What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?

Brett A. Becker
University College Dublin
Belfield, Dublin 4, Ireland
brett.becker@ucd.ie

Thomas Fitzpatrick
University College Dublin
Belfield, Dublin 4, Ireland
thomas.fitzpatrick.1@ucdconnect.ie

ABSTRACT

A well-received ITiCSE 2016 paper challenged the orthodox view that programming is hard to learn. It contended that CS1 educators' expectations are too high, which can result in poor teaching and learning, and could impact negatively on diversity and equity. The author posed a challenge to the community to collect research-based evidence of what novice programmers can achieve, and use evidence to derive realistic expectations for achievement.

We argue that before rising to this challenge we must determine: *What exactly do educators expect of introductory programming students?* This paper presents our efforts toward answering this question. We manually curated hundreds of CS1 syllabi, providing a fresh perspective of expectation in CS1 courses. We analyzed learning outcomes and their concepts, in addition to languages utilized and other useful CS1 design and delivery information.

This work contributes to a current picture of what is expected of introductory programming students, and provides an interactive online tool linked to all collected syllabi and containing all learning outcomes and other associated information. We hope this will aid the community in deciding whether or not we have unrealistic expectations of our CS1 students and if so, our contributions provide a starting point for the community to adjust them.

CCS CONCEPTS

• **Social and professional topics → Computer science education**; **CS1**.

KEYWORDS

CS1; CS1 languages; curriculum design; introductory programming; language choice; learning objectives; learning outcomes; novice programmers; syllabi; syllabus; teaching languages

ACM Reference Format:

Brett A. Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287485

1 INTRODUCTION

A central focus of computing education research is the teaching and learning of a first programming language [21]. Despite this, a well-cited ITiCSE working group found that "Three decades of active research on the teaching of introductory programming has had limited effect on classroom practice" [19, p. 204]. It is also frequently stated that programming is hard to learn [1, 21], a claim that regularly goes back to the McCracken working group that found "many students do not know how to program at the conclusion of their introductory courses" [15, p. 125]. The reasons for this are believed to be multifaceted, and that difficulties in learning to program contribute to low motivation and high dropout rates [2].

At ITiCSE 2016, Andrew Luxton-Reilly presented a paper titled 'Learning to Program is Easy' [12]. The paper was well-received, earning a commendation from the program committee, and quickly generating discussion in the community [8]. Luxton-Reilly's paper challenged the orthodox view that programming is hard to learn, and claimed that this view results in uncritical teaching practices, poor student learning, and may also impact negatively on diversity and equity in computing. The paper also presented substantial evidence that this view has deeply permeated the computing education community's culture, literature and psyche. Luxton-Reilly claimed that computing educators make introductory courses difficult by establishing unrealistic expectations for novice programmers. He proposed that by revisiting the expected norms for introductory programming we may be able to substantially improve the learning of novice programmers, address negative impressions of disciplinary practices, and create a more equitable environment.

Specifically, Luxton-Reilly showed that the literature on failure rates and novice programmer ability represents a substantial body of evidence supporting the claim that students are not able to program at the level expected by instructors. He argued that it is not clear that this means that programming is inherently difficult. Instead he proposed that the computing education community should view our learning outcomes for CS1 courses as being unrealistic. He concluded with a challenge to the community: *Collect research-based evidence of what novice programmers can achieve in CS1, and use evidence to derive realistic expectations for achievement.*

Intrigued by this challenge, we realized that addressing it would most likely be complex and time consuming. It would likely require at least one large-scale, multi-institutional, multi-national study, involving significant resources and time-frames. We argue that before rising to this challenge, the community must be able to answer the question: *What exactly do we expect our introductory programming students to achieve?* This paper presents our efforts toward answering this question. We view this as a first step towards meeting Luxton-Reilly's challenge.

1.1 Overview and Motivating Questions

We manually curated CS1 syllabi by searching the websites of all 916 institutions making up the 2016-2017 QS World University Rankings® [1]. We analyzed these syllabi, seeking to answer our questions below. We also provide a web front-end to our database including learning outcome information and links to syllabi, and offer it to the computing education community as a tool, as described in Section 5. To move toward answering our overarching question *What exactly do we expect our introductory programming students to achieve?* we formulated four questions specific to CS1 learning outcomes, as these are central to Luxton-Reilly's challenge (Q1-4). To dive deeper into current CS1 teaching practices we formulated two additional questions on languages and syllabi (Q5-6):

Q1: What percentage of CS1 courses have explicit learning outcomes?

