A Proposal for Informatics 1B

Paul Anderson <dcspaul@ed.ac.uk>
Volker Seeker <volker.seeker@ed.ac.uk>
School of Informatics
University of Edinburgh

1 Background

INF1B is a new 20 point course intended to replace the existing Inf1-OP[1] and Inf1-DA[2] courses (10 points each).

The main motivation is to provide more time and effort to help students to acquire a reasonable level of practical programming ability:

- The material from the existing Inf1-DA course will largely be deferred to later years.
- An introduction to practical imperative programming will be included at the end of INF1A for those students with no prior experience. This will allow INF1B to assume some initial (imperative) programming knowledge from all students.

The previous version (1) of this discussion document presented some initial thoughts on a possible approach to the course. This version is an attempt to provide more detail, and to address some of the issues raised by the feedback to the first draft (see [appendix B]). Some details remain to be decided, but we believe that this represents a workable proposal.

2 Motivation

Informally we would like to:

- Focus on practical programming skills and increase the students' ability and confidence in producing realistic, practical applications.
- Foster an awareness of good engineering practice in an informal way - including, version control, testing, and documentation.
- Provide a course which is stimulating and challenging for those with more experience, at the same time as being accessible to those with less.
- Encourage student interaction and collaborative development of larger applications, and show how an object-oriented approach can facilitate this.
- Provide the practical motivation for a more formal approach to the topics studied in later years.

[1] https://www.inf.ed.ac.uk/teaching/courses/inf1/op/

[2] https://blog.inf.ed.ac.uk/da18/

Note that student engagement has historically been a problem in Inf1-OP. This proposal aims to incorporate several best practices for improving engagement and encouraging under-represented students[3], in particular: collaborative learning and student interaction, opportunities for self-reflection, and offering a range of options.

3 Learning Outcomes

We propose the following outcomes:

- Implement components of an object-oriented program, given a specification, and demonstrate the use of an object-oriented approach to enable group development of larger applications.
- Justify implementation decisions, compare implementations, and comment on their strengths and weaknesses.
- Demonstrate an awareness of good software engineering practice, including the use of version control, testing and readable code.
- Locate and use additional sources of information (to include discussion with peers where appropriate) to facilitate independent problem-solving, and reflect on one's own and others' contribution to a collaborative learning environment.
- Plan and organize time, working consistently to a goal.

Note that we do not expect the students to make significant decisions about the overall application design at this stage.

4 Proposed Format

We propose taking an "Objects First"[4] approach to the course material by starting with object-oriented concepts and gradually introducing the features of the Java language in this context. We would like to take advantage of the students' previous experience of functional programming by comparing and contrasting the two approaches.

In practical terms, we would provide a realistic framework of pre-written objects which the students could

[3] https://www.engage-csedu.org/, https://www.ncwit.org/resources/top-10-ways-engage-underrepresented-students-computing/top-10-ways-engage-underrepresented

[4] Objects First with Java: A Practical Introduction using BlueJ, David J. Barnes & Michael Kölling. http://www.bluej.org/objects-first/.

initially connect and modify in small ways, and later augment by writing their own classes from scratch. This framework would provide the background to the entire course.

We intend to choose a suitably flexible application that the same framework could be reused in subsequent years by adding new classes to provide additional functionality. It is also possible to envisage the classes being chosen to highlight various areas of the curriculum - for example databases, graphics, algorithms, natural language, etc.

Appendix A shows a sketch of one possible idea for such a framework.

4.1 Groups

We propose asking the students to work in small groups (probably of three or four), with each student in the group responsible for a different class (or a small set of classes). The classes could then be combined to produce a group implementation of a realistic application. Each group would have the same set of classes, and the interface specifications would be given in advance. This would allow implementations to be compared and exchanged between groups.

