Real-World Simulation: Software Development

John Blake [1]

Email jblake@u-aizu.ac.jp

[1] University of Aizu, Aizuwakamatsu, Japan

Abstract

The abstract is published online only. If you did not include a short abstract for the online version when you submitted the manuscript, the first paragraph or the first 10 lines of the chapter will be displayed here. If possible, please provide us with an informative abstract.

Students who triumph at school may flunk in the workplace. Undergraduates who excel at university do not necessarily excel at work, and vice versa. In a opinion survey of 400 employers in the US, the majority felt that recent college graduates were ill-prepared for the workplace. Researchers have also pointed out that university graduates have difficulty applying the skills learnt in higher education settings in real-world situations.

Introduction

Situation

Students who triumph at school may flunk in the workplace. Undergraduates who excel at university do not necessarily excel at work, and vice versa. In a opinion survey of 400 employers in the US, the majority felt that recent college graduates were ill-prepared for the workplace (Hart Research Associates, 2015, p. 11). Researchers have also pointed out that university graduates have difficulty applying the skills learnt in higher education settings in real-world situations (Brodie, Zhou, & Gibbons, 2015, p. 2). A recent study that found mismatches in students' and employer's perceptions concluded that higher education institutions need to work with students and guide them to take "individual responsibility to acquire and develop" transferable skills (Succi and Canovi, 2019, p. 1). However, the cause of this disjuncture is under-researched (National Research Council, 2013).

Transferable skills, also known as soft, generic, enterprise or portable skills, refer to skills that are developed in one context (e.g. university) and able to be used in another (e.g. work) (Fallows & Steven, 2000). Examples of transferable skills include teamwork, problem solving, time management and public speaking. Based on a big data term extraction project of 4.2 million job advertisements, the FYA (2016) compiled a list of eight transferable skills that are required in a broad range of jobs. These are problem solving, communications, financial literacy, critical thinking, creativity, teamwork, digital literacy and presentation skills. In contrast to transferable skills, there are technical (specific or hard) skills that are discipline specific. The concept of core transferable skills, however, is not without critics. Hyland and Johnson (1998) argue rather unconvincingly that context-independent skills are a chimera.

Skills developed in educational settings provide a basis for those students to build on when entering employment. The rapid development of new technologies and constant innovations in many professions, including software development and engineering, means that even after graduation, learning needs to continue (Uden & Dix, 2004). Learning is not limited by location but my mindset. Students need to develop life-long learning skills so that they can learn new skills independently, effectively and efficiently.

Problem

In contrast to vocational colleges, educators in higher education tend to focus on imparting knowledge, developing behaviours and skills that enable undergraduates to succeed academically. Many university assessment specifications focus on evaluating the knowledge, behaviours and skills in academic contexts. There appears to be little concern for producing workers who can function effectively in their target vocation immediately on graduation.

Given that some students are able to function well in the context of education but not in employment, causal factors appear to be at play. One of the probable causal factors is the lack of awareness of the expectations of the community of practice (Lave & Wenger, 1991). Students may be unaware of the symbols, stories, rituals, rules, artefacts, attitudes and beliefs that permeate the communities of practice in their target vocation. This lack of knowledge of the organizational culture and its associated

values makes it difficult for newcomers to fit in. For example, delivering clean working code in a timely manner is a primary goal of software developers. In the university setting missing one or two homework deadlines to hand in code is unlikely to result in any serious consequence. However, in the software development world missing a contractual deadline may result in a severe financial penalty or forfeit of contract.

Hollywood has long portrayed software developers and programmers as loners with a preference for interacting with computers rather than people. This rather negative image of loners, however, does not match the reality in leading technology companies, such as Google, Facebook and Amazon. In large corporations software engineers work on tiny snippets of huge blocks of code. This is at odds to academic contexts in which computer science students develop whole projects while those in, for example, business and management tend to work in teams. Hogan and Thomas (2005) claim the focus on individuals working alone on projects contributes to this loner stereotype. In short, software engineers need to be able to function in teams to secure posts at top-tier companies.