Q2: What concepts do explicit CS1 learning outcomes cover?

Q3: How do These Concepts Align With CS2013?

Q4: What do explicit CS1 learning outcomes look like?

Q5: What is the current CS1 teaching language distribution?

Q6: What are the most common computing terms found in CS1 syllabi?

In forming these questions, we aimed to provide information from three vantage-points:

(1) That of other educators, by presenting information such as their explicit CS1 learning outcomes *word for word*, for further analysis/use by the community.
(2) That of our tool, by presenting *derived* information such as a list of concepts included in learning outcomes.
(3) A somewhat agnostic view, by including information such as full-syllabus word frequency counts.

Our contributions are the beginnings of a current, multi-faceted, picture of what is expected of CS1 students, and an online tool where educators can utilize and contribute to our dataset. It is our intention that these results and the tool that we provide will better position the community to decide if we, as a community, have unrealistic expectations of our CS1 students, as proposed by Luxton-Reilly, and to move towards addressing his challenge.

Before continuing, we comment on terminology. We use the term *learning outcome* to mean an explicit statement expressing something that students are expected to know, or be able to do, upon successful completion of a course. These are sometimes called "learning objectives". We do not include as learning outcomes any consequences of learning such as student grades. It is important to recognize that it is through learning outcomes that we *tell* our students what we expect of them. This is important, as what we tell students can be adjusted, as Luxton-Reilly noted.

This paper is organized as follows. In Section 2 we review related work. In Section 3 we present our methodology, followed by our results and discussions in Section 4. In Section 5 we provide an overview of the tool that presents our data. We discuss threats to validity in Section 6 before concluding in Section 7.

[1] www.topuniversities.com/university-rankings/world-university-rankings/2016

2 RELATED WORK

What is taught in CS1 has been the subject of numerous studies over several decades [3, 13]. Although many computing degrees have similar structures, and CS1 often fills a common place in them, there are many approaches. Reflecting this, CS2013 [11] discusses various trade-offs between design decisions in first year courses, but is generally non-prescriptive with respect to first year curricula [12]. Additionally, most CS1 research does not focus on explicit learning outcomes, instead focusing on course content, material, assessment, and other practical aspects of delivery. Thus, most of these studies do not directly investigate what is expected of CS1 students.

In their 2001 ITiCSE working group, McCracken et al. [15] (mentioned in Section 1) developed a set of learning outcomes that described the expected level of skill for students at the end of their first year of study. Students were expected to be able to complete the following steps: (1) Abstract the problem from its description; (2) Generate sub-problems; (3) Transform sub-problems into sub-solutions; (4) Re-compose the sub-solutions into a working program; and (5) Evaluate and iterate. These were formulated in response to the question "What should be assessed [in CS1]" [15, p. 126]. The authors state that this is an ideal and generalized situation, and do note some problems with their abstraction.

In 2006, Schulte and Bennedsen sought to create a general, worldwide picture of teachers' opinions about what should be taught in introductory programming courses [22]. They specifically studied what educators believe to be important to teach, what they actually teach, and what they believe students find most difficult. They culled 28 topics from the literature and used these in a survey where instructors were asked to rate the difficulty and relevance of each.

Goldman et al. [6] constructed a list of 32 CS1 topics in 2010, using a structured multi-step (Delphi) process that uses a group of experts to achieve a consensus opinion. Specifically, they sought a set of key topics for which if a student fails to demonstrate a conceptual understanding of, then confidence could be given to the conclusion that the student had not mastered the course content.

In 2011, Petersen et al. [20] noted that final exams are a useful proxy for deriving curricular expectations and determining what instructors understand to be important. They presented the analysis of nine experienced CS1 instructors who reviewed final examinations from a variety of North American institutions. Their instrument drew concepts from [22] and [6] discussed above. Their final instrument contained 28 CS1 topics.