The group setting is intended to keep student engagement high, and to help them recognise the value in working together. We would encourage this by requiring the use of a shared version control system, for example. However, we would not expect them all to be sufficiently mature at this stage to negotiate responsibilities and manage interactions. Students in the group would therefore be allocated their own components which they would be expected to develop independently. Substitute components could be "imported" from another group to accommodate students who failed to produce a working implementation.

4.2 Assessment

We do not believe that the practical programming objectives of this course can be meaningfully assessed by examination, and we propose that the assessment be based entirely on the coursework. We anticipate something similar to the following:

- A series of small weekly formative exercises, similar to the current Inf1-OP lab exercises.
- One exercise each week to be submitted for summative assessment (10 in total, using CodeRunner or similar). Passing the course to require a minimum number of these to be submitted (70%). Examples of students' submitted code will also be used for formative feedback on style and efficiency during tutorials (§4.3).
- A first, formative assignment requiring the submission of code for a component of the framework, and a group demonstration of the running code to a tutor.
- A second, summative assignment also requiring the submission of code for a component of the framework, and a group demonstration of the running code to a tutor.
- A final 2-3 page document produced by each individual student which describes their implementation decisions and reflectively compares their code with solutions from other groups, and their experience of working within the group.

To pass the course, students would be expected to (a) submit the minimum number of weekly exercises, (b) produce a minimally working implementation for the second assignment which is reasonably well-structured and readable, and (c) to provide a meaningful written discussion of their implementation, and experiences.

Marking: We anticipate using a comparatively coarse-grained, criteria-based marking scheme similar to that used for undergraduate projects. The criteria would include:

- Completion of weekly exercises - assessed automatically.
- Sophistication of the assignment implementation - assessed from the demonstration.
- Readability and structure of the code - assessed from a brief scan of the submitted code.
- Awareness of alternative approaches - assessed from the final document.
- Reflection on group work - assessed from the final document.

Both the varied levels of possible implementations, and the final discussion document provide scope for exceptional students to be awarded high marks in accordance with the common marking scheme. However, it should be possible for students with little previous experience to meet the minimum requirements.

Academic Misconduct: We are aware that an assessment based entirely on coursework is more susceptible to academic misconduct. There is a real concern that some students will struggle or fail to engage, and may be tempted to submit work which is not entirely their own. In order to mitigate this, we will:

- Run MOSS similarity detection software[5] all on submitted code.
- Require students to explain their code and answer questions during the demonstration.

[5] https://theory.stanford.edu/~aiken/moss/

Note however, that the ability to locate and evaluate relevant resources (including libraries, discussions of bugs, and appropriate code samples, etc.) is an essential skill in developing any real software. So is the ability to discuss approaches and potential solutions with others. These are essential attributes which we would like to encourage - they only become "plagiarism" and "collusion" respectively when they are done in such a way as to misrepresent the origin of the work.

Resits: The groupwork components of the assessment would be difficult to replicate exactly for a resit, and may require some alternative exercise such as a written discussion of a hypothetical situation. The comparison of alternative approaches may also need to be based on different components.

4.3 Activities

The most effective combination of activities is not yet clear. Traditional lectures would be probably be used to present material, possibly in combination with some "live programming" and/or "flipped" sessions. Tutorials and/or lab sessions would be used to discuss student work, provide advice on the assignments and give formative feedback. Students would be expected to submit the (required) exercises before the tutorials, so that sample solutions could be discussed.

5 Resources

We do not expect the total resource requirements for the 20 point course to be more than twice those for the existing 10 point course. The exact details depends on the final choice of activities. However:

- 400 students implies a minimum of 26 tutorials (15 students).
- Around 130 groups of 3-4 students would be required for the demonstrations. We anticipate these normally being handled by the tutors, which would require an additional 3 hours per group - this includes about 20 mins per demo, 4-5 groups, and some time to provide written feedback.
- The marking is estimated to require about 200 hours to scan the code and read the final reports - allowing about 20-30 minutes per student, and additional time for moderation and training.
- Additional TA and lecturer resource will be required for development.

We are slightly concerned about the ability to find the necessary number of qualified TAs/demonstrators/tutors/markers, and the training necessary to maintain a consistent standard.

6 Issues Raised

Appendix B shows the detailed feedback received on the previous draft of this paper. In general, there was broad support for the proposed approach [1,3,7,28,30,34]. This section discusses some of the issues raised:

6.1 Learning Outcomes

There appears to be general agreement on the content and learning outcomes, although there was some discussion of whether we should expect *all* students to be able to program. The level of programming required at this stage is extremely basic, and we believe that it is reasonable to expect students with a degree from Informatics to have achieved this minimum level.

There is also broad support for including wider topics such as version control and documentation [33]. However the students will not be expected to design their own class structures, and we do not think that formal tools such as UML [5] would be appropriate at this stage.

6.2 Groupwork

The potential for groupwork to enable the creation of larger and more interesting applications was recognised [4]. There was some concern that the groupwork would be challenging to organise [16], that first-year students may not be mature enough to manage this type of work [43], and that there would be problems if some group members disengaged [19,22]. We have attempted to clarify the type of group work which we had envisaged (§4.1), and we believe that this addresses these concerns.

There are still some open questions about how the groups are selected [26] and supported [16], but we do not believe that these are particularly problematic - except perhaps for resourcing (§6.3). Similarly, there were some concerns about the ability to design a framework with sufficiently clear interfaces [18], but we are reasonably confident that this is not a significant problem, and appendix A shows a possible example.

6.3 Assessment

The proposed assessment appears to be the most contentious aspect of the proposal. This received some positive [2,31] and some negative [8,37] feedback. The alternative proposals mostly involve some (often large) proportion of exam-based assessment [15,24], but we strongly believe that the learning outcomes of the course cannot be meaningfully assessed in this way. Having a non-assessed course has been suggested as an alternative [14] - this is a possibility (if the regulations

permit), although we would be concerned that students may prioritise their assessed courses over this.

Academic misconduct: The possible opportunities for academic misconduct are clearly a concern [13]. This includes both *plagiarism* and *collusion* [10], as well as "commissioning" work from elsewhere [11]. There is a genuine concern that students will be able to pass the course without having acquired the programming skills which are necessary for the following years, and which such a result might be expected to imply [12]. It has also been noted that assessed coursework can inhibit exactly the kind of discussion and collaboration which we are trying to foster [9,39].

A traditional exam-based assessment is the most robust way of avoiding such academic misconduct, but we do not believe that this is appropriate for the skills that we would like the students to learn. We believe that the proposed approach is a reasonable compromise, and we have attempted to mitigate the possibilities for misconduct as far as possible (§4.2).

Automarking: Several comments [6,29] suggested that there may be more scope for automarking than admitted in the proposal. While this is undoubtedly useful, especially as an initial guide for a marker, we do not think that this is a panacea: it does not address the non-functional aspects of the code (structure and readability, for example); it is difficult to construct and maintain for non-trivial applications (graphical interfaces, for example), and it does not confirm the student's understanding of the material - for example code which performs a linear search on a HashMap to locate the value corresponding to a given key![6].

Marking: There appears to be some support for the coarse-grained criteria-based marking scheme [36], and no other alternatives have been proposed.

Resits: It seems likely that it will be difficult to replicate the main assessment exactly for resits [25,41]. It will require some thought to design an acceptable alternative.

Resources: Running such a large course, and developing new and innovative course materials from scratch is challenging. Resourcing is major concern - including development and maintenance of the framework [21], and marking [38].

One suggestion is to allow students to gain an exemption from the course by passing a test at the end of the first semester [42]. This would reduce the course numbers, and potentially provide a pool of student mentors/tutors. However, we are a little sceptical that it is possible to effectively assess the necessary attributes in this way, and that there would be a sufficient number of students who would not benefit at all from the course.

We would expect the course to make heavy use of student demonstrators in the same way as existing courses, and this could be extended to other years [35]. However, the availability of qualified TAs and markers is a problem for existing courses, and we would like to explore ways of making more less-casual resources available.