Undergraduate computer science students also tend to ignore Hofstadter's Law, which states that "it always takes longer than you expect, even when you take into account Hofstadter's Law". (Hofstadter, 1980, p. 52). The corollary of poor time management is that students may end up submitting software that fails to meet the specifications in terms of both quantity and quality. While this may still result in a passing grade at university, in the real-world failure to deliver fully-functional software would most likely lead to loss of a client. Students in academia are sheltered from the harsh reality of cause-effect. This is particularly so for many Japanese students who are not chasing a high grade point average to maintain their scholarship status or pursue a further degree.

Case Study

Vocational education provides educators with the opportunity to orientate students to the world of work. By raising the awareness of students or the realities of the workplace, expectations of students may more closely align with their future employers. Internships and work placement provide real-world experience. In this paper, a simulated workplace experience is created within a university course in which students adopt the role of software developers and the teacher as the project manager. The primary aim of this project is to reduce the disjuncture between study and work environments.

Context

This simulation was run at the University of Aizu, which is the only university dedicated to computer science in Japan. The university is located in northern Japan. Almost all students are Japanese (95%) with approximately 90% of the students being male. Forty percent of the university faculty are non-Japanese who deliver courses in English.

This simulation is run during an elective course delivered by the Centre for Language Research named "language and patterns" which is a euphemism for computational linguistics. Elective courses are open to third and fourth-year undergraduates in the school of computer science and engineering. This course is popular and is usually oversubscribed. The course is limited to fifty participants due to room capacity, and so the first 50 students with the highest grade point average are selected. This means that selected participants may be slightly more hard-working and/or academic than the student population as a whole.

The course focuses on using computer science to develop online tools that address language problems. This is achieved through a real-word simulation of software development, replicating a work-like environment within the classroom. The tuition, including written materials, is delivered primarily in English although most students communicate together in Japanese when working in teams.

Course Content

In the real-world the target users of a particular software project are central to the development process. Projects that are vision-led and have easily understandable tangible outcomes are more likely to motivate students. Projects are also selected that are on the cusp of the students' technical proficiency, which forces all students to strive. Software developers who produce fully-functional and user-friendly code will receive positive reviews, gain higher usage and in a commercial situation generate revenue. As this is a university project, the aim is not to generate revenue, but to satisfy the target users. For this reason, the target users are defined and the software developed is deployed online for the users to access. It is important that the project is not just a vehicle for assessment purposes, but has a real purpose satisfying the needs of real users.

The software developed in this project targets Japanese speakers who deliver scripted presentations in English, but are unable to read the scripts aloud in an appropriate manner. The specific target users are the undergraduates in the school of computer

science at the University of Aizu who have to deliver a fifteen-minute presentation in English on their final-year project as a graduation requirement.

Course Organization

The course is allocated fifteen 90-min sessions. Sessions are held twice a week and are conducted over eight weeks, one academic quarter. The fifteen sessions are split into three overlapping phases each consisting of between four and six sessions depending on the students. The elective course is designed so that learners work in teams from the outset. Teams progress at different speeds, and so this provides teams with some flexibility in terms of scheduling their job queue. The course begins with teacher-controlled whole-class activities undertaken in teams and transitions to student-directed group-based activities conducted in project teams both independently and in collaboration with other teams. During this simulation, the students formed their own self-managed teams comprising four people and elected a team leader.

The three phases of the course are knowledge acquisition, skill practice and final project. Table 1 shows a skills matrix which details the specific skills that are focused on in each phase of the course.

Table 1

Skills matrix

Skill		**Phase**		
Category	**Details**	**1: Knowledge acquisition**	**2: Skills practice**	**3: Final project**
Technical	Discriminate between random and pattern	✓	✓	✓
	Identify language patterns	✓	✓	✓
	Understand syntax of regular expressions	✓		
	Use regular expressions to match patterns		✓	✓
	Use version control system appropriately		✓	✓
Transferable	Problem solving	✓	✓	✓
	Communication: messages		✓	✓
	Communication: formal report			✓
	Communication: presentation			✓
	Critical thinking	✓	✓	✓
	Creativity		✓	✓
	Teamwork	✓	✓	✓
	Digital literacy	✓		✓

Phase 1: Knowledge Acquisition

In the first phase, students learn the technical knowledge necessary to complete the subsequent phases. In the first few sessions, a series of learning activities were provided for student teams to complete to gain the technical and linguistic knowledge necessary for the final project. The key computer science-related knowledge is the use of regular expressions, which have been described as “wildcard searches on steroids”. In this particular project, students needed to learn the syntax and semantics of regular expressions to create rule-based pattern-matching code. They also needed to learn how to create a function using JavaScript that is executed on matching expressions. As this project relates to pronunciation, students needed to learn the basic phonological systems of English including sounds, word stress, sentence stress, intonation and linking.

