Teaching of Computing in the Engineering Tripos

A working party of the Mathematics and Computing Subject Group has been considering some major changes to the way we teach computing in the Engineering Tripos. Their report is attached, the main recommendations being:

- Replace C++ with Python in Part IA
- Continue in Part IB with further Python and a new C++ device programming activity

The expectation is that all Part II computing activities, as well as any remaining use of Matlab in Part I, will remain viable. While a detailed plan is yet to be drawn up, it would be helpful to receive initial reactions from the SSJC. We would particularly welcome members' thoughts as to how to fully engage all students in computing in the absence of a formal examination. Alternatively, how might a formal computing examination be received by the student body?

The earliest date any new courses might commence is Michaelmas 2016 in Part IA.

Andrew Gee
11 May 2015

Report of the MCSG Working Party on the Teaching of Computing

26th April 2015 (revision 2)

Summary

This report was prepared by a working party appointed by the Mathematics and Computing Subject Group to review the objectives of computing teaching in Part I of the Engineering Tripos. The Terms of Reference included considering whether there are pedagogical advantages to changing the teaching language. Following a review of how computing is presently taught, consideration of how computing is taught elsewhere at comparable institutions and current trends in computing and engineering, we recommend that the teaching of computing in Part I be restructured to focus more on computing as an accessible and indispensable tool in modern engineering and to provide students with positive experiences of computing. In particular, it is recommended that teaching be restructured to:

1. Increase the emphasis on problem solving, software engineering and good software development practice (especially when working in teams).
2. Decrease the emphasis on programming as an independent activity.

To support the above objectives, it is recommended to:

3. Switch from C/C++ to a high-level, scripted language as the primary language used to support the teaching of computing.
4. Use Python as the primary teaching language in Part I.
5. Support an online, notebook-style programming environment to reduce the threshold for using computing as a tool in the Tripos.
6. Introduce a device programming activity in Part IB to expose students to a compiled language (C or C++).

In the event of a change in Part I teaching, we believe that all current Part II activities will remain viable. Leaders of activities that make use of a programming language should remain free to choose the language that best supports the learning objectives of the specific activity. A change in Part I has resource implications, and a number of staff have volunteered to support the changes that would be necessary.

The members of the Working Party are listed in Appendix A. The section numbers in the report correspond to points in the Terms of Reference (see Appendix B).

1 Objectives of Part I computing teaching

The primary objective of Part I computing teaching should be to teach computing as an invaluable and accessible tool/framework for solving engineering problems. At the end of Michaelmas Term, Part IA students should:

i. See computers as accessible tools to support their own study and work.

ii. Have a positive overall experience with regard to computing on which to develop further skills.

These points should be reinforced throughout Part I.

Awareness Students should become aware of:

iii. How computers are used for technical applications, including numerical computation, computer algebra and visualisation.

iv. The basics of how computers operate and how this can affect problems solved using a computer (e.g. precision).

v. What are algorithms and elementary complexity analysis.

vi. Elements of good software engineering practice.

vii. The main differences between language models, e.g. scripted versus compiled languages.

Skills Skills that students should develop during Part I include:

viii. The ability to translate a problem into an algorithm.

ix. The ability to translate an algorithm into a working computer program.

x. The ability to effectively test: (a) programs that they develop; and (b) programs that they use.

xi. The ability to use different programming paradigms, e.g. procedural and object-oriented.

xii. The ability to employ good software development/engineering practice in the development of their own programs and when working in teams, including the use of version control and systematic code testing.

xiii. The ability to select appropriate data structures for specific problems.

xiv. The ability to write simple scripts in an appropriate language to automate processes.

Primary teaching language The listed aims and objectives do not include learning a programming language. Ideally, the teaching of computing would be agnostic with respect to language. However, it is important that students write programs to reach the learning objectives and to develop their skills. The Department must be able to support students in this activity, which necessitates the choice of a programming language(s) for teaching. A language used in teaching at Part I should be as unobtrusive as possible, accessible, support the primary learning objectives and serve students going forward into different disciplines.

Positive first experience A positive first experience is important for encouraging students to see computing as an accessible tool that they can employ in other parts of the Tripos and beyond. The choice of programming language has a major impact on students' first experiences of computing.

Computing versus programming The distinction between the teaching of computing and the teaching of programming is often blurred. The current Part IA course has a heavy emphasis on the learning of a programming language (C/C++), and the consensus in the Working Party is that the current Part I emphasis on the teaching of programming is too heavy.

1.a Changes in student skills, expectations and devices

Student skills The variability in the skills of incoming students with respect to computing and programming is large. Many students have limited or no prior technical computing experience and no programming experience. A small fraction of students are highly skilled in computing and programming. With recent syllabus changes at primary and secondary school level, it is likely that future students will have more exposure to computing, but it will be some time before we see this.

Student expectations With the limited exposure that most incoming students have had to computing, it is hypothesised that most students have either limited or naive expectations of computing for engineering.