Around the same time, Elliott Tew and Guzdial were working towards developing a validated assessment of CS1 topics known as the FCS1 (Foundational CS1) assessment instrument [26]. In [25] they sought to identify concepts that a wide variety of introductory courses and approaches had in common. They chose textbooks as the external artifact representing the content of a course, because they deemed other measures such as course syllabi or assignments as not feasible to analyze at a large scale. They conducted a document analysis of the table of contents of the two most widely adopted CS1 textbooks from each of the six major publishers of computing textbooks (a total of 12 books). Using a bottom-up approach, they aggregated topics listed in the tables of contents, noting which concepts were covered by which texts. However their

list became unwieldy with over 400 concepts, ranging from low-level topics such as byte code and computer architecture details to advanced topics traditionally covered later in the curriculum. They used the framework of the Computer Science volume of Computing Curricula 2001 [17] to revise their initial list of topics, resulting in 188 topics. They further refined this list by analyzing the content of canonical texts representing each of the common introductory approaches (objects-first, functional-first, and imperative-first). A concept was included in this step of revision if it was covered by all texts, or excluded by any one of the canonical texts. This resulted in a final list of 29 fundamental computing concepts common across languages and pedagogical approaches. As this list was only one step toward developing an assessment instrument, they further refined it to arrive at a list of 10 constructs amenable to testing.

The studies mentioned above involve investigating what is taught in CS1, commonly involving constructing CS1 concept lists, and often with an extra dimension such as the difficulty students have with the concepts. Also related to the present work at a different angle is the CITIDEL (Computing and Information Technology Interactive Digital Educational Library) Syllabus Collection [27], now a part of Ensemble [10]. This work aimed to provide a syllabus information tool to the computer science education community, which is also an aim of the present work.

The CITIDEL Syllabus Collection currently contains 5,083 computing syllabi, organized into 14 subjects. The subject 'Programming Fundamentals' has 406 syllabi from 67 institutions. An inspection of filenames revealed syllabi dating as far back as 1994, with possibly older syllabi also included. We decided that we wanted to compile a current collection representing more institutions, resulting in our present approach. Nonetheless we completed a preliminary analysis of the CITIDEL syllabi and plan on using this for a historic comparison with the collection we curated.

3 METHODOLOGY

Our methodology was shaped by several factors. First, we aim to provide a foundation for the community to answer the question *What exactly do we expect our introductory programming students to achieve?* We view this as a first step toward enabling the community to move towards addressing Luxton-Reilly's challenge of collecting research-based evidence of what novice programmers can achieve in CS1, and using evidence to derive realistic expectations for achievement. We felt that the core of this foundation should be *explicit* which led us to the decision to examine learning outcomes from as many up-to-date CS1 syllabi as possible. In this way we are getting to the root of what we as a community are *expecting* of our CS1 students. In addition to the analysis presented in this paper, we make these learning outcomes available word for word, in one place, as described in Section 5.

Second, as reviewed earlier, many efforts to identify CS1 concepts focus on what concepts are most important, or most difficult. This is problematic, as noted by [6, p. 5:9]: "Finding consensus on the most important and difficult concepts for a CS1 course is inherently challenging given the diversity of approaches in languages ..., pedagogical paradigms ..., and programming environments used. These factors influence the perceptions of the importance and difficulty of the topics." Thus we did not want to focus on 'important' or 'difficult' learning outcomes, but a large and representative sample of all types of CS1 learning outcomes.

Third, we noted the difficulty identified by Elliott Tew and Guzdial in analyzing course syllabi on a large scale. Facing this difficulty, we decided to curate the learning outcomes of CS1 courses on a meaningful scale, by hand. We view learning outcomes as a fundamental starting point to answer our overarching question. We believe that although it may be possible to attempt to answer this question by analyzing proxy instruments such as assessments, these would need to validly assess the learning outcomes, which is not a guarantee. Additionally, computing education lags other discipline-based education research in the number and range of validated assessments available to the research community [18]. Further, it has been suggested that the assessments we currently use to evaluate learning are simply too difficult in the first place [12], which brings into question their suitability for purposes such as ours.

3.1 Syllabus and Learning Outcome Curation

We began by randomizing the 916 institutions making up the QS World University Rankings® 2016-2017. We then procedurally searched for CS1 syllabi by following the below steps:

(1) Navigate to institution website
(2) Navigate to Computer Science / Information Technology / Computer Engineering faculty or program page
(3) Search module lists or lists of major requirements for likely CS1 courses - allowing for multiple courses per institution
(4) If at this point a syllabus was not found, we often had a course name (but it did not link to a syllabus). We then performed a Google search for <course title + university name + "syllabus">.

We populated a database with the details of these syllabi including: URL, learning outcomes, if the learning outcomes were explicitly stated, programming language, prerequisites, and other details. If the learning outcomes were explicitly stated, we saved these (verbatim) for later analysis. By explicitly stated, we mean if the syllabus had a set of statements of student expectations, often under a heading of 'learning outcomes', 'learning objectives' or similar, often appearing before any other material such as 'course content', etc. These frequently begin with wording such as 'At the end of this course the student will be able to...' or similar.