Phase 2: Skill Practice

To ensure students develop the knowledge, skills and behaviours expected, a series of tasks are set. Task-based learning (TBL) has been used to allow students to apply knowledge in specific contexts and critically reflect on their application (O’Halloran, 2001). TBL enables students to experience learning in the context of completing real-world tasks and helps align their

expectations with the reality in the workplace (Harden et al., 1996). Some of the tasks are presented as problems that teams need to solve to complete the task. Problem-based learning activities (PBL) (Savery, 2006) have been popular in medical education for decades, but have been less widely adopted in other disciplines. PBL enables learners to critically analyze and complex real-world problems by sourcing, evaluating and using appropriate resources. This enables learners to combine and consolidate their subject-specific knowledge with their other knowledge sets (Duch, Groh, & Allen, 2001). PBL activities were adopted akin to Brodie, Zhou, & Gibbons (2015).

AQ1

Each team was tasked with the remit to complete four tasks to practice their skills before undertaking the final project. The project manager created approximately fifty tasks, graded into five difficulty levels. Task specifications were housed on an online workflow system, Trello. Student teams bid to complete the first-level tasks, and on completion became eligible to bid for subsequent tasks. The first two tasks are straightforward with obvious answers, the next two tasks are more complex, preparing students for the final project. The earlier tasks were designed to enable students to learn how to create regular expressions required for the final task. Students were required to develop their software for each task on Github or CodePen. Slack was used for communications between and among groups.

Phase 3: Final Project

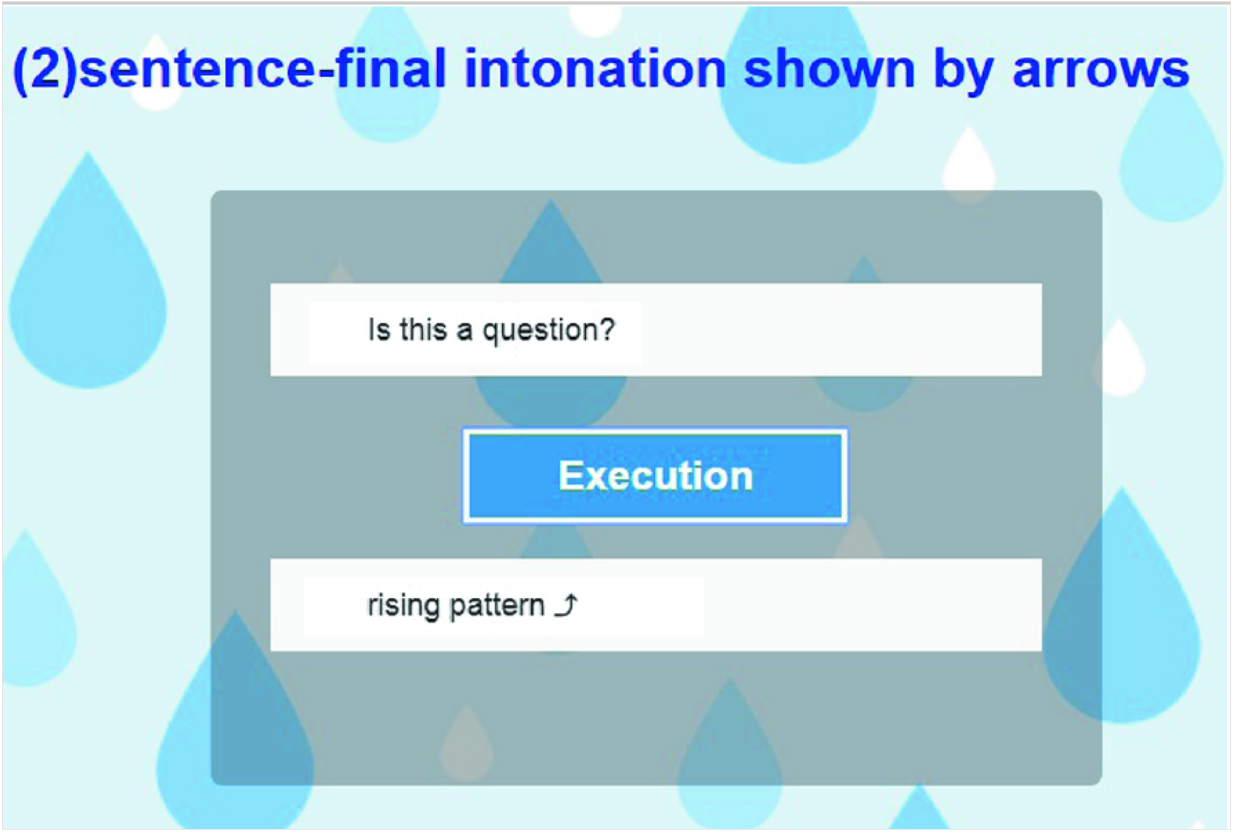
The final task was the development of a high-fidelity working prototype of a discrete language visualization function. The course approach could be described as project-based learning as the class project is to create a fully-functional prototype. This final project is the main part of the course and receives the lion's share of the assessment weighting. However, creating a high-fidelity prototype is a stretch goal for many students at the start of the course as they lack both the knowledge and skills. For this reason, the initial intensive knowledge-acquisition and task-based learning phases are included to provide students with a stronger foundation to build on. Student teams with excellent analytical skills and superior programming ability could probably complete a high-fidelity prototype for their discrete function in one working day. However, in the vast majority of cases, student teams take the whole duration of the course to complete their prototype. Screenshots of two prototypes are given in Figs. 1 and 2.

