Learning Programming Languages by Pantomime

Andrew Fagan
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
andrew.fagan@strath.ac.uk

Alasdair Lambert
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
alasdair.lambert@strath.ac.uk

Martin Goodfellow
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
martin.h.goodfellow@strath.ac.uk

Abstract

It is challenging to teach students new programming languages by lecturing while retaining interest and student engagement - direct lectures on syntax are inherently dry, and the benefits of more hands-on approaches to allow learners to make mistakes and experiment are well documented.

This paper is based on a year teaching two languages, C and Haskell, inspired by the concepts of both cognitive apprenticeship and team teaching. Both of these languages are widely considered to be difficult for new users to learn, and the classes in question have in the past been deemed as challenging and unpopular by students.

We describe our approach, then demonstrate that the application of these methods led to these previously unpopular classes becoming well-regarded by students. We support this by way of a survey, and provide some analysis of the source of this improvement and give some qualitative insights and other incidental benefits.

CCS Concepts

• **Social and professional topics** → **Computing education**.

Keywords

Introductory Programming, Live Programming, Cognitive Apprenticeship, Team Teaching

ACM Reference Format:
Andrew Fagan, Alasdair Lambert, and Martin Goodfellow. 2025. Learning Programming Languages by Pantomime. In *Computing Education Practice (CEP '25), January 07, 2025, Durham, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3702212.3702213

1 Introduction

Anyone who has taught or learned a programming language in a higher education context can attest to the fact that lectures are in constant danger of becoming a long and noninteractive way to disappoint and disengage students. While it is critical to learn the grammar of the language, the nuts-and-bolts of variables and memory management, and in which particular-yet-different way a `for`loop is implemented, it does not make for especially dynamic or interesting content. This is in addition to the deeply held instinct of all programmers that there is little substitute for the experience that comes from actually designing, writing and running a piece of code (then realising your mistakes, rewriting and rerunning it).

In this paper we detail our use of a live programming approach to deliver two introductory programming modules CS260 – an introductory functional programming course using Haskell, and CS210 – which first introduces programming in C then uses it as a vehicle for exploring operating systems concepts. These are large compulsory classes for second year students within the University of Strathclyde, with approximately 160 students each. The students enrolled in these classes have completed a year of Java programming and as such only have basic programming experience. The topics these classes cover are generally considered difficult and have in the past been received poorly by students.

In recent years these classes have shifted their delivery model from slides to live programming. This has brought substantial benefits to both classes, as by necessity the students are immersed in each language and the need to focus on details such as syntax is reduced. By bridging the gap between lecturing syntax and design work, students are able to extract far more value from lectures.

In addition, we have merged this with a multi-lecturer style. Each live programming session consists of two or more lecturers who are free to interact in a variety of ways. Most commonly, the lecture is planned ahead of time and split into topics with representative code examples or problems to be solved. The lecturing team will then, starting with an empty text file or cut down template, attempt to recreate these code examples live for the benefit of the students.

The lecturers can also choose to plan specific interactions, such as deliberate mistakes for the other to point out and fix. These can be planned to help understanding or as fun pantomime routines to improve student engagement. It has been discovered by rigorous testing that jokes don't necessarily need to be "funny" to improve engagement, much to the relief of the second author in particular.

2 Background

This work is about combining live programming with a multi-lecturer approach. It is important to consider the pedagogical basis and conventionally stated advantages of each of these approaches, before considering how best to incorporate them into a single approach. With this in mind, we present summaries of the existing literature on Cognitive Apprenticeship and Team Teaching.

2.1 Cognitive Apprenticeship

Programming can be a difficult skill to teach in a classic lecture setting. Slides full of code can overwhelm students and removing it from its natural context of working to solve a problem means that while students may be able to understand examples they may struggle to solve problems. Added on to this are the onerous details such as syntax, style and programming environments. There is a body

CEP '25, January 07, 2025, Durham, United Kingdom

ACM ISBN 979-8-4007-1172-5/25/01
https://doi.org/10.1145/3702212.3702213

of literature showing that live programming is an effective tool for introductory programming classes to address these and other shortcomings [5, 7]. This is where a lecturer actively writes code during a lecture, solving problems using the techniques introduced. This approach naturally dovetails with the cognitive apprenticeship model of education where the "master" explains their thought process to the "apprentice" in order to make intrinsic knowledge apparent [2]. While the master has great understanding of their trade it can be easy to assume knowledge that the master does not even need to consider. The cognitive apprenticeship model is designed to draw this assumed knowledge out so that it can passed on to the apprentice rather than assuming that they will pick it up in their own time. In the setting of live programming it is very possible that a single lecturer will not fully employ the cognitive apprenticeship approach and will fall into the same pitfalls of assuming implicit knowledge is obvious.

