Duran, Rodrigo; Rybicki, Jan Mikael; Sorva, Juha; Hellas, Arto

Exploring the value of student self-evaluation in introductory programming

Published in:
ICER 2019 - Proceedings of the 2019 ACM Conference on International Computing Education Research

DOI:
10.1145/3291279.3339407

Published: 30/07/2019

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Duran, R., Rybicki, J. M., Sorva, J., & Hellas, A. (2019). Exploring the value of student self-evaluation in introductory programming. In *ICER 2019 - Proceedings of the 2019 ACM Conference on International Computing Education Research* (pp. 121-130). ACM. https://doi.org/10.1145/3291279.3339407

Exploring the Value of Student Self-Evaluation in Introductory Programming

Rodrigo Duran
Aalto University
Finland
rodrigo.duran@aalto.fi

Jan-Mikael Rybicki
Aalto University
Finland
jrybicki@cc.hut.fi

Juha Sorva
Aalto University
Finland
juha.sorva@aalto.fi

Arto Hellas
University of Helsinki
Finland
arto.hellas@helsinki.fi

ABSTRACT

Programming teachers have a strong need for easy-to-use instruments that provide reliable and pedagogically useful insights into student learning. Currently, no validated tools exist for rapidly assessing student understanding of basic programming knowledge. Concept inventories and the SCS1 questionnaire can offer great benefits; this article explores the additional value that may be gained from relatively simple self-evaluation metrics. We apply a lightweight self-evaluation instrument (SEI) in an introductory programming course and compare the results to existing performance measures, such as examination grades and the SCS1. We find that the SEI has a similar correlation with a program-writing examination as the SCS1 does, although both instruments correlate only moderately with the examination and each other. Furthermore, students are much more likely to voluntarily answer the lightweight SEI than SCS1. Overall, our results suggest that both the SEI and other instruments need to be greatly improved and outline future work towards that end.

CCS CONCEPTS

• **Social and professional topics → Computer science education**; *Model curricula*; Student assessment.

KEYWORDS

Self-evaluation, assessment, CS1, introductory programming

ACM Reference Format:
Rodrigo Duran, Jan-Mikael Rybicki, Juha Sorva, and Arto Hellas. 2019. Exploring the Value of Student Self-Evaluation in Introductory Programming. In *International Computing Education Research Conference (ICER '19), August 12–14, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3291279.3339407

1 INTRODUCTION

Teachers assess students for many reasons. For example, they want to provide feedback, evaluate their own teaching, attend to a class's prior knowledge in course design [18, 42, 46, 52, 56, 57], recommend suitable learning activities or modules to individual students [7], help students assess themselves and self-regulate their learning [60], identify at-risk students [54], and compare students to cross-institutional benchmarks [11, 38].

These ambitions create a need for a range of assessments. Sometimes, a detailed assessment is necessary; at other times, a numerical overall grade will do. It is sometimes fine for students to self-assess their abilities; at other times, an external evaluator or formal test is required. While certain assessments must be mandatory, others should remain voluntary. In some cases, it is acceptable that assessment takes time; in others, the assessment must be quick even at the expense of accuracy.

In some fields of education, validated assessment tools are available for teachers to use, such as the Common European Framework of Reference for Languages (CEFR) [11], while computing education still lacks such tools. Nevertheless, the assessment of computing skills has attracted much interest. One strand of prior research aims to build concept inventories and similar tests of students' programming ability and knowledge. The *SCS1* