Shaping Software Engineering Education with Lessons from the Past

University of Edinburgh Informatics Teaching Festival 2022

Talk
Presentation of published work

Overview
Shaping Software Engineering Education with Lessons from the Past

Who am I?
Timeline
Cooperative Programmes
Principles for Partnership Degrees
Joseph Maguire
School of Computing Science, University of Glasgow

• Elected Vice Chair of ACM United Kingdom Special Interest Group in Computer Science Education (UKSIGCSE) chapter.

• Programme Chair alongside Brett Becker for the United Kingdom and Ireland Computing Education Research (UKICER) 2022 conference.

• Deputy Convenor of Learning and Teaching for the School of Computing Science at the University of Glasgow.
Joseph Maguire
School of Computing Science, University of Glasgow


• In advance of developing the Graduate Apprenticeship in Software Engineering programmes we conducted a research programme to determine the optimal requirements and elements of such a programme.
Overview
Shaping Software Engineering Education with Lessons from the Past

Who am I?  
Timeline  
Cooperative Programmes  
Principles for Partnership Degrees
Timeline
1950s: Moon shoots and Magic Tricks

Timeline

• The classroom for most programmers was far more likely to be that of a military base or computing manufacturer office block, than a university campus.

• The reality is that those organizations developing and constructing computing systems required experts who could get the most out of the machinery. Consequently, computer manufacturers were committed to training programmers to get the most out of their systems, and so were significant training providers in the 1950s and 1960s [12].

• Limited transferable knowledge and skills as rooted in specific systems produced by manufacturers
1950s: Moon shoots and Magic Tricks

Timeline

• Not only did this result in a fragmented job market, there was also just not enough capacity even to meet the limited demand at the time.

• In 1954, for example, it was estimated by UNIVAC that only 260 programmers could be trained annually by all computer manufacturers [12].

• Another classic example is that of the Apollo Guidance and Navigation program. David Hoag, technical director, stated that “effort needed for the software turned out to be grossly underestimated” with over 350 programmers working on software at the peak of activity in 1968 [17].

• By the end of the decade, reaching into the 1960s there was growing recognition by government and industry to address the problem.
1960s: Forming the Foundation

Timeline

• In the 1960s, computing science undergraduate programs began to emerge with many institutions delivering the first such graduates.

• The number of institutions delivering computing science programs increased rapidly through the 1960s, but the residency of the subject varied between institution. The common ‘homes’ for computing science were mathematics, engineering or business [37].

• Richard Varga, Professor of Mathematics in 1964, commented that “the systems programming expert is neither a numerical analyst, a pure logician or at the other extreme a computer coder.” Varga argued that the activities of the computing professional were more closely aligned to that of engineers, rather than mathematicians [39].
1960s: Forming the Foundation

Timeline

• Nevertheless, Mathematics remained the favoured foundation for the discipline and the Association of Computing Machinery (ACM) solidified this with Curriculum 68 [2].

• The curriculum recommended eight mathematics courses among many traditional computing science electives. Not surprising as the committee was largely staffed with Mathematicians [3].

• In fairness, Curriculum 68 argued that gaining important skills such as programming would not be achieved alone with undergraduate programs, stating “it is also desirable that each student participate in a ‘true-to-life’ programming project. This might be arranged through summer employment, a cooperative work-study program, part-time employment in computer centers, special project courses or some other appropriate means.” [2].
1970s and 1980s: Reality or Theory

Timeline

- The spread of computing science continued with many more institutions forming dedicated departments and delivering undergraduate degree programs.

- Coates argued that computing science graduates were simply not productive when they entered the workforce and required additional training [10].

- Changes in funding, at least in the United States, focused on students having “the general power of choice in the education marketplace” [1], which may have resulted in students favoring education programs that afforded them skills and knowledge valuable to industry in order to secure fruitful employment.
1970s and 1980s: Reality or Theory

Timeline

- These courses often took the form of a single-semester project course where a team of students tackles a specific task [37].

- Parnas reports on one such course at Carnegie-Mellon University, focused on software engineering techniques, that was targeted at students both with prior relevant industry experience and with no experience [28].

- Despite challenges, single semester courses on software engineering began to spread through the 1970s and into the 1980s.
1970s and 1980s: Reality or Theory

Timeline

• For example, in 1984, University College London (UCL) introduced such courses into their programs to deliver graduates that were arguably more valuable to industry upon graduation [40].

• Similarly, the University of Southampton introduced a compulsory single semester software engineering course in 1986 to address the concerns of industry [15].

• However, while there was growing recognition of the importance of software engineering education, the solution was often the same from institution to institution, i.e., a single-semester project-based course. Plus, focus was always on blue-sky solutions, rather than existing code bases.
1990s: Experiments in SE Education

Timeline

• Ford suggested that while the practice of software engineering had been steadily evolving, the same was not true of software engineering education [14].

• In 1995, Bill Mitchell and Mitch Kapor devised and delivered a software design course to undergraduate students at the Massachusetts Institute of Technology (MIT), using studio teaching [19]. Studio education is core to the training of professional architects and Mitchell, Dean of Architecture and Planning at MIT, and Kapor, founder of Lotus Software, felt the approach could deliver valuable software engineering graduates.
1990s: Experiments in SE Education

Timeline

• Khun reflects on the success of the studio approach and the importance of environment and community in creating professional practitioners [20]. He argues that a studio space is typically a vibrant environment that stimulates learning, but also affords the development of reflective practitioners.

• Lee et al. state that student feedback was very positive towards their use of studio education for software design at Lancaster University in the United Kingdom [21]. Students were supportive of the approach with, for example, one student stating “I was significantly more productive in there than anywhere else I tried working” and another student, who participated only in a single studio session as part of a non-computer science program, claiming “I learnt more in one 2-hour Studio session than I did in this entire module.”
1990s: Experiments in SE Education

Timeline

• Nevertheless, the effectiveness of studio education in software engineering is often gauged with end-of-course questionnaires. Consequently, additional research will have to be undertaken to determine if the approach is effective, as well as enjoyable, for most students [7].

• The other concern with many of the experiences reported from instructors on studio education is that they largely focus on new systems or “green field” projects; again, the student is not necessarily engaged in existing, messy codebases.
2000s: Reconsidering SE Curricula
Timeline

• Despite ongoing experimentation in software engineering education and appreciation that software engineering was important to advanced economies, concerns continued that curricula for software engineering were not advancing.

• The need for software engineering education to evolve was further exacerbated as many more parts of the world demanded software engineers. China, for example, not only needed high-quality software engineers, but had to develop and deliver them rapidly to remain competitive [18].

• The importance of a university-educated workforce became increasingly clear at the turn of the millennium with many more individuals in emerging economies beginning to access education, while participation in higher education began to stagnate in many advanced economies [1].
2000s: Reconsidering SE Curricula

Timeline

- Consequently, the software engineering community became focused on organizing existing knowledge and reconsidering curricula and pedagogy [38].

- The community generated the Guide to the Software Engineering Body of Knowledge, and it represented what a relatively established software engineer should know.

- The Software Engineering 2004 curriculum guidelines were released outlining a set of outcomes, but more importantly they defined what every software engineering graduate should know. The motivation of both these efforts was really to focus on graduates from computer science departments as engineers, rather than scientists.
2000s: Reconsidering SE Curricula

Timeline

• Nevertheless, despite all these efforts, there was still much discussion around what sort of software engineering graduates computing science departments should be producing.

• Parnas envisioned three types of degree programs, one that produced computing scientists for research, another that produced software engineers for critical systems and a third professional degree program that delivered software engineers for non-critical systems [29].

• Shaw argued that curricula for software engineering were not adapting to the then emerging models of software design and development, such as open-source software development.
2000s: Reconsidering SE Curricula

Timeline

• The use of open-source software projects in-class is one such example. Buchta et al. argue that open-source projects can give students a far more realistic experience of software engineering as they are more likely to encounter large, complex and messy source code repositories [7].

• Pinto et al. conducted 21 semi-structured interviews with students who completed software engineering courses that involved open-source projects across five institutions [30]. Pinto et al. reported that students felt the approach not only strengthened their self-confidence, but also their technical skills with some students stating they received employment opportunities from participating in the open-source projects.
2000s: Reconsidering SE Curricula

Timeline

• Begel and Simon report on the experience of eight graduate software developers at Microsoft, revealing that much of their time is spent on communication, documentation, and working on bugs [5].