Range of experience: The range of experience of the students on this course is extremely wide, and this is something which has presented a challenge in the past [17]. The proposed framework is intended to be flexible enough to support a wide range of tasks, from simple components for beginners, to more interesting and challenging components for those with more programming experience [32]. We hope that this will motivate students at all levels by allowing them contribute according to their experience.

Further investigation: It has been suggested that we might undertake some further investigation into similar courses elsewhere [44] to see how the proposed approach compares. this would clearly be interesting, but is not feasible within the timescales if the proposal is to be implemented for next year. However, there is some evidence that the more collaborative approach that we are proposing is effective in engaging students[7].

[6] A real example from IJP.

[7] https://www.engage-csedu.org/, https://www.ncwit.org/resources/top-10-ways-engage-underrepresented-students-computing/top-10-ways-engage-underrepresented

Appendix A - An Example

The following example shows three individual components which might work together as part of a framework but do not strictly rely on each other[8]. Each module has one simple and one or more advanced implementations. The students must complete the simple implementation at the very least, in groups of three (each student in the group implements a different component). If they are able, and have enough time, they move on to the advanced implementations of their modules.

In practice, we would expect to offer an application with considerably more challenging options than this simple illustration[9].

Cat and Mouse

A playing field is provided where actors can be moved with keyboard input or via AI behaviour models. Three modules are present, a cat, multiple mice and a dog.

1st Module - the Cat:

- Implement the cat's movement with arrow keys (simple).
- Implement eating mice and getting score (simple).
- Implement linear or exponential acceleration for the cat's movement (advanced).
- Implement an AI which automatically catches mice (advanced).

2nd Module - the Mice:

- Implement random mouse movement (simple).
- When the cat gets into the vicinity of the mouse, it turns around and runs in the opposite direction (bit faster maybe, only for 2 seconds maybe, not faster than the cat's max speed) (advanced).
- Add random spawning of new mice with different movement patterns (advanced).

3rd Module - the Dog:

- Implement the dog's basic movement (bouncing off walls and constant speed) (simple).
- When touching the cat, the game is lost (simple).
- Similar to the mice when in the vicinity of the cat, the dog follows the cat until it is too far away again (bit faster, for a few seconds) (advanced).

Going Beyond: Some suggestions in case students finish all of this and want to do even more:

- Add special collectables such as milk which gives the cat a speed or score boost for 10 seconds.
- Change the graphics of an actor by implementing animated sprites or particle effects.

[8] This may be implemented using a system system such as Greenfoot (https://www.greenfoot.org/)

[9] This IJP exercise shows a more complex example: https://groups.inf.ed.ac.uk/ijp/2018/public/assignment1/assignment.html

Appendix B - Feedback

Steve Renals

[1➤] I think that this is a very good proposal, and I very much like the proposed format. [2➤] I am very comfortable with the proposed assessment, and believe it is a very good way to measure the learning outcomes achieved on a course that focuses on 'practical programming skills'. In my opinion, this is a more reliable way to assess the learning outcomes compared to an exam-based approach

Bob Fisher

Some thoughts:

1. [3➤] I'm quite happy with the proposal in general.
2. [4➤] Will the groupwork be staged, so that they can build objects that contribute to larger achievement? Eg. build some sort of application by starting with base objects and work up to integrating these into something bigger? It could be for a mobile phone app. Having something integrated may be more motivating that lots of classes related somewhat arbitrarily to different subject areas.
3. [5➤] Maybe a lecture on UML early to help them visualise data relationships and how data is structured?
4. [6➤] Given the class and interface methodology, there might be an element of automated testing like Dubach has done for one of his classes. This is only part of the assessment of the submissions, with other factors like you outlined.