We used Guo's methodology [7] and Hertz [9] as guides in identifying CS1 courses. Where a course slightly deviated from these requirements (for instance having prerequisites), or if it was possibly a CS2 course) we tagged it so that in the future these courses could be removed from analysis if desired.

3.2 Learning Outcome Concepts

We manually analyzed the learning outcome statements to derive a list of concepts they cover. For instance if a learning outcome mentioned recursion, we added recursion to our list of learning outcome concepts. If we later found another course that mentioned recursion in a learning outcome, we marked that syllabus as such, and updated our total 'recursion count'. For syllabi that did not specifically include learning outcomes, we used the 'course content' or similar sections to continue populating our concept list. We also marked that syllabus entry with the tag '!Explicit' indicating that

there were no explicit learning outcomes, and that we 'scraped' concepts from other syllabus content. This allows these syllabi to be excluded from future analysis when required, for instance, allowing researchers and educators to use our tool to only analyze those syllabi that contained explicit learning outcomes.

We decided to generate a concept list instead of using an existing one from the literature as we wanted the resulting list to represent the concepts explicitly involved in what educators expect of CS1 students, without bias inherent in lists derived from assessments, or those that focus on 'important'/'difficult' concepts, etc.

4 RESULTS AND DISCUSSION

From the 916 institutional websites we searched, we found 234 CS1 syllabi from 207 institutions. Table 1 shows the 30 countries represented in this collection and the number of syllabi, institutions, and the percent of all syllabi.

Table 1: Number of syllabi (#S) and institutions (#I) per country, and percent of total syllabi (%T), $n = 234$.

Country	#S	#I	%T	Country	#S	#I	%T
USA	118	114	50	India	4	4	2
England	33	31	14	South Africa	3	3	1
Australia	15	15	6	Turkey	3	3	1
Scotland	9	8	4	Portugal	2	1	1
Ireland	8	5	3	Sweden	2	2	1
Canada	7	7	3	Netherlands	2	2	1
New Zealand	6	5	3	Lebanon	2	2	1
Wales	5	5	2				
China, Czech Republic, Denmark, Egypt, Ghana, Hungary, Jordan, Kenya, Pakistan, Philippines, Qatar, Singapore, South Korea, Switzerland and United Arab Emirates have 1 syllabus each (<1%).							

We note that this seems biased towards Anglophone countries. We did expect some bias, particularly as we can only process English-language material. However the bias seems to not be coming so much from the list we searched, but possibly from what we are looking for. In total our list had universities from 81 countries, 72 of them not English speaking, representing 65 percent of institutions. However Table 1 shows that most syllabi found are from Anglophone countries. Nonetheless, we note that almost all websites we visited were in English, even those of institutions in non-Anglophone countries. Additionally we only saw a handful of what we believe might be syllabi in languages other than English. It is also possible that other educational systems are not strongly based on learning outcomes as countries like the USA, and EU countries. As our tool is available for researchers to contribute to (see Section 5), we would welcome syllabi from a more diverse set of countries.

4.1 Q1-4: Explicit Learning Outcomes in CS1

4.1.1 Q1: What percentage of CS1 courses have explicit learning outcomes? 154 (65.8%) of the 234 CS1 syllabi we curated have explicitly stated learning outcome or learning objective statements. We believe this is important, as explicit learning outcomes provide a direct mechanism to gauge the expectations we have for students, and are therefore central to Luxton-Reilly's challenge.

4.1.2 Q2: What concepts do explicit CS1 learning outcomes cover? Table 2 shows the learning outcome concepts we identified along with the percentage of: explicit learning outcomes containing these concepts (column Explicit); syllabi without explicit learning outcomes that contain these concepts elsewhere in the syllabus (column !Explicit); and percentage of syllabi containing each concept either in explicit learning outcomes or otherwise (column All).

Table 2: Most frequent syllabus concepts. 'Explicit' is the percentage of CS1 courses with explicit learning outcomes addressing the given concept. '!Explicit' is the percentage of CS1 courses without explicit learning outcomes where the given concept was featured somewhere else in the syllabus. 'All' is the percentage of all CS1 courses where the given concept appeared either in an explicit learning outcome, or elsewhere. $n_{Explicit} = 154, n_{!Explicit} = 80, n_{All} = 234$. Concepts are in descending order of column 'Explicit'.