Fig. 1

Screenshot of function discriminating the pronunciation of "s"

Fig. 2

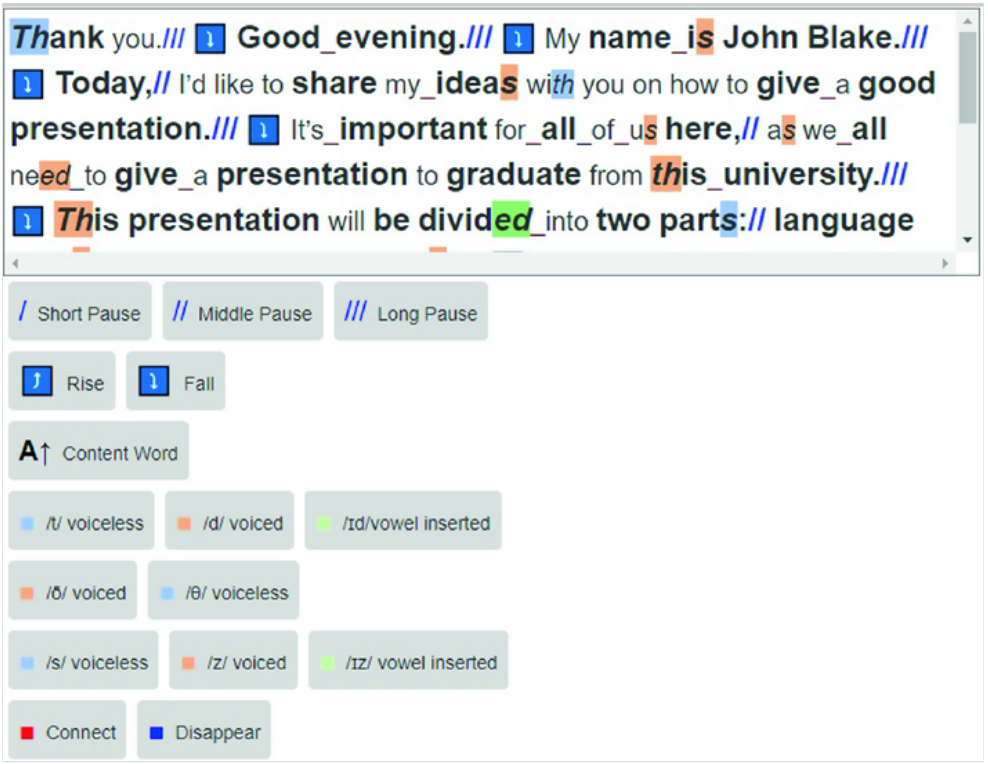
Screenshot of function identifying the type of intonation at the end of a sentence



This project provides the course with a clear direction and provides the class with a shared vision. After the course, these discrete functions were combined into a single program and a suitable graphical user interface constructed. Figure 3 shows the output of the first deployed Pronunciation Scaffolder (Blake, 2018).