2.2 Team Teaching

Team teaching refers generally to any teaching with more than one teacher. Different styles of team teaching can be broadly categorised according to the relationship between the teachers [1]. For our purposes, the most useful of these models to consider is the Teaming model, wherein two or more teachers share every aspect of the teaching process equitably, collaborating on preparation, planning and delivery. In particular it means that multiple lecturers give each lecture. While this seems to double the work needed for each lecture we present evidence to suggest that it can bring both staff and students significant benefits.

This model has been applied in a CS context previously. This approach can improve student engagement and understanding while providing junior staff training, reducing some of the perceived cost of pair lecturing [4]. It may seem wasteful to have two senior staff deliver the same lectures however contact time is often not the expensive part of delivery. In fact, preparation, administration and organisation tend to consume the most time. It has also been argued that pair lecturing provides similar benefits to pair programming [8] such as pair pressure, pair relaying and pair review, allowing the plan of a lecture to be improved by collaboration. The authors also mention that junior staff can be used as a critical friend - a trusted figure that can ask critical questions about current understanding or thoughts in order to provoke a different point of view or to facilitate critical thinking [3]. A final interesting benefit is that team teaching can be used in conjunction with the cognitive apprenticeship model where trainee teachers follow the traditional steps of the cognitive apprenticeship model and gradually take over more responsibility for the course as they learn [6]. This makes a very strong argument for this approach in the training of new staff.

3 Our Approach

While it is clear that there is potential synergy between team teaching and cognitive apprenticeship in the sense that one teacher can be the master of teaching, while one acts as the apprentice and learns to teach more effectively, our focus going forward will be on fundamentally combining the two for a true cognitive apprenticeship based teaching approach which utilises two masters and has the students fill the apprentice role.

Due to a quirk of administration, both CS260 and CS210 found themselves with multiple lecturers assigned to a fraction of the class. Since it was assumed that this would be changed going forward, the staff involved decided to deliver lectures collaboratively - initially simply to minimise the difficulty of handover. In fact, with only a slight change in approach, the lecturers discovered that this lead to a wide variety of benefits to both staff and students.

Under this new approach, every lecture for both classes was delivered by at least two lecturers almost entirely through live coding. Examples were generally designed to show common mistakes which could be theatrically corrected by the other lecturer. In the authors' experience the ability to plan fun, if often silly, routines made lecture preparation a much more enjoyable experience. During the lectures staff would take turns on their parts of the content with their colleague free to join in or provide alternative explanations.

While the authors do not imagine that they are the first to live code with the participation of multiple teachers, we have found no other research which addresses this synergy, and therefore to the best of our knowledge this combination - and in particular our analysis of its benefits - is academically novel.

Outside of lectures both classes followed a similar structure. Weekly lab sessions were held where students could work through problems on the week's topics with the support of Teaching Assistants. This aimed to provide the coaching and scaffolding phases of the cognitive apprenticeship where the apprentice is asked to work with support of an expert.

3.1 Co-lectured live coding in practice

Below, we present an illustrative example of our approach, based loosely on real events. This example shows how lecturing responsibility can be shared and how the lecturer not currently speaking can still contribute. Readers unfamiliar with the C programming language may feel free to mentally substitute phrases they do not recognise for topics of their choice from another language or field - the example should still be comprehensible. Our principle actors we shall call Fred and Andrew, as these are their names.

Fred begins a lecture on C, with today's topic ostensibly being how to include and utilise a local library. He is assured by Andrew that a library, consisting of `stack.h` and `stack.c`, has been prepared in advance for his use. He consults `stack.h` and begins writing a program utilising it, explaining his design process to the students, while Andrew asks questions from a prepared bank if no student does so. The students sense the more conversational tone the lecture is taking, and some seem to feel more confident asking questions or making comments.

Eventually, Fred compiles the code, and - to his great surprise and distress - the compilation fails. The students laugh at his obvious irritation, and a few who had been sleeping are jerked awake by this change from the apparent plan. Fred discovers that `stack.c` has been overwritten and is now empty. He asks Andrew as the author of the file whether he might be able to reproduce it, and so Andrew takes over the lecture.

Andrew begins showing the students his process of analysing Fred's code and the `stack.h` file to create signatures for each of the functions he will have to recreate. In addressing this "mistake", he finds the opportunity to give the students an early glimpse at

next week's topic - memory allocation in C - which he claims they had not planned to cover today. Fred asks some questions about memory allocation to prompt a useful discussion and Andrew adds some notes in comments around the code he was writing.

