
There may be differences between this version and the published version. You are advised to consult the publisher’s version if you wish to cite from it.

© 2023 Copyright held by the owner/author(s). This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in CEP ’23: Proceedings of 7th Conference on Computing Education Practice. https://doi.org/10.1145/3573260.3573263

http://eprints.gla.ac.uk/285852/

Deposited on: 2 December 2022
How to Learn a New Language: A Novel Introductory Programming Course

MATTHEW BARR, University of Glasgow, UK

‘How to Learn a New Language’ is an introductory programming course, delivered during the first semester of a work-based Software Engineering degree programme, or apprenticeship. Students are introduced to foundational programming concepts, up to and including object-oriented principles. The course takes a language agnostic approach, using examples from a range of programming languages and highlighting the underlying similarities and key differences between them. The course is also designed to expose students to existing codebases from the outset, rather than focusing on writing small programs from scratch. The aim is to create adaptable Software Engineering graduates, capable of mastering whatever technologies they encounter in the workplace. As such, the course helps address concerns that graduates lack important employability skills, such as adaptability, and significant emphasis is placed on working in teams. The course has received positive feedback from students, preparing them to succeed both at university and in the workplace.

CCS Concepts: • Social and professional topics → Software engineering education.

Additional Key Words and Phrases: introductory programming, education, work-based learning, apprenticeships, adaptability

ACM Reference Format:

1 INTRODUCTION

What is it? The ‘How to Learn a New Language’ (HTLANL) course is one of two modules that comprise the first semester of a work-based Software Engineering degree programme [2]. The course runs for eight weeks, with students in class eight out of every ten days, for two hours per day. While the other module covers professional skills and knowledge, the HTLANL course is designed to introduce students to foundational programming concepts, from basic ideas such as variables, scope, and iteration, to object-oriented principles including inheritance, polymorphism, and encapsulation. The HTLANL course takes what might be termed a language agnostic approach, drawing on examples from a range programming languages including Python, Java, JavaScript, C#, and more. Most concepts are illustrated in at least two languages, with the underlying similarities and key differences between languages highlighted. Usually, the differences are merely a matter of syntax or terminology, while the underlying concepts are similar.

Furthermore, the course is designed to expose students to existing codebases as early as possible. For example, in Week 1, students are asked to make specified modifications to a fully functional version of a typical tile matching game [9], built on the Pygame framework. The requested modifications are presented in order of estimated difficulty; in the Pygame exercise, for example, the first task is to move the game’s sound files to a sub-directory, requiring only a simple modification of the file path in the code. Subsequent tasks require a deeper understanding of the code. For example, the next task is to add a penalty (i.e., deduct points) if an invalid move is attempted, then add a high score feature, and so on. The final, more challenging tasks are clearly marked as optional, allowing more confident students to stretch themselves, without placing undue pressure on less experienced students to complete everything. The very last task is to improve the game in some other, unspecified way – without breaking any of the previous additions. The intention here is to provide an opportunity for students to be creative, as an aspect of Software Engineering that may not be obvious to novice programmers who lack the breadth of experience to generate creative solutions to a given problem, let alone choose the problem they wish to solve. Creativity is also a characteristic of programming that is known to appeal to female students [10], and emphasising the creative aspect – albeit subtly – is intended to broaden the appeal of the exercise.

Students are also encouraged to support one another during practical exercises such as these, with stronger students asked to support others within their allocated team. While there are different schools of thought as to whether students should be grouped according to ability or experience [5, 7], the teams here are allocated manually, based on self-reported experience. As such, each team should possess a mixture of previous programming experience, including at least one student who is relatively confident with code, and thus in a position to help their less experienced team mates. The rationale for devising the teams in such a way – and the expectation of peer support – is clearly communicated during class.

Of course, this approach is not without precedent, and the emphasis on gaining practical experience of applying the taught material aligns with educational theory, particularly that related to constructivist approaches to CS education [3]. Other aspects of the course are familiar to CS educators, and draw on best practices related to peer instruction. For example, in addition to encouraging peer learning through group work, the course makes significant use of peer instruction in the form of audience response questions [11]. Here, Mentimeter² is used to pose in-class questions, simply because this is the platform for which the institution has purchased a license. Questions typically require students to predict the output produced by executing a piece of code.