Fig. 3

Screenshot of the output of the pronunciation scaffolder version 2.0
(Available at: https://john6938.github.io/PronunciationScaffolder/)



The course assessment was criterion based with full marks awarded for successful on-time completion for the first four tasks and zero marks awarded for partial or none completion, which reflects typical workplace environments. The final task was evaluated based on both successful completion and also on the ability to present the results through written reports and oral presentations.

Course Approach

Roles

Rather than the conventional roles of teacher and student, the teacher adopted the role of project manager and students acted as freelance developers. Students formed teams who bid for and develop working prototypes of language visualization software. Group-based project work has been shown to be more effective at developing transferable skills than traditional university-style lectures (Duffy & Bowe, 2010). Working in teams provides team members with numerous opportunities to develop their communication and time management skills. Team members work together to create their discrete function. When requested by another team, teams are required to provide assistance. The most common requests are to act as users to evaluate the usability of the software, but some of the less proficient teams request assistance with programming.

Throughout the course learning is designed to occur during team discussions. The activities, tasks, problems and projects all act as vehicles through which students gain knowledge, practice skills and develop appropriate behaviours. The teacher adopts the role of project manager and facilitates discussion rather than the role of knowledge-giver and problem-solver. When a particular group is unable to progress, a team member of a different group is seconded to their group to provide assistance. This again replicates the workplace when consultants are employed to solve problems.

Rewards

A key concept was to create an environment in which students were rewarded in the same way as freelance software developers. Rewards are tied to measurable events, such as the completion of a project. To provide students with the motivation to take responsibility for their work and deliver in a timely manner, a bidding system was used for tasks. Developers who produce clean code on time secure the best work and gain first choice on future bidding contracts for tasks. This creates a virtuous circle for those who are able to produce the deliverables. As with freelancers, small-scale contracts (tasks) were first awarded and on completion of four tasks, the final task was awarded.

Technologies

To familiarize students with the types of technologies that they are likely to use in the workplace, students were required to use Github or CodePen to develop prototypes. GitHub is a cloud-based version control system that is extremely powerful, but has a steep learning curve, which many of the students had not yet climbed. Feliciano, Storey, & Zagalsky (2016) describe how GitHub can be used in courses for software engineers. Marquardson and Schuetzler (2019) reflect positively on their use of incorporating the use of GitHub in a collaborative programming task. In the assessment criteria, appropriate use of GitHub is emphasized. Students were not limited to a particular programming language, but had to ensure that the input and output of their function was compatible. Extensive use was made of Slack and Trello, which are commonly used cloud-based workflow and communication platforms for software developers.

Deliverables

Software developers in the real-world produce two key deliverables: software and technical reports. In addition, they have to present their work to both technical and non-technical audiences. To replicate this, each team was required to submit their software (usually housed on GitHub or CodePen), and a technical report explaining the software development process and critiquing their software. This report tended to focus on the false negative and false positive results. In addition, they produced a five-minute video for target users describing and explaining how their software works. A notable advantage of video presentations is that giving a presentation to a web camera or a fellow team member is less stressful than giving a live presentation in front of the teacher. Clear criteria was provided so that learners were able to estimate their expected grade, and then judge whether to submit work that would meet expectations and receive a passing grade, or aim for a higher grade. Students had to submit a video presentation as a course requirement, but were given the option of giving a live presentation as well. They could then elect to use either the grade from the live presentation or the video presentation. However, they had to make the selection between the live or recorded presentation grade before either of the grades were released. Only two teams opted for live presentations and no teams opted to use the live presentation grade. In the subsequent courses, live presentations were not given as an option.