As Andrew continues writing code, the attention of the class begins to wander during one of the more repetitive parts - writing the `peek` function which is fairly similar to the `pop` function he has written already. Noticing this, Fred begins speaking to the class about how the scope of variables works when code is split across multiple files - a topic they had intended to discuss later on, but which could easily be moved up to fill this gap. He makes a note to tighten up this part of the lecture next year, allowing them to address it instead of forgetting this minor point as Andrew might have alone.

Andrew completes work on `stack.c`, and so he hands back over to Fred. They have a brief discussion as to why the `struct` he defined, `struct STACK` needs to be in the `stack.h` file when it seems like it could be in the `stack.c` file instead, which inspires several students to ask questions.

Finally, Fred compiles the code and discovers, this time to his actual confusion that the code doesn't compile. Andrew spots that he has failed to compile `stack.c` and has only compiled the main program file, he points this out and then gives an explanation of the linking process, playing off an actual mistake as part of the plan.

The astute reader might realise at this stage that the majority of the "mistakes" made in the course of the lecture were strictly theatrical, and were in fact part of the plan all along. Despite the lecturers' claims to the class that they were improvising and moving topics around, they were in fact presenting each topic in a context in which it could be used for problem solving. This element of feigned spontaneity aids with the cognitive apprenticeship approach in that it gives the impression of the class being a team solving a problem collaboratively. In the experience of the authors, this approach is challenging without a partner - it is fairly taxing to convincingly maintain the act alone, especially if the students don't choose to engage in the apprentice role. This also allowed a real mistake to be disguised as a further piece of theater towards the end, preserving the student's perceptions of the lecturers as knowledgeable.

4 Analysis

Our results show that the students surveyed were overwhelmingly in favour of our model of pair teaching. This analysis is based on the survey results* presented in Figure 1 and Table 1. In each quantitative question asked, the majority of students believed that pair teaching had improved their understanding and engagement. While the survey did not directly ask students to compare to classes which use a live programming cognitive apprenticeship approach, many of their other classes do use that approach, and the clear consensus was a preference for our model.

4.1 Benefits to students

Based on our results and feedback from our student cohort we identify several key benefits for students.

*This survey was targeted at all students in the two classes under study, and was carried out with approval from the Ethics committee of the University of Strathclyde's Computer and Information Sciences Department.

Engagement and interaction. Our results show the overwhelming majority of students believe they were more engaged during pair lectured classes, and the results shown in table 1 show that pair lectured classes were preferred by most students to other styles. Pair lecturing can be used to provide a clear split between topics, this helps mark out the start of a new topic or concept. At a more basic level the change of voice can jolt students awake and so it can be helpful to swap lecturers after a topic which is expected to be less engaging.

If used well, pair lectures can have a more conversational tone as the lecturers interject and consult each other. This helps to break down the usual formality and encourages student questions. Interjections can be helpful but it is important that both lecturers are aware of the learning outcomes the other is trying to deliver.

Table 1: Question: Please rank the following based on your own preference

Ranking	1st	2nd	3rd
Co-lectured classes (multiple lecturers in the room at any one time)	41	4	3
Classes with multiple lecturers, but which are traditionally lectured (a team of lecturers over the course of a module, but with each lecture given by a single member of the team at a time)	4	24	20
Classes which are only traditionally lectured (one lecturer who leads the module and gives every lecture alone)	3	20	25

Multiple Perspectives. While one lecturer usually takes the lead, the other should be prepared to jump in, providing alternative perspectives on ideas being introduced or alternative explanations. In general no single explanation will reach every student, and having the option to ask a colleague for an alternative opinion can significantly improve student understanding. Our responses show that students believed pair lecturing had improved their understanding of the topic. It is the authors' belief that this is the case as a result of the two above points however this is only the students opinion and more data would be needed to show this concretely.

4.2 Benefits to staff

Our survey results show that our approach to pair lecturing is very popular and provides many benefits to students. However, this model provides substantial benefits to staff as well.

Lower cost of planning. Having two colleagues brainstorm together it can make it much easier to generate new ideas on how to present material. Overall we found that planning time was significantly reduced in contrast to the findings of [8]. It is often easy to leave course content as it is year in year out, with a new member of staff joining an existing course providing a shot of energy for course redevelopment. It is possible in the case of two expert lecturers who both know the content very well - and are comfortable working together - to create a lecture with only a few minutes of preparation time, or even none at all, creating even greater time savings.