Devices and services Almost all incoming students now have a laptop computer, and all can access the Internet from colleges and at the Department. Major recent changes that are relevant to this point and how we teach computing going forward are:

- Access to non-proprietary, cloud-based programming environments, e.g. `http://jupyter.org/`, `https://cloud.sagemath.com` and `https://wakari.io/`.
- Online development and version control systems, e.g. `https://github.com/` and `https://bitbucket.org/`.

Cloud-based services can substantially lower the threshold for students to apply computing skills, remove operating system dependence (and hence any installation support burden), and allow students to compute from lightweight devices, e.g. tablets. It also opens new possibilities for how students can be supported in their learning of computing.

We note that MET teaching is now based around tablet computers.

1.b Pragmatism, software engineering and multi-lingualism

Pragmatic problem solving and software engineering The Working Party does not see any conflict between pragmatic problem solving and software engineering. We should aim to teach students to solve problems reliably, robustly and sustainably. Good software engineering supports these aims. Subject to a suitable change in the teaching language, we recommend an increased emphasis on software engineering in Part I, especially in light of the recent leap in the accessibility and ease of use of tools that support good software development practice, such as cloud-based version control.

Multi-lingualism In loosening the teaching of computing from programming languages, there was support within the Working Party for making students more aware of different languages and providing more advanced students with the opportunity to explore other languages via unsupported exercises.

Multi-lingual approaches are a feature of much of modern software engineering. The application of multi-lingual approaches is too complicated at Part I level, but students should be made aware of why multi-lingual approaches are used, specifically with respect to compiled versus scripted languages.

1.c Supporting Part II teaching and the needs of industry

Part II teaching The present use of computing and programming in Part II is diverse (see survey in Appendix C), with a number of languages used. The use of C++, the teaching language in Part IA, in Part II is surprisingly limited. Most Part II courses that involve programming require very limited (even disappointing) independent program development by students. On the basis of the survey, we conclude that Part II courses that involve computing are best supported by Part I teaching emphasising computing skills, and the choice of Part I primary teaching language is not a significant concern.

Industry The industries to which graduates go are diverse, and the details of the computing practices, even in closely related fields, differ. The development of strong general computing skills is compatible with the needs of industry. As part of equipping students for life beyond the Tripos, some members of the Working Party felt very strongly that we should use viable non-proprietary languages in teaching, especially in Part I.

2 Survey of current Part I and Part II teaching activities that involve the use of a programming language

A summary of teaching activities that involve a programming language, as best as could be ascertained, are presented in Appendix C.

2.a Assessment of any advantages to switching teaching languages

Comparable institutions There was a very strong consensus within the Working Party that there are major advantages to switching the primary teaching language in Part I. Anecdotal evidence suggests the teaching of computing in the Department is out of step and behind innovation elsewhere at comparable institutions.

Current state C++ is currently taught in Part I. C++ is a difficult language to learn and the complexity of C++ limits the extent to which its features are taught to students in Part I. It is a verbose language, with many syntactic constructs that obscure a program for the inexperienced (and even experienced!) C++ programmer. As a compiled language with limited support for introspection, it is difficult and time consuming for students to 'explore' and check their programs. Within the University, others involved in teaching actively advise against learning C++ as a first language[1].

Alternative teaching languages It is not possible to properly consider the advantages of switching teaching language(s) without consideration of viable alternatives. There are strong pedagogical arguments for switching to a general purpose, high-level scripted language that has a rich set of standard, easy to use data structures, graphical tools and good support for numerical computation. In an appropriate language, algorithms and programs are less obscured by syntax. In teaching, the availability of built-in data structures shifts the emphasis in introductory teaching from implementing low-level data structures to using standard data structures to solve more sophisticated problems.

Switching to a scripted language opens the possibility for students to use online or cloud-based services to easily and readily write programs, and to present and annotate their work. This has a number of teaching advantages, including lowering the threshold for use.

Of the high-level, scripted languages that are widely used and/or promising for technical computing, prominent examples are Julia, MATLAB and Python. While promising and used in teaching elsewhere, Julia is not yet sufficiently mature for adoption as a primary teaching language. It does however offer a number of novel features that would permit interesting unsupported extension exercises for advanced students. Some members of the Working Party (and others in the Department) feel very strongly that we should use open, non-proprietary languages over proprietary languages when high quality implementations exist. It was also felt that students should first be exposed to a general purpose, rather than a domain-specific, language. This points to Python over MATLAB as the more suitable teaching language in Part IA. Adopting Python as a teaching language would align the Department with numerous comparable institutions.

Compiled languages A number of teaching staff in the Department feel that it is important to retain some teaching of a compiled language in Part I. To address this concern, it is proposed to introduce a device programming exercise in Part IB that uses C or C++. The activity will address a problem for which a compiled language is necessary. This will prevent the perception that the exercise could have been achieved with less effort using a high level language, and thereby make it a computing rather than a programming exercise.