Digital Artefacts

Digital artefacts are digital documents produced in an educational context (Barton and Collins, 1997). The submitted software, reports and videos are, therefore, digital artefacts. With permission from the creators, these digital artefacts can easily be reused and adapted to help future cohorts. For example, students may produce software and create a video explaining the shortcomings in their software. A future cohort of students could be set the challenge to fix the bugs by using information in the video explanation. The inspiration for this stems from Chewar and Matthew's (2016) implementation of using video with software engineers. As improvements can always be made, this cycle is never ending. Digital artefacts can also be assessed more easily. For example, a video presentation can be replayed whereas in a live presentation, the assessor needs to make on-the-spot decisions. Humans are inherently subjective, swayed by multiple cognitive biases and may at different times on different days evaluate exactly the same presentation differently. Digital artefacts do not change this subjectivity, but do provide a way to ameliorate the subjectivity using multiple raters, moderators and using the videos for rater standardization.

Course Feedback

The official end-of-course summative feedback questionnaire was very positive. However, in this Japanese university context, formal feedback tends to be positive regardless of the course or instructor. At the end of this course, many students contacted me directly to talk about their projects. This is the first time that students in this university had wanted to continue to discuss course content after a course has finished. Their enthusiasm to deliver the best working prototype for the users was not constrained by the submission of the final assessment nor the official end of the course. This is a clear indication of taking responsibility for a project, which is a behaviour that employers value.

Confidence

One student came to my office with some suggestions to improve the code his team had created after the course. During this meeting he said that, "because of [this] course, I can talk to my supervisor". His final-year project supervisor was a Russian professor who does not speak Japanese and so their *lingua franca* is English. The student was painfully shy at the beginning of the course, but he managed to express his ideas directly to me (and hopefully his supervisor) by the end of the course.

Cultural Space: Technologies—Git, Slack and Trello

On a different day a four-person team of students came to discuss their code. During his meeting they explained that although they knew the importance of version control systems, such as GitHub, they had never had to use it for a real project. They had previously been forced to use some features in a course, but never used it after that. They were surprised at how easy GitHub

was to use once they had remembered the key commands. During the course, many students enjoyed using Slack. Perhaps, this was because they usually used similar sites to communicate with their friends. Slack, however, is much more powerful, and so once they discovered that, they set up their own Slack groups with their friendship groups. In the same vein, students found Trello was practical and easy to use. Students realized that Trello could be used to manage their university assessments and help them sequence the tasks for their final-year project.

Technical Skill: Regular Expressions

A number of students continued to improve their prototypes after the end of the course. The vast majority of improvements involved regular expressions. Many of the less proficient programmers knew the term regular expression but were unfamiliar with their syntax and their power. Regular expressions are used within many programming languages. However, each language has some systematic changes to the semantics and syntax. The skill practice phase gave students the chance to practice using them in a non-threatening environment. Regular expressions look incredibly difficult and so many students simply never tried to learn them until this course. Many students stated that they were pleased to get reintroduced to regular expressions and expected it to have a positive effect on their general programming.

Vision

Strategic management introduced mission statements and visions as a way to focus the workforce on achieving a particular aim. This shared vision is designed to create a context of shared goals, and increase employee buy-in and willing participation. The vision itself is probably key. Some visions may help focus employees (or in this case students) while others may not. In this instance, the vision of helping graduating students read out their presentation scripts aloud more easily would not only help others but could also potentially help themselves as well.

Through this process, student feedback shows that they better understood the need to take responsibility, produce deliverables on time and with sufficient quality and understood some of the practicalities that freelance software development teams experience. A key learning experience being that those who deliver get rewarded, while those that fail to deliver, do not. Overall, the students adopted some of the valued behaviours, used the skills and applied the knowledge in the same manner as needed in the workplace. It appears that their understanding of the cultural space in which software developers operate has increased although this “gain” has not been measured either directly or using proxy variables.

Lessons Learned

This novel approach helped students develop both transferable and life-long learning skills. The specific skill set necessary for software developers and engineers includes use of version control systems (VCSs) such as Github and Gitlab, workflow systems, (e.g. Trello) and professional communication via Slack, a cloud-based collaboration tool. Student feedback shows that they better understood the need to take responsibility, produce deliverables on time and with sufficient quality and understood the cultural space in which software developers operate. One of the technical skills developed is the use of regular expressions for rule-based pattern matching. This project used regular expressions in just one programming language, but regular expressions are a feature in many programming languages.