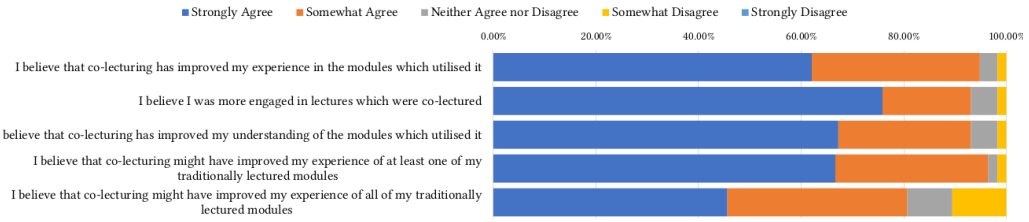

Figure 1: Responses to the Survey

Training for junior staff. Pairing a more experienced lecturer with a more junior member of staff provides significant on-the-job training. The joint planning gives the same benefits as the cognitive apprenticeship model discussed in Riehm et al. [6]. In particular, the junior member of staff learns why their colleagues make the choices they do when presenting material. This also helps to transfer institutional knowledge to new staff and could be a useful way to onboard new staff.

Redundancy. While co-lecturing has the myriad of benefits we've explored, a live programming lecture which has been prepared for two lecturers can also include a plan to give the lecture alone. Should either lecturer be absent for any reason their colleague can deliver the lecture traditionally, this also allows staff more flexibility to take annual leave during term. Arguably this is also a benefit to students as it avoids disruption caused by staff absence.

Workload Management. Lecturers might consider dividing preparatory time in such a way as to optimise their individual workloads. Generally contact time with students is cheap compared to designing lectures and creating materials. We have found that this paradoxically reduces the cost of running classes, with fewer preparatory hours for the class lecturer - usually the member of the teaching team whose time is most expensive - at the cost of additional hours from a second, usually more junior member of staff. Even in the case of staff on the same level, it is probable that the total number of preparation plus contact hours would be reduced, but a more exhaustive study would be required to make this point concretely.

5 Further Work

This model works well in the context of introductory programming and it is likely not an ideal model for every context and topic. However, given that such modules are core to any CS undergraduate program our model provides significant scope to improve both student engagement and understanding.

As our data is limited we would initially like to conduct a larger survey. Ideally this would include more classes, multiple year groups and potentially other institutions. This would significantly improve on the limitations of our data and provide a stronger argument for the merits of our approach.

As with our data our results would have been improved by having more classes which use a live programming cognitive apprenticeship model without pair lecturing for comparison. Both methods are popular with students so more classes with this model would act as a control so that pair lecturing could be isolated as the difference.

Another beneficial control would increasing the pool of lecturers used. This would strengthen our argument that the model is correct instead of only working for the small number of lecturers involved.

6 Conclusion

We present a model for introductory programming modules which combines the known benefits of both the cognitive apprenticeship approach with those of live programming and pair lecturing. Our approach does not claim to be a panacea however we do believe that there are significant benefits to both staff and students with this approach. In particular, students can benefit from better engagement and improved understanding, while staff can enjoy support in what is generally a solitary role. We also believe that these benefits can be achieved without increasing staffing costs, or with some potential to reduce them.

Acknowledgments

We would like to thank our co-lecturers Drs. Jules Hedges, Fredrik Nordvall Forsberg, John Levine and Conor McBride, as well as Graham and Claire for their help with several Pressing issues.

References

[1] Marlies Baeten and Mathea Simons. 2014. Student teachers' team teaching: Models, effects, and conditions for implementation. *Teaching and Teacher Education* 41 (7 2014), 92–110. https://doi.org/10.1016/J.TATE.2014.03.010

[2] Allan Collins, John Seely Brown, Ann Holum, et al. 1991. Cognitive apprenticeship: Making thinking visible. *American educator* 15, 3 (1991), 6–11.

[3] Arthur L Costa, Bena Kallick, et al. 1993. Through the lens of a critical friend. *Educational leadership* 51 (1993), 49–49.

[4] Grischa Liebel, Håkan Burden, and Rogardt Heldal. 2017. For free: continuity and change by team teaching. *Teaching in Higher Education* 22 (1 2017), 62–77. Issue 1. https://doi.org/10.1080/13562517.2016.1221811

[5] John Paxton. 2002. Live programming as a lecture technique. *J. Comput. Sci. Coll.* 18, 2 (dec 2002), 51–56.

[6] Katja Riehm, Eva Hellmuth, Kevin Eichhard, and Thomas Waitz. [n. d.]. Teamteaching Chemistry: a Concept to Promote the Acquisition of Professional Knowledge in the Context of University Internships. ([n. d.]).

[7] Marc J. Rubin. 2013. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 651–656. https://doi.org/10.1145/2445196.2445388

[8] Daniela Zehetmeier, Axel Böttcher, and Anne Brüggemann-Klein. 2018. Designing Lectures as a Team and Teaching in Pairs. *4th International Conference on Higher Education Advances (HEAD'18)* (7 2018), 873–880. https://doi.org/10.4995/HEAD18.2018.8103