Concept	Percentage of Courses		
	Explicit	!Explicit	All
Testing & debugging	56	28	45
Writing programs	55	30	46
Selection statements (if/else,etc.)	46	43	44
Problem solving (including computational thinking terms)	45	47	45
Arrays, lists, vectors, etc.	41	37	39
Basic OOP	40	32	36
Variables, assignment, arithmetic operators, declarations, data types	40	35	38
Functions, methods, procedures	38	25	33
Repetition & loops	37	29	34
Designing algorithms	29	37	31
Classes & objects	25	18	22
File handling & I/O	23	28	24
Documentation	21	11	17
Recursion	20	16	18
Data structures (general or specific - e.g. stacks)	19	30	23
The following concepts occur in < 19% of Explicit LOs, < 16% of !Explicit, and < 16% of All (presented in decreasing order of column *Explicit LOs*): Program design methodology & style, Abstraction, How computers work & history of computing, Inheritance, IDE use, Strings, Searching algorithms, program comprehension, Sorting algorithms, Exception handling, "Fundamentals of programming", Evaluating time & space complexity, Basic graphics & GUIs, Teamwork skills & communication; Encapsulation; Polymorphism; Pointers; Abstract classes & interfaces; Scope; Memory allocation; Tracing program execution; Detecting syntax errors; Detecting semantic errors; Information representation; Command prompt use; UML; Code manipulation; Functional programming; Web development; Pseudocode; Induction; Security; IT & Data; Scientific skills; Version control; Boolean logic; Multithreading & concurrency			

Many interesting observations and discussions could come from Table 2, and a full discussion is limited here by space and participants. This is one of the reasons that we decided to make our data available, so that others can perform their own analyses. First, it should be noted that Table 2 is not *the* answer to Q2 – it is one of several possible, many of which can be found with the tool we

have made available. Second, Table 2 provides evidence that CS1 syllabi that include explicit learning outcomes may be quite different to those that don't. This is an interesting topic for further study. Finally, we note the high rank of 'testing & debugging' (which we decided to group together) which occurs more frequently than 'writing programs'. We are confident that when 'debugging' and 'testing' are considered separately, the most frequent concept would be quite fittingly, 'writing programs'. This is something that a user of our website could determine readily.

4.1.3 Q3: How do These Concepts Align With CS2013? A full comparison of the results of this study and CS2013 [11] is a direction for future work, but we wondered how many of the concepts in Table 2 are covered by the *Software Development Fundamentals* (SDF) Knowledge Area (KA) in CS2013 [11], and specifically the Knowledge Unit (KU) *Fundamental Programming Concepts* (FPC) in that KA. Eight of the top 15 concepts presented in Table 2 are covered by the FPC KU, and 13 are covered when including the other three KUs within the SDF KA (with some overlap). The only two concepts not covered by SDF are *Basic OOP* and *Classes & objects*, which are covered by the KU *Object-Oriented Programming* under the KA *Programming Languages*. This is not surprising, given the fact that FPC "identifies only those concepts that are common to all programming paradigms" [11, p. 167].

4.1.4 Q4: What do Current CS1 Learning Outcomes Look Like? Of the 154 CS1 syllabi with explicit learning outcomes, we found a total of 1,029 learning outcome statements, an average of 6.68 learning outcomes per syllabus. All of these learning outcomes are available, word for word, on the website we describe in Section 5.

4.2 Q5: Current CS1 Language Distribution

Our fifth question was: *What is the current CS1 teaching language distribution?* Table 3 shows the languages we encountered as sole teaching languages (152 syllabi). To compare our list to [23] (a large 2012 survey of teaching languages) we omitted the 65 syllabi with no language specified (as [23] utilized more information than just syllabi, at times directly contacting instructors, they recorded a much lower number of no-language courses). To keep the comparison simple we also omitted the 17 syllabi reporting multiple languages. By this count, Java is by a good margin the most frequent CS1 language, used in 74 (49%) of the 152 syllabi. The second most frequent is Python at 36 (24%). C++ comes next at 30 (20%) followed by C at 8 (5%). Neither column adds to 100% as we do not report several languages with < 5% share. These results broadly align with those of Simon et al. [24] who conducted parallel surveys of introductory programming courses in Australasia [14] and the UK [16] (both in 2016), with a view to examining the programming languages being used.

These studies, compared to those we discuss now, provide evidence that the distribution of the top CS1 languages hasn't changed too dramatically in 20 years. In 2007, Pears et al. [19] noted that Java, C, and C++ 'topped the list'. This was also the case for surveys conducted in 1998, 2001, 2003, and 2005 (see [19]). Also in 2007, Giangrande corroborated the dominance of Java, C, and C++, notably not mentioning Python at all [5].

The most obvious shift has been the rise of Python, evidenced by comparison with Siegfried et al. [23], whose 2012 list of CS1 languages comprised of data from 404 US institutions (356 reporting single languages). Unsurprisingly, the biggest changes are Java (down from 55 to 49%) and Python (up from 12 to 24%). This shows that the rise of Python has likely come at the expense of Java, and to a lesser degree C++. Nonetheless, Python's rise was not as dramatic as anticipated given that in 2014 Guo reported that Python had overtaken Java as the teaching language of choice at the top CS schools in the United States [7]. This could be explained to some extent by the fact that Guo only examined the top 39 PhD-granting institutions, and considered CS0 courses as well as CS1.

For a comprehensive review of publications on teaching languages, the reader is guided to [13], a 2018 review of the introductory programming literature that processed more than 5,000 papers.

Table 3: Current CS1 teaching language distribution. $n = 152$. The final column is data from 2012 [23], $n = 356$.

Language	This study (2016-17 data)		From [23] (2012)
	Number of Courses	%	%
Java	74	**49**	55
Python	36	**24**	12
C++	30	**20**	23
C	8	**5**	5

Haskell, JavaScript and R appeared ≤ 1% in both studies. We do not report languages appearing in [23] but not in our data such as Alice.

4.3 Q6: CS1 Syllabus Terms

Our final question was: *What are the most common computing terms in CS1 syllabi?* We analyzed the frequency of all words in each of the HTML syllabi ($n = 185$). We are currently working on PDF and other file types ($n = 49$). Table 4 shows the 14 most frequent computing terms overall. For the sake of presentation, and as an example of the kind of analysis our tool enables, we excluded what we deemed to be non-computing terms.

Table 4: Most frequent computing terms in 185 CS1 syllabi.

Term	Count	Term	Count
Programming	1,384	Class/es	275
Design	627	Assignment	227
Data	520	Object/s	207
Algorithm/ic/s	410	Web	196
Test/ing/s	356	Control	181
Method/s	330	Array/s	175
Function/s	317	Security	175

We intended this data to be agnostic, but for a meaningful analysis we realized that certain terms were problematic and needed some human intervention. This started with removing non-computing words. Other more specific words were also problematic. For instance, the third most frequent term in Table 4 is *Data*, but many of these occurrences probably came from the two-word term *Data Type/s*. Similarly, the ninth most frequent word is *Assignment* –

some of these probably come from the term *Assignment operator*, but many could refer to assessment tasks. On the other hand we excluded the plural word *Assignment**s*** which came up 407 times, because this term quite likely refers to assessment tasks almost exclusively. Nonetheless, these word frequency lists could be regenerated by others with a different methodology if desired as the sources are all available on our website. This, for example, would allow a researcher to quickly locate the syllabi that contain a specific word or combination of words, allowing them to query what languages these courses use, what their learning outcomes are, etc.

5 A TOOL FOR THE COMMUNITY

We have made available a tool at csed.ucd.ie/sigcse2019/ containing all of the data used to generate our results. It allows all 234 syllabi we curated to be searched and sorted by language, location, if the learning outcomes are explicit, learning outcome concepts (including their counts from matching linked syllabi), and more. Researchers and educators are encouraged to add their own syllabi to the collection through a submissions page. Future enhancements will be geared towards making it more useful for the computing education community, and feedback is welcomed.

As a very brief example of the kind of analysis that can be done with this tool, Table 5 shows the countries (with 4 or more syllabi) with the highest percentages of syllabi containing explicit learning outcomes. The fact that European countries rank so highly is most likely due to the fact that explicit learning outcomes are an integral part of the Bologna Process [4].

Table 5: Countries with 4 or more syllabi with the highest percentages of explicit learning outcomes (%ELOs), $n = 234$.

Country	%ELOs	Country	%ELOs	Country	%ELOs
Ireland	100	Scotland	89	Canada	71
New Zealand	100	England	82	USA	58
Wales	100	Australia	73	India	25

6 THREATS TO VALIDITY

Our results have some threats to their validity. First, all of our syllabi are from institutions on the 2016/2017 QS World University Rankings®. Using rankings for similar purposes has been done by other studies (e.g. [7]), but any biases in the rankings may bias the results. We know that the ranking we used biases results towards research universities. We also noted a likely language bias discussed in Section 4. We chose to use this ranking as it included nearly 1,000 institutions which we felt fit the time and resources we had available. Future work directions include adding institutions from other rankings and sources. We also hope that members of the community avail of the opportunity to add syllabi to the collection.