2.b Effort required to make any changes

The estimated effort required to translate the current Part I teaching exercises to a high-level language (in which a number of staff and students are experienced) is limited. As part of the investigations

[1] `http://people.ds.cam.ac.uk/nmm1/C++/index.html`, `http://www.cl.cam.ac.uk/freshers/`

of the Working Party, the Part IA Michaelmas Term C++ computing exercises were developed and extended in Python[2] (in an online IPython Notebook), and this took only several hours.

In changing the IA Michaelmas Term C++ exercises to Python, the exercises became considerably shorter, and in cases trivial. For Part I, it is estimated that switching to a high-level language will free some 1/2 to 2/3 of time currently spent on computing teaching and exercises. The Part I material will therefore need to be considerably extended in terms of problem solving and software design.

Changes to other activities, e.g. the Mars Lander exercise, will require thought as to what form an updated exercise should take. It is natural that a change in language will lend itself to some changes in an exercise. In the case of the Mars Lander exercise, the core student task is largely independent of the language and is straightforward in a high-level language (this part has already been done in both Julia and in Python). The substantial code provided to students to create the graphical interface is tied tightly to C/C++. In such a case, options to consider include providing wrappers around the existing code, or supporting students in developing their own graphical interfaces using high-level libraries/tools.

2.c Willing helpers

A pleasing number of staff members have expressed a willingness to assist with updating teaching material if there is a change in the teaching language. It is anticipated that willing students can also be recruited, if necessary. A number of staff members who have been closely involved in the development of Part I computing teaching have expressed a willingness to develop new, extended material if the teaching language is changed to Python.

2.d Viability of teaching activities in the event of a Part I change

Based on the similarities between Python/NumPy and MATLAB, we believe that all current teaching activities will remain viable in the event that the Part I teaching language is changed.

A new embedded device activity in Part IB using C/C++ will provide students with some C/C++ experience in preparation for the few Part II activities that use C++. In addition, C++ support lectures targeting Part II students that are already offered can be recommended to students and students can be directed to some of the freely available, excellent online learning material.

A Working Party members

- Bill Byrne
- Alex Kabla
- Tim Love
- Sumeet Singh
- Garth Wells (Chair)
- Jim Woodhouse

B Terms of Reference

1. To define the aims and objectives of computing teaching in Part I of the Engineering Tripos, the deliberations to include but not be limited to:

   (a) the impact of recent changes in incoming students' skills, expectations and devices (e.g. web, apps, Raspberry Pi, Scratch, tablets vs laptops), and also the natural divergence of skills over the four years of the course;

[2] `http://nbviewer.ipython.org/url/bitbucket.org/garth-wells/cued-computing-examples/raw/master/python/IA-Michaelmas.ipynb`

(b) to what degree our aims are pragmatic (i.e. to empower undergraduates to apply computing to real world engineering problems, perhaps by ad hoc means), and to what degree we might aim to teach Software Engineering as a formal discipline in its own right. What about multilingualism and the design aspects of Software Engineering?

(c) whether we prioritize the needs of Part II and/or the needs of industry

2. To survey all Part I and Part II Engineering teaching activities that involve the use of a programming language (including laboratories, projects, examples papers, vacation exercises and lecture courses) and establish:

   (a) whether there would be any pedagogical (or other) advantage in switching to, or offering in addition, other languages, and if so which languages;

   (b) approximately how many person-hours would be required to make the change;

   (c) whether there is a willing and able person ready to do the necessary work; and

   (d) whether the activity would remain viable in the event that the programming language it relies on is no longer covered in Part I.

3. To report back to the MCSG at its February 2015 meeting before considering possible changes to the computing languages and activities offered in Part I.

C Survey of Part II coursework activities involving a programming language

Course/project	language	open-ended	data structures	from scratch	teamwork
Lego	Matlab	Y	N	N	Y
IA Michaelmas	C++	N	Y	Y	N
IA Lent	C++	Y	Y	N	Y
Mars Lander	C++	Y	Y	N	N
IB Matlab	Matlab	N	N	Y	N
1B C++	C++	N	N	Y	N
IDP	C++	Y	Y	N	Y
3A5	Fortran	N	N	N	N
3F5	C++	N	N	N	N
3F6	Java	N	Y	N	N
3G2	Matlab/Python	N	Y	N	N
3G4	C++	N	Y	N	N
SA1	Matlab	Y	Y	N	Y
GA1	Matlab	Y	Y	N	Y
SC3	Matlab	Y	Y	N	Y
GF1	Matlab	Y	Y	N	Y
SF1	Matlab	Y	Y	N	Y
GF2	C++	Y	Y	N	Y
SF2	Matlab	Y	Y	N	Y
4A2	Fortran	N	N	N	N
4C7	Matlab	N	N	N	N
4F13	Matlab	N	N	?	N
4G3	Matlab	N	N	?	N
4G5	Matlab	N	N	Y	N

Table 1: Overview of the current use of programming in Part II coursework activities. Adapted from `http://www2.eng.cam.ac.uk/~tpl/cuedcomputing/`.