at n.giacaman@auckland.ac.nz.