Perdita Stevens

[7➤] I support the idea of expanding the time available for students to learn Java, including an increased focus on code quality and design issues. [8➤] However, the proposed assessment mechanism is unacceptable. Assessing this course purely on the basis of coursework would be worse than worthless:

1. [9➤] Basing the course around the same development which is assessed for credit will inhibit conscientious students from freely discussing their work with one another (even in ways that the staff would actually be happy with), thereby damaging a major learning modality.
2. It will be utterly trivial for any student who wishes to obtain a higher mark than their competence merits to obtain one (by means ranging from [10➤] being over-supported by well-intentioned members of the same team, who naturally want good overall behaviour of their application, all the way to [11➤] commissioning work from any of the many commercial providers);
3. regardless of the extent to which this actually happens, it will be obvious to everybody involved that we cannot prevent it;
4. [12➤] thereby rendering the marks of students who don't cheat, as much as of those who do, unreliable;
5. and undermining a major aim of the curriculum reform, that is, the aim of ensuring that all students entering the later years of our degree have strong programming skills.

[13➤] *There is a very serious risk of bringing the School and the University into justified disrepute*. This is not a theoretical possibility. Older members of the school will remember the Great CS1 Scandal. For those who do not, it should suffice to observe that even now, almost 20 years later, if you google "university edinburgh plagiarism scandal" you will find press articles about it on your first page of hits. We could expect the next scandal to be considerably worse, because the scale is now larger, because students today are more worried about their precise marks than students were two decades ago, and because the marketplace for buying coursework has burgeoned.

If this course went ahead in this form I, for one, would be very unwilling to have any involvement with its teaching or assessment: I would do so only under protest.

Assessing a course taught at this scale, using resources we can muster, is difficult and there is no perfect answer. I think either of these two approaches would be strictly better than the proposal:

1. [14➤] Do not assess the course. Let it be pass/fail based on attendance.
2. Assess the course by programming exam(s), even if the exam(s) cannot assess everything that we aim for students to learn during the course. (Bear in mind the possibility of offering several programming exams, either as a ladder where passing a basic one is required to enter a more advanced one, or in the manner of tiered GCSEs, by the way.)

[15➤] I do not think coursework should contribute to a student's mark in more than a trivial way (10% say).

[16➤] Turning to the teaching aspects: there are naturally few details here. Assuming that the element of summative assessment is removed from the groupwork, I think it will be very challenging, but perhaps not impossible, to run the course this way. Perhaps one might use the teaching studios for large meetings of

many teams together, one per table, which would enable them to have timetabled work together supported by fewer tutors than one per group. With groups of 4, though, capacity will be an issue, and I would not suggest groups larger than 4.

Two things that might be one another's solutions:

1. [17➤] Even with the addition of the slow start at the end of Inf1A, there will still be a very wide range of programming ability in the class. How is this to be handled?
2. [18➤] The idea that you can have sufficiently precise interface specifications to allow implementations to be swapped between groups, while still allowing flexibility of design, needs to be demonstrated.

An interesting approach might be to have a first phase with very precise interfaces, then actually do a bunch of swapping between groups, encouraging students to evaluate and assess one another's code, and using the results of that to re-form different groups, with some groups comprising students who struggled at the early phases, and others comprising students who produced perfect implementations. The former groups could then be supported into completing a relatively easy application, while the latter were given their head and encouraged to invent variants (perhaps with a culminating competition).

Further points for consideration:

1. [19➤] What will happen when one or more members of a group do not engage?
2. [20➤] Will outside students still be welcome on the course, or will it be restricted to Informatics students only?
3. [21➤] The development of a suitable framework, together with appropriate interface specifications and tests, will be a lot of work – far more work than we will be able to throw away and re-do every year, so the balance of extension/redevelopment/reuse will be tricky. This is yet another reason for separating it from assessment.

Sharon Goldwater

1. I'm not sure how clear it was to Perdita that early phases of the course would involve formative feedback on parts of the design and implementation.
2. Although Perdita's position is far more extreme than mine, [22➤] there is a real concern that some students will struggle or fail to engage regardless of our best efforts, and [23➤] will end up getting other people (whether teammates or paid) to do their work for them. I think we absolutely want to encourage teammates to help each other, but it also needs to be clear that the point of this is for individuals to learn, not to cover for them.
3. [24➤] Are we required to have the final mark be X% of exam and Y% coursework? Could we instead adopt something more like the Medicine model, where students must pass all components? And also make the marks very coarse-grained, eg A/B/C/D/F only. There could separately be prizes for good group projects, but these would not contribute to the mark. So, perhaps there could be a coding exam (maybe less time-pressured than now; ie shorter?), but in addition or instead, have an exam where students are given implementations of the components they would have seen in the coursework, and asked to identify errors or comment on style/design. So the exam is inextricably linked to the actual coursework they should have done.

Under this model I am wondering about using something like the following four learning outcomes.

1. Implement components of an OO program, given a specification.
2. Compare different designs and implementations of OO programs, and comment on their strengths and weaknesses.
3. Coordinate with other team members, contribute to a group project, and provide constructive feedback on their own and others' contributions.
4. Plan and organize time, work consistently to a goal.

#3 [25➤] would be the trickiest to deal with at resit, though arguably one could have an exam that asks about hypothetical situations and what you'd say; or perhaps the constructive feedback could consist of commenting on strengths and weaknesses of different designs (which is much easier to assess by exam).

#4 could be passed by doing some minor continuous assessment and/or group planning exercises. I do think there should be some kind of continuous assessment or early class test to check basic programming skills; because if students do not get these early, they will not be able to contribute effectively to the group project.

[26➤] Unlike Perdita, I do not think it's necessarily a good strategy to put together groups of weak students and groups of strong students. I would rather see students encouraged to recognize the different starting points of members of their group, and help all members progress from there. (Although, the latter is probably more difficult to achieve and might require more

resource.)

[27➤] One reason I'm still a bit unsure about the group project idea is because I haven't explicitly seen that done anywhere, in contrast to things like pair programming. And if everyone else is doing individual assignments, then it will be a lot easier to find good ones to re-use. (Perhaps Volker could even test one or two of them out in OP this year?)

Don Sannella

[28➤] I support the proposal. [29➤] The only thing that I somewhat disagree with is the statement that there are unlikely to be significant opportunities for automarking, just because it must be possible to at least use unit tests to check correct functionality.

Iain Murray

[30➤] I'd be happy to see a move towards students writing and submitting more substantial and interesting programs. [31➤] Making one of their first year courses coursework-only seems like an assessment/learning trade-off worth making.

Several of my tutees have been unhappy completing first year not confident with their ability to write real programs, and I think this move will help. [32➤] Hopefully the framework can be designed so that the students who already have extensive programming experience can be given challenging things to do.

Ian Stark

Thanks for sending this around, and sorry for not replying sooner; I realise this means my response may not be included in your summary at the meeting. Here are the topics of my comments on the Inf1B proposal as tabled:

1. Version control, testing and documentation
2. Small-group work and ideas on teaching support
3. Course-grained grading
4. Code quality and other measures of higher-level performance
5. Some concern about ensuring passing-grade performance
6. Serious concern about assessment overload of students and staff
7. Suggestion of permitting some students to test-out on request

I've expanded on these below, and would be happy to meet to talk about any of them at any stage in the course development:

1. [33➤] It's good to see mention of version control, testing and documentation. I think it would be excellent if these were named in the Learning Outcomes for Inf1B – at the moment those for Inf1-OP include testing but not version control or documentation.
2. [34➤] The small-group development framework in "Proposed Format" seems appealing.
   [35➤] With around 100 such groups I expect there will be a substantial overhead in supporting and monitoring progress. My experience on Inf1-DA has been that a large number of the higher-performing honours students are keen to help instruct new students. They have also been good at doing so provided they have precise clear guidance as well as support for them in turn. I think some who are on their second or third time around doing this tutoring would also be able to coordinate groups of other tutors.
3. [36➤] I think a coarse-grained grading scheme is a good idea. I would recommend looking at CMS5 for a university-approved mapping of letter-only grading into marks. CMS5 is actually for ECA, and has other aspects that may not be appropriate, but it does at least set a precedent in displacing the pervasive assumption of linearly-additive percentages everywhere. I think having Inf1B return purely letter grades would be a great step forward.
4. I think it's an excellent idea to include areas like documentation, code quality and advanced functionality in the assessment criteria, so that higher grades can reflect broader excellence than simply working programs.
5. [37➤] I am concerned that the mechanism proposed will not reliably confirm an individual student has demonstrated the basic requirement of being able to write code that runs.
   In the current learning outcomes this is "Given a detailed design, develop a working program that implements the design". Prior to the deployment of assessment that directly checked this, it was routine for staff to report students in second-year right up to final-year projects who could not demonstrate this. This was, of course, to the huge disadvantage of these students. It's also been visible in the current Inf1-OP exams that the many students passing in August do so by writing code that runs where they could not do that in May; some also report the dramatic prompt from their experience of the mock exam during semester, when they recognise what's required.
   I think it's essential that any Inf1B course assessment includes for each student some identified occasion on which they write a program that runs correctly. For many this may be entirely straight-

forward and done in passing; for a few it will be a key measure of what they need to achieve before moving on to richer course content.

6. [38➤] I'm extremely concerned about assessment overload for both students and staff in the mechanism proposed. [39➤] As written, it seems to say that all of the students' coursework developed as a group through semester will be assessed for the final grade. If true, then this would seriously damage its role in helping students learn – limiting collaboration, disadvantaging those who take longer to learn through semester, and poisoning all group work with worry about grade impact. The proposal states a desire to avoid "assessment which would compromise the learning opportunities" — yet sets out a scheme that does exactly this.
   Conflation of teaching with assessment also makes for poor assessment: as in the rather different domain of machine learning, using the training set as the test set is a terribly unreliable way to measure performance.
   Is there any possibility of instead using some specially-prepared coursework for the assessment, distinct from that used to help the student learn the material? For example, running sessions all through Week 11 distinct from the teaching through semester? [40➤] Where students have had to take exams on separate days to avoid clashes, we have in recent years simply accepted signed statements that they will not discuss content with others until the exam is over. We could easily do that across a single week, and thereby use the same material in 8-10 sessions of 40-50 students at a time rather than try to coordinate one or two giant exams. (This has also been recommended by the external examiner for the current Inf1-OP exam sittings, where it's not clear there is any value in not reusing the same paper.)
   If this assessed coursework week built directly on students' experience through semester then it could make it easier to come up with tasks, rather than the current challenge of setting a problem domain that can be approached from scratch within a single exam sitting. In addition, this would mean only one week of material needs reworking in successive years, rather than the whole programme. [41➤] Finally, it's at least conceivable to run such a session in the resit diet, which would not be possible for assessment using a full 11 weeks of coursework.
7. [42➤] Could we consider permitting students to test out of Inf1B, by taking an assessment at the end of Semester 1? This would allow them to take other outside courses instead, or even to help with the course itself, and increase the resource available to teach the students for whom Java is new and the need is greatest.
   We already have a mechanism to award students credit in "Recognition of Prior Learning", which I guess this might engage. However, even if the students don't get transcript credits, I imagine we might have those who would rather test out and take something else instead.

Other Feedback

The following points were raised at the teaching committee meeting[10] on 10th October:

1. [43➤] Stephen Gilmore questioned whether first year students were mature enough to manage group work, and pointed out the issues with SDP, including the possibility of some students completing workfiuss on behalf of others.
2. [44➤] Alan Smaill suggested that we investigate the content and organisation of similar courses at other Universities.

[10] http://web.inf.ed.ac.uk/infweb/admin/committees/teaching-committee/meetings-directory/tc-minutes-10th-october-2018