During this course, participants had the opportunity to use the knowledge and skills they had developed during their university studies. Given the tight timeframe of just two months, those with excellent time management skills and strong teamwork skills rose to the challenge. The final deliverables of a video and report provided students with the opportunity to showcase their clear communication skills.

Life-long learning skills were developed by enabling students to function as self-directed, autonomous learners. This project was successful in terms of both developing the learners technical and transferable skills and delivering a practical language learning tool for the target users. Despite this success, a number of lessons were learnt that inform future iterations of this course. These lessons are listed below.

Teams of One

Although developing teamwork skills is of importance to employers, some students who fit the geek loner stereotype complained bitterly about being forced to work in teams. It became clear that some people are simply not suited for teamwork. Not everyone is a team player, and forcing unwilling students to work in teams not only does not create team players, but also results in asymmetric workloads and negative course evaluation (Carroll, Markauskaite, & Calvo, 2007). In future courses, a strong incentive in terms of expected workload is given to encourage students to form teams of four. However, students are given the option to opt out of teamwork by creating a team of one.

Git

Future versions of this course focus more on using GitHub as an online repository. This is because GitHub enables easy tracking of contributions to projects. The default setting is public so it is easy to check which students contributed to what extent. Students can also easily deploy their prototype online directly from this repository and version control is provided as a default. Codepen is much easier to use, but provides no version control.

Video Quality

In the software development stage, usability studies were conducted in which feedback was sought on the software. However, in the video submission stage student teams created the video and submitted. The quality of some videos fell short of expectations for a number of different reasons, including poor sound quality, poor editing skills, forgetting to use PowerPoint in display mode and excessive background noise. In future courses, student teams are required to submit their video to another team for an initial quality check. A simple checklist adapted from the rubrics developed by Brine et al. (2015) is now provided in the online materials to ensure the videos are of a good enough quality to be reused if permission is received.

Deployed Software

Students can also add the prototype of deployed discrete function to their resume using a GitHub address which gives a professional appearance. The resultant tool incorporating all the discrete functions created as a class was deployed, enabling students to add "co-developed language visualization software" to their resume. Students are shown how to create a frame and provide access to the jointly created deployed tool via their preferred web address. This advice was given *ad hoc* during this course, but is now added to the syllabus.

Societal Benefit

This open-source source language visualization tool not only served as the vehicle to help computer science majors acquire technical and transferable skills, but also helps thousands of users on a daily basis throughout the world. Following this course, the functionality of the tool has been extended and version 3.0 is now deployed. Students in future courses could be advised to add a visitor counter to log the number of visitors to their prototype. The realization that users are visiting their site, this might increase their incentive to create an even better tool.

Future Work

Assessments focused on the products of the work, viz. the digital artefacts, for example, software code, video presentations and development reports. These could act as proxies for skills. However, it is possible to assess the skills more directly using a specially developed set of rubrics (e.g. Hu and Shepherd, 2014; Hu et al., 2019). This could be trialled in a future iteration of the course.

References

Barton, J., & Collins, A. (1997). *Portfolio assessment: A handbook for educators. Assessment bookshelf series*. New York: Dale Seymour Publications.

Blake, J. (2018). Pronunciation scaffolder [online tool]. Available https://john6938.github.io/PronunciationScaffolder/ .

Brine, J., Kaneko, E., Heo, Y., Vazhenin, A., & Bateson, G. (2015). Language learning beyond Japanese university classrooms: Video interviewing for study abroad. In *Critical Call–Proceedings of the 2015 EUROCALL Conference* (pp. 91–96).

Brodie, L., Zhou, H., & Gibbons, A. (2015). Steps in developing an advanced software engineering course using problem based learning. *Engineering Education. 3*(1), 2–12. https://doi.org/10.11120/ened.2008.03010002 .

Carroll, N. L., Markauskaite, L., & Calvo, R. A. (2007). E-portfolios for developing transferable skills in a freshman engineering course. *IEEE Transactions on Education, 50*(4), 360–366. https://doi.org/10.1109/TE.2007.907554 .

Chewar, C., & Matthews, S. J. (2016). Lights, camera action! Video deliverables for programming projects. *Journal of Computing Sciences in Colleges, 31*(3), 8–17.

Duch, B. J., Groh, S. E., & Allen, D. E. (2001). Why problem-based learning? A case study of institutional change in

undergraduate education. In B. Duch, S. Groh, & D. Allen (Eds.), *The power of problem-based learning* (pp. 3–11). Sterling, VA: Stylus.

Duffy, G., & Bowe, B. (2010). A framework to develop lifelong learning and transferable skills in an engineering programme. In *3rd International Symposium for Engineering Education.* Ireland: University College Cork.

Foundation for Young Australians (FYA). (2016). *The new basics: Big data reveals the skills young people need for the new work order*. FYA. https://www.fya.org.au/wp-content/uploads/2016/04/The-New-Basics_Update_Web.pdf .

Fallows, S., & Steven, C. (2000). *Integrating key skills in higher education*. London: Kogan Page Publishers.

Feliciano, J., Storey, M. A., & Zagalsky, A. (2016). Student experiences using GitHub in software engineering courses: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 422–431). New York: ACM. https://dl.acm.org/citation.cfm?id=2889195 .

Harden, R. M., Uudlaw, J. M., Ker, J. S., & Mitchell, H. E. (1996). AMEE medical education guide no. 7: Task-based learning: an educational strategy for undergraduate postgraduate and continuing medical education, part 1. *Medical Teacher, 18*(1), 7–13.

Hart Research Associates. (2015). Falling short? College learning and career success. Association of American Colleges and Universities. https://www.aacu.org/leap/public-opinion-research/2015-survey-results .

Hofstadter, D. R. (1980). *Gödel, Escher, Bach: An eternal golden braid: [A metaphorical fugue on minds and machines in the spirit of Lewis Carroll]*. New York: Penguin Books.

Hogan, J. M., & Thomas, R. (2005). Developing the software engineering team. In *Proceedings of the 7th Australasian conference on Computing education*-vol. 42 (pp. 203–210). Australian Computer Society, Inc. https://dl.acm.org/citation.cfm?id=1082424.1082450 .

Hu, H. H., Mayfield, C., & Kussmaul, C. (2019). Special session: Process skills in computer science. *In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 171–172)*. New York: ACM. https://doi.org/10.1145/3287324.3287520 .

Hu, H. H., & Shepherd, T. D. (2014). Teaching CS 1 with POGIL activities and roles. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (pp. 127–132)*. New York: ACM. https://doi.org/10.1145/2538862.2538954fa .

Hyland, T., & Johnson, S. (1998). Of Cabbages and Key skills: Exploding the mythology of core transferable skills in post-school education. *Journal of Further & Higher Education, 22*(2), 163–172. https://doi.org/10.1080/0309877980220205 .

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation.* Cambridge University Press.

Marquardson, J., & Schuetzler, R. M. (2019). Learning by teaching through collaborative tutorial creation: Experience using GitHub and AsciiDoc. *Journal of Information Systems Education, 30(*1), 10–18. http://jise.org/Volume30/n1/JISEv30n1p10.pdf .

National Research Council. (2013). *Education for life and work: Developing transferable knowledge and skills in the 21st century.* National Academies Press.

O'Halloran, D. (2001). Task-based learning: a way of promoting transferable skills in the curriculum. *Journal of Vocational Education & Training, 53*(1), 101–120. https://doi.org/10.1080/13636820100200150 .

Savery, J. R. (2006). Overview of problem-based learning: Definitions and distinctions. *Interdisciplinary Journal of Problem-Based Learning, 1*(1). https://doi.org/10.7771/1541-5015.1002 .

Succi, C., & Canovi, M. (2019). Soft skills to enhance graduate employability: comparing students and employers' perceptions. *Studies in Higher Education*, 1–14. https://doi.org/10.1080/03075079.2019.1585420.

Uden, L., & Dix, A. (2004). Lifelong learning for software engineers. *International Journal of Continuing Engineering Education and Life-long Learning, 14*(1), 101–110. https://doi.org/10.1504/IJCEELL.2004.004578 .