We could have missed some courses. We aimed for a processing time of 5 minutes per website, amounting to one researcher working full-time for over three weeks on data collection only. We also could have incorrectly included non-CS1 courses although we found Guo's methodology [7] and Hertz's description [9] helpful.

The learning outcome concepts presented in Section 4.1 are affected by our concept of 'explicit', and our methodology in developing our concept list. However, as all of our data is available, it would be easy for these results to be replicated or the query modified to suit individual requirements. Our results on language frequency in Section 4.2 are similarly affected by the sample of syllabi. Additionally, the syllabus word frequency results are, as discussed in section 4.3, subject to some judgment calls—for example in dealing with singular and plural versions of the same word. However as above, these results could be easily reproduced, or the queries modified. Finally, it is also important to recognize that as we are looking at learning outcomes, we are analyzing what we *tell* our students we expect of them. This is important, as what we tell students can be adjusted, but also, what we tell students of our expectations may not correlate perfectly with what we actually expect.

Additionally, we acknowledge that our analysis of the curated learning outcomes focus on concepts, and not the depth to which these concepts are covered. We note however that an analysis of the depth to which these concepts are covered is possible as we provide the full wording of all learning outcomes on our website, in addition to links to the original syllabi. Finally, we do not address the possibility of reinforcement bias among syllabi. For example, it is probable that many syllabi are designed either by consulting model curricula or are inspired from other, more established syllabi at other universities. Such a possibility aligns with Luxton-Reilly's claim that certain views have deeply permeated the computing education community's culture, literature and psyche.

7 CONCLUSIONS AND FUTURE WORK

We presented a tool that allows researchers to access hundreds of current CS1 syllabi and associated data. We also present initial insights based on these syllabi, focusing on CS1 learning outcomes, languages, and syllabus keywords. These results paint a picture of what we expect CS1 students to achieve. We view this as a first step in allowing the community to answer the question: *What exactly do we expect our introductory programming students to achieve?* Answering this question, in turn, will put the community in a better position to rise to the challenge proposed by Luxton-Reilly in 2016 [12]: *Collect research-based evidence of what novice programmers can achieve in CS1, and use evidence to derive realistic expectations for achievement.*

Future work involves adding more syllabi (particularly from more diverse countries) to the collection, and conducting further analysis on the data the community now has at its fingertips. As a public tool, this work could be useful for those who are conducting CS1 research, and for educators who are designing or modifying courses.

Finally, we note that there is limited evidence on what we expect of our students on a large scale. This work demonstrates that gaining a representative picture of what we expect of our students fraught with biases and details that make gathering such evidence difficult. Nonetheless we think that the information we provide may be useful to the community.

8 ACKNOWLEDGEMENTS

The authors would like to thank the SIGCSE Board for supporting this work with a SIGCSE Special Project Grant (December, 2016).

REFERENCES

[1] Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective Compiler Error Message Enhancement for Novice Programming Students. *Computer Science Education* 26, 2-3 (2016), 148–175. https://doi.org/10.1080/08993408.2016.1225464

[2] Brett A. Becker, Cormac Murray, Tianyi Tao, Changheng Song, Robert McCartney, and Kate Sanders. 2018. Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 634–639. https://doi.org/10.1145/3159450.3159453

[3] Brett A. Becker and Keith Quille. 2019. 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA. https://doi.org/10.1145/3287324.3287432 *in press*.

[4] Ursula Fuller, Arnold Pears, June Amillo, Chris Avram, and Linda Mannila. 2006. A Computing Perspective on the Bologna Process. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '06)*. ACM, New York, NY, USA, 115–131. https://doi.org/10.1145/1189215.1189181

[5] Ernie Giangrande, Jr. 2007. CS1 Programming Language Options. *J. Comput. Sci. Coll.* 22, 3 (Jan. 2007), 153–160. http://dl.acm.org/citation.cfm?id=1181849.1181881

[6] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2010. Setting the Scope of Concept Inventories for Introductory Computing Subjects. *Trans. Comput. Educ.* 10, 2, Article 5 (June 2010), 29 pages. https://doi.org/10.1145/1789934.1789935

[7] Philip Guo. 2014. Python is Now the Most Popular Introductory Teaching Language at Top US Universities. *Communications of the ACM Blog (BLOG@CACM), July* (2014).