• Begel and Simon argue institutions are not preparing students for industry as they are still focusing on assessments that are largely artificial, e.g., small teams that are assigned tasks to create green field systems.
2010s: Focusing on SE skills

Timeline

• Radermacher et al. interviewed hiring and technical managers responsible for hiring software engineers across the United States and Europe [32]. The managers stated that they felt graduates generally lacked experience of working in large teams and had poor communication skills. The concern was that students were not afforded enough opportunities to develop effective communication and collaboration skills [31].

• Erdogmus and Peraire report on the use of active learning in a software engineering foundations course at Carnegie Mellon University [13]. The single-semester project course requires students to tackle a different topic each week, with each week resulting in a functioning piece of software. The students continue to iterate and refine their solution until submission at the end of course.
2010s: Focusing on SE skills

Timeline

• The pressure on universities to deliver graduates with valuable skills and knowledge has only increased since the millennium. Governments around the world, mindful of the cost of higher education, have become increasingly focused on understanding the effectiveness and value of education [1].

• In the context of software engineering, this puts more pressure on computer science departments to generate graduates that have relevant skills for roles in the software industry.

• Indeed, the Shadbolt review in the United Kingdom reported in 2016 that those graduates that had some industry experience, for example through an internship at a company, were far more employable [34].
Almost every decade represents a significant moment of progress for software engineering education. Reflecting on the history of software engineering education suggests that while considerable progress has been made, more needs to be done to create the optimal learning environment.

The challenge is that while academics have tried to introduce industry into the academic classroom, they have done so largely in an artificial manner. It could be that an alternative solution is to partner with industry, for students to put theory into practice.

An alternative approach could be to partner with industry to utilise the workplace as part of the classroom for Software Engineering.
Cooperative Programmes
Cooperative Programmes

- The limitations of traditional higher education classrooms to deliver graduates ready for industry has been recognised for almost a century.

- In 1899, upon returning to his alma mater Lehigh University in the United States, Herman Schneider was concerned for the next generation of engineers [36].

- Schneider was a successful architect and rapidly realised that lecture theatres and laboratories alone would not deliver the professional civil engineers required by industry.

- Schneider argued that the workplace must become part of the learning environment, a solution that would afford students not only the ability to put theory into practice, but also form social connections with established professionals.
History

Cooperative Programmes

• Schneider’s superiors at Lehigh University were less convinced at the time, not sharing his vision for the approach and sceptical of the perceived benefits [36].

• Nevertheless, Schneider, not dissuaded, departed and joined the University of Cincinnati, become the dean of the College of Engineering, and pushed forward with the idea of cooperative education.

• In 1905, Schneider won the approval of the trustees with agreement that he could explore a cooperative program with the proviso that “the failure of which we will not assume responsibility.” [36]
History

Cooperative Programmes

- Schneider recruited 15 companies to commit to the initial program where students were expected to follow each semester on-campus with another in the workplace, putting theory into action.

- Fortunately, the program was successful, evident given that 800 applications followed for the 60 approved spaces in the second year of the program.

- The success of the approach caught the attention of many universities and business leaders, resulting in the expansion and growth of the approach across the United States.
In the 1930s, after collaborating with Schneider and prominent inventor Charles Kettering, wealthy industrialist Walter P. Murphy made the substantial gift of $36 million to North-western University to fund cooperative education in engineering. The then university president, Walter Dill Scott, stated “any one of these three (Murphy, Kettering, Schneider) might have been the first to recognise that industry furnishes a training laboratory for engineers which no college can equal.” [24]

The growth and success of the cooperative programs continued but was disrupted by the Second World War. Institutions, such as Northwestern University which had flourishing cooperative programs by then, focused more on producing graduates rapidly, disposing of industry terms and replacing them with terms on-campus to produce graduates quickly for the war effort.
University of Waterloo
Cooperative Programmes

• The University of Waterloo is a public research university in Canada and part of the U15 Group of Canadian Research Universities. The institution was founded in 1957 with 74 students and has since grown to be the largest cooperative provider in the world with over 20,000 students enrolled in cooperative programs [26].

• Software Engineering graduates from the institution are highly sought by companies: Waterloo graduates are second only to those from the University of Berkeley for Silicon Valley giants such as Amazon, Apple, IBM, and Google [16,23,25].
University of Waterloo

Cooperative Programmes

• The institution generates such desirable graduates using a degree program lasting 4.5 years that combines workplace learning with theoretical learning on-campus.