Another feature of the course is the permeable boundary between lecture and lab. Acknowledging that they will divide their time between the workplace, university, and home, each apprentice is allocated a laptop to support their studies, regardless of where

---

1https://www.pygame.org/
2https://www.mentimeter.com/
they are working. Thus, it may be assumed that every student has
their own machine with them in the classroom, obviating the need
to schedule practical sessions in computer labs. As a result, we are
able to take a much more agile approach to delivering the HTLANL
course, moving between taught theory and practical sessions as
required. This blurring of theory and practice is ideally suited to a
work-based degree programme, where the goal is to get our appren-
tices applying what they have learned as soon as possible. Practical
exercises vary in size and complexity, with some larger challenges
being allocated a full two-hour session, and smaller exercises taking
just a few minutes to complete. For example, when dictionaries are
introduced, students are asked to take a few minutes to create and
parse a Python dictionary based on a given JSON data source.

2 BACKGROUND

Why are you doing it? This is a work-based degree, meaning that
students on the course are employed by a range of different com-
panies, each using a different set of programming languages and
technologies. Indeed, even within the same organisation, students
on different teams may be working different technology stacks. Thus,
the taught course cannot focus on a single programming language,
as is generally the norm for introductory programming courses.

This is partly to do with parity of student experience: it might
be perceived as unfair if a subset of students were working with,
say, Java in the workplace, and the course focused solely on Java.
However, the rationale for incorporating a range of programming
languages (and, eventually, programming paradigms) is more peda-
gogically fundamental. The aim is to prepare students for the reality
of working as software engineers in industry, where technology
never stands still, and where individual projects may require a work-
ing knowledge of more than one programming language.

Having students work in teams of mixed ability and experience
also reflects the nature of many software engineering teams in
industry. One argument in favour of teams of mixed ability on a
work-based programme is to normalise the act of asking for (and
receiving) help. Having the confidence to ask for help is an important
attribute to possess in the workplace, as is having the confidence to
provide help where required [1]. Of course, the pedagogical benefits
of peer discussion in Computing Science – not only on a work-based
programme – are well documented [12], and equally relevant here.

Where does it fit? As alluded to above, the course is delivered
within the context of a work-based degree programme, or appren-
ticeship. However, it is important to note that the taught component
of the Graduate Apprenticeship programme is delivered on a uni-
versity campus. Over the four years of the programme, apprentices
spend around 80% of their time in the workplace, as junior software
engineers, but they also study alongside our traditional undergrad-
uate CS students in their third and fourth years. Thus, the HTLANL
course must prepare students both for working in industry and for
studying at university.

Ultimately, HTLANL aims to help create adaptable Software Engi-
eering graduates, capable of mastering whatever technologies they
encounter in the workplace during their apprenticeship, and beyond.
As such, the course helps address concerns that graduates lack im-
portant employability skills, such as adaptability [6]. The course is
also intended to prepare apprentices for working in industry, with
the emphasis on working as part of a team and early exposure to
existing codebases. This relates to the point about confidence above,
as it is known that CS students’ perceptions of professional prac-
tice can affect their self-efficacy, and whether they persist with a
challenging task [8].

Colloquial terms such as ‘in at the deep end’ and ‘ripping the
bandage off’ are frequently used to describe our approach to the
HTLANL course, and this is intentional: as daunting as it may be,
it is better that our apprentices’ first experience of being asked
to modify a complex piece of existing code takes place within the
classroom environment, rather than in the workplace. To quote
Berland, cited in Blikstein & Moghadam (2019), the course creates
“safe spaces to fail, to play, to tinker ... This is where you get the
bang for the buck. That’s where the learning happens” [4].

3 EVALUATION

Does it work? The course has now run for three years, with student
feedback collected at two points during course delivery: formative
feedback at the half-way point (at the end of Week 4) and summative
feedback (after Week 8).

The formative feedback is administered directly by the course
coordinator, via the virtual learning environment, and asks students
to answer a number of open questions, including: Which methods of
learning or aspects of the course would you like me to reduce? While
the responses to these formative feedback questions are generally
brief (“I like the format of this course the way it is”; “everything
works well”), some themes do emerge. Certainly, the use of Menti-
ter is appreciated by students: “I quite like your approach towards
using mentimeter and actively engaging with us”; “Mentimeter is
great. It really helps me to check if I’ve followed what I’ve been
taught in real time”). Feedback also indicates that students would,
variously, like to see more of the “Interactive segments”, “group
work with mentors”, and ‘Baby Yoda’. However, these data are too
sparse to support any serious conclusions.