[8] Mark Guzdial. September 30, 2016 (accessed August 18, 2017). *No, Really - Programming is Hard and CS Flipped Classrooms are Complicated: ITICSE 2016 Award-winning Papers.* https://computinged.wordpress.com/2016/09/30/no-really-programming-is-hard-and-cs-flipped-classrooms-are-complicated-iticse-2016-award-winning-papers/

[9] Matthew Hertz. 2010. What Do "CS1" and "CS2" Mean?: Investigating Differences in the Early Courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*. ACM, New York, NY, USA, 199–203. https://doi.org/10.1145/1734263.1734335

[10] Gregory W. Hislop, Lillian Cassel, Lois Delcambre, Edward Fox, Rick Furuta, and Peter Brusilovsky. 2009. Ensemble: Creating a National Digital Library for Computing Education. In *Proceedings of the 10th ACM Conference on SIG-information Technology Education (SIGITE '09)*. ACM, New York, NY, USA, 200–200. https://doi.org/10.1145/1631728.1631783

[11] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* Technical Report. New York, NY, USA. 999133.

[12] Andrew Luxton-Reilly. 2016. Learning to Program is Easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 284–289. https://doi.org/10.1145/2899415.2899432

[13] Andrew Luxton-Reilly, Simon, Ibrihim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings of the 2018 ITiCSE Conference on Working Group Reports (ITiCSE-WGR '18)*. ACM, New York, NY, USA. https://doi.org/10.1145/3293881.3295779 *in press*.

[14] Raina Mason and Simon. 2017. Introductory Programming Courses in Australasia in 2016. In *Proceedings of the Nineteenth Australasian Computing Education Conference (ACE '17)*. ACM, New York, NY, USA, 81–89. https://doi.org/10.1145/3013499.3013512

[15] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. *SIGCSE Bull.* 33, 4 (Dec. 2001), 125–180. https://doi.org/10.1145/572139.572181

[16] Ellen Murphy, Tom Crick, and James H Davenport. 2017. An Analysis of Introductory Programming Courses at UK Universities. *The Art, Science, and Engineering of Programming* 1, 2 (2017).

[17] The IEEE CS / ACM Joint Task Force on Computing Curricula. 2001. Computing Curricula 2001, Computer Science. (2001). https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2001.pdf

[18] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. 2016. Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, New York, NY, USA, 93–101. https://doi.org/10.1145/2960310.2960316

[19] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A Survey of Literature on the Teaching of Introductory Programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '07)*. ACM, New York, NY, USA, 204–223. https://doi.org/10.1145/1345443.1345441

[20] Andrew Petersen, Michelle Craig, and Daniel Zingaro. 2011. Reviewing CS1 Exam Question Content. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 631–636. https://doi.org/10.1145/1953163.1953340

[21] Anthony Robins. 2010. Learning Edge Momentum: A New Account of Outcomes in CS1. *Computer Science Education* 20, 1 (2010), 37–71. https://doi.org/10.1080/08993401003612167

[22] Carsten Schulte and Jens Bennedsen. 2006. What Do Teachers Teach in Introductory Programming?. In *Proceedings of the Second International Workshop on Computing Education Research (ICER '06)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/1151588.1151593

[23] Robert Michael Siegfried, Daniel Greco, Nicholas Miceli, and Jason Siegfried. 2012. Whatever happened to Richard Reid's list of first programming languages? *Information Systems Education Journal* 10, 4 (2012), 24.

[24] Simon, Raina Mason, Tom Crick, James H. Davenport, and Ellen Murphy. 2018. Language Choice in Introductory Programming Courses at Australasian and UK Universities. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 852–857. https://doi.org/10.1145/3159450.3159547

[25] Allison Elliott Tew and Mark Guzdial. 2010. Developing a Validated Assessment of Fundamental CS1 Concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*. ACM, New York, NY, USA, 97–101. https://doi.org/10.1145/1734263.1734297

[26] Allison Elliott Tew and Mark Guzdial. 2011. The FCS1: A Language Independent Assessment of CS1 Knowledge. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 111–116. https://doi.org/10.1145/1953163.1953200

[27] Manas Tungare, Xiaoyan Yu, William Cameron, GuoFang Teng, Manuel A. Pérez-Quiñones, Lillian Cassel, Weiguo Fan, and Edward A. Fox. 2007. Towards a Syllabus Repository for Computer Science Courses. *SIGCSE Bull.* 39, 1 (March 2007), 55–59. https://doi.org/10.1145/1227504.1227331