• The program is composed of academic and workplace semesters. Each semester is 16-weeks long and students are required to complete three each year, with a total of six work-based and eight academic semesters.

• A typical sequence might have students initially spending the autumn (September to December) and winter (January to April) semesters on-campus focused on theoretical learning, before switching between workplace and academic semesters for the reminder of the program.
The interesting aspect of these workplace semesters is that each one is potentially with a different company, in a different environment. Students are not necessarily married to a specific career that is tied to their discipline, enabling a broad and liberal workplace education.

A software engineering student could work for a veterinary practice or accountancy firm. The approach and structure are not dissimilar to a traditional broad-based academic education where a software engineering student may study psychology or philosophy alongside mathematics and computing science.
University of Waterloo
Cooperative Programmes

• The University of Waterloo has constructed a bespoke facility, the William M. Tatham Centre, to support a streamlined recruitment process.

• Students receive an industry partner evaluation form in week 12, asking the student to rate the employer and their experience. The semester itself concludes typically around week 16 with both the student performance evaluation and industry partner evaluation expected to be completed and submitted to the institution.
University of Waterloo
Cooperative Programmes

• The outcome of each evaluation is crucial for both students and the industry partners. The accumulating student performance evaluations follow the student on their journey through the cooperative program.

• When the student enters the recruitment process for the next workplace semester, the prospective company can review student performance evaluations from previous workplace sessions.

• Consequently, a strong evaluation is sought from students and as such they treat each workplace semester seriously. Similarly, the feedback from the student about the industry partner when they complete their workplace semester feeds directly into the company’s profile on university systems.
Ostbayerische Technische Hochschule
Cooperative Programmes

- The concept of industry being part of a student’s learning environment has long been established in Germany via apprenticeship programs. In the context of higher education, the country operates Duales Studium or “dual studies” where students complete practical training in the workplace and academic education in an institution, typically a University of Applied Sciences (UAS) or Fachhochschule (FH).

- These institutions specialise in professional education and are more practice-oriented than traditional universities, concerned with the application of theory and professional skills. Consequently, UASs or FHs are distinct in that they typically do not award doctoral degrees and only appoint academic staff with several years of experience or service outside of academia.
Ostbayerische Technische Hochschule
Cooperative Programmes

• The institution offers dual study programs akin to co-operative programs in the University of Waterloo, effectively periods of time on-campus covering academic theory inter-spersed with time in the workplace. There are two models, mit vertiefter Praxis or “study with practice” and Verbundstudium or “company degree.” However, while programs from OTH and Waterloo share similarities, the OTH/German approach differs in that the student is employed by a single company for the duration of the program.
Both models are used for all disciplines; the structure is not specific to software engineering. Regardless of the model, in conversation with students they clearly favored the Duales Studium model.

An OTH student on the software engineering program stated “The decision to take the Duales Studium was, next to my wedding, the best in my life. I didn’t have to work during my semesters, thus being able to concentrate on my studies. The most important part for me was the combination of studying and practical experience.” Similarly, another student stated the workplace periods afford them the opportunity to “gain skills that they wouldn’t have otherwise, effectively and efficiently.”
Ostbayerische Technische Hochschule
Cooperative Programmes

• Students were concerned that while they had formed strong social connections within their company, scaffolding opportunities for such social connections in the academic periods could be improved. This emerged from the fact that academic courses comprised of students from multiple degree programs, not just those enrolled on the dual study program.

• Another interesting aspect is the institution and the programmes requires staff to be actively engaged with industry.
Trinity College Dublin
Cooperative Programmes

- The interesting aspect of the University of Waterloo and OTH programs is the focus on delivering university graduates that are primed and industry ready.

- Computing science teachers are an archetype example of such professionals. Schools cannot afford to lose them for significant periods of time, and yet, as professionals, they need to evaluate and improve their practice within their own context.

- Consequently, any effective program needs to incorporate its own classroom into part of the learning environment. Researchers and educators in Trinity College Dublin have devised such a program, the postgraduate certificate in 21st Century Teaching and Learning.
Summary
Cooperative Programmes

• The case studies represent a broad mix of approaches to cooperative education. An important aspect to compare across each of the different cooperative programs, however, is how each enables the student to put theory into practice and how practice in the workplace informs understanding of theory at university.