Summative course feedback is administered centrally by the uni-
versity, via the EvaSys evaluation system. The EvaSys surveys ask
students to rate the quality of the course and its delivery, and asks
two open questions: What was good about the course? and How could
this course be improved? Many of the responses to these questions
do not relate directly to the discussion here – many refer to the
instructor’s delivery, for example – but some students do comment
on the course design. For example, one student from 2021 notes that
“Lectures were engaging and funny, course was very useful and has
been the best method of learning a language I’ve come [sic] across
so far”. Another, from 2019, notes the “great range of topics covered,
covering many different languages to get a broader knowledge, great
eamples”. Others list the opportunities for peer learning under what
was good about the course; for example, “Collaboration, peer to peer
learning, seeing how others tackle problems that I may do differently”
(2019 cohort). Many students comment on the engaging nature of
the sessions, referring to the practical work that is interspersed with
the taught material, for example, “Good practical examples presented
in a fun way” (2019); “Classes are engaging and fun, breaking up the
monotony of many lectures” (2020).
In terms of what could be improved, students have offered relatively few suggestions. However, more than one student noted that it would be useful to allocate additional time to getting their development environments set up during class, for example: “A bit more time could have been spent at the start helping people get IDEs setup, as later weeks were impacted due to IDE errors in labs.” (2020); “programmes such as IDEs should have a separate time for set up as issues with them ate into lab time” (also 2020). To address this, students are now encouraged to install the required software in advance, so that the number of issues encountered during class time is reduced. This is not unreasonable when apprentices are expected – and paid – to be working or studying between scheduled classes. Notably, feedback for 2021 did not make mention of these concerns.

Finally, an unexpected source of feedback on the course came from a series of reports submitted by apprentices later in the academic year, as part of their workplace assessment. In one such report, apprentices are asked to reflect on the meta-skills – transferable skills such as communicating and adapting – they have developed in the workplace (see [1]). Given that these reports are written several months after the end of the HTLANL course, and following an extended period in the workplace, it is interesting to note that the course is mentioned explicitly by several students. For example, the following student refers to the course in relation to developing their ‘sense-making’ meta-skill:

I think this apprenticeship has greatly improved my skills when it comes to large codebases. Before we had started university, I got the chance to look at some of the code for my project at work. I didn’t know how to tackle understanding the code. It was intimidating and felt hopeless. How To Learn New Language taught us how to break down big pieces of code and understand the functionality first and then we can understand how it works as a whole.

In another of these workplace reports, apprentices are asked to address the question: What have you learned at university this year that you feel will remain relevant to your work in the years to come? Here, again, the HTLANL course is acknowledged, for example:

That HTLANL formalized a process of identifying components and concepts within a language was a very successful approach for me. It gave me the mental toolkit to do things such as compare Java’s arrays and Python’s lists, and understand that their uses are not identical.

4 CONCLUSION

‘How to Learn a New Language’ represented a significant departure from our traditional introductory programming courses. As a work-based degree programme, however, the Graduate Apprenticeship in Software Engineering necessitated a different approach, in order to adequately prepare students for working as software engineers in a wide range of employer contexts. With our first cohort now entering their final year of the programme, we can say with some confidence that the approach outlined here has worked: our apprentices are succeeding in the workplace, and student feedback suggests that this course, specifically, has played an important role in that success.

Furthermore, by developing our students’ confidence and adaptability, we believe that we are helping to create software engineers that possess the tools to succeed as their career develops beyond the apprenticeship.

We also believe that our approach is of wider interest, beyond work-based degree programmes. If the goal of a CS programme is to create competent, adaptable graduates, then perhaps the kind of course described here should be the norm. Certainly, the ability to work as part of a heterogeneous team, with a range of programming languages, and on large existing codebases has proven to be beneficial to our apprentice software engineers.

ACKNOWLEDGMENTS

The development of the Graduate Apprenticeship programme was supported by Skills Development Scotland.

REFERENCES