• The first case study, Waterloo, provides students the opportunity to experience multiple professional working cultures and affords students the ability to gain a multitude of transferable skills, but the various workplace environments are not necessarily relevant to the student’s degree program.

• The approach allows the student to make an informed decision about a career path and companies can have confidence they are appointing a graduate with considerable workplace experience.
Summary
Cooperative Programmes

• The second case study, Regensburg, provides students the opportunity to focus on a specific career and gain discipline skills and knowledge in a professional working environment for the duration of their study.

• The approach is powerful as it affords students the opportunity to gain a wealth of experience in industry relevant to their degree program, and, for some students, a professional as well as academic qualification.

• However, both these approaches, Waterloo and Regensburg, while providing excellent opportunities for students, do not necessarily integrate theory into the workplace and the workplace into attainment of theory. The workplace activities and university semesters are largely separated and disconnected. This is not true of the third case study, Trinity College, where students need to incorporate lessons in class with practice in their own working environment.
Principles for Partnership Degrees
Principles for Partnership Degrees

- Legitimate Peripheral Participation
- Rigorous and Relevant Curriculum
- Communities of Practice
- Academic and Industry Benefiting Society
Legitimate Peripheral Participation
Principles for Partnership Degrees

• The first principle is that the design of the cooperative degree program and its component courses must afford legitimate peripheral participation opportunities for students in their companies.

• This term, used in the apprenticeship literature, [e.g., 6], refers to the work that a relatively unskilled apprentice can undertake for their employer: it must be legitimate, in the sense that it is of real value to the company who, after all is paying the apprentice; it should be peripheral, such that if the apprentice makes a mistake (not unlikely as they are a learner) then the company won’t be put at risk; and it should be participatory, meaning that the apprentice should be working alongside others.
Rigorous and Relevant Curriculum
Principles for Partnership Degrees

• The student may become specialised to the company systems and software such that they are not able to transfer to a position elsewhere. In many ways, this would be resetting software engineering education to the situation in the 1950s where there was no solid foundation and programmers lacked sufficient theory and transferable skills.

• Consequently, universities must lean on established curricula as well as academic programs to ensure students are exposed to the core of discipline and relevant subjects.
Therefore, relevant Professional, Statutory and Regulatory Bodies (PSRBs) for software engineering should be consulted to ensure the program is relevant to the industry, not just a few companies.

Moreover, accreditation should be sought to ensure the qualification earned by the students is internationally recognised. This gave us our second principle for the design of the cooperative degree program in software engineering, that the curriculum must be both academically rigorous and industry relevant.
Communities of Practice
Principles for Partnership Degrees

• The experimentation in software engineering education in the 1990s revealed that some of the most engaging pedagogies, such as software studios [8], involved students forming a strong community of practice.

• Consequently, in the design of our program we need to ensure that assessments and course practices not only utilise the community of junior software developers, but also help shape and form the community.
Academic and Industry Benefiting Society

Principles for Partnership Degrees

• The level of integration between the university and the partner companies in which students work and learn determines the nature of the education that can be provided.

• In the final two years of the degree program, students will be both undertaking a blend of work-based projects and studying upper-level elective courses, perhaps specializing in areas such as data science, systems, or security. The electives, often research-led topics delivered by leading academics in the field, and the project work provide an excellent opportunity for low-cost, low-risk knowledge exchange from university into industry.
Principles for Partnership Degrees

- Legitimate Peripheral Participation
- Rigorous and Relevant Curriculum
- Communities of Practice
- Academic and Industry Benefiting Society
Summary
Summary

Shaping Software Engineering Education with Lessons from the Past

• Reflected on the history of software engineering education, it is rich, long and it is clear that challenges still persist.

• A significant challenge is to consider what is computing science and what is software engineering.

• It would seem no single lecture theatre or learning environment may be able prepare students as software engineerings, consequently partnering with industry may prove the most optimal for such professionals.

• Such partnership must make sense for both partners, i.e. in this case research-led HE and industry.
Questions
Questions

What is the difference between Software Engineering and Computing Science in terms of degree programmes?

What is the aim of the first year of a computing science programme versus a software engineering programme?

What does the learning and teaching as well as assessment practice look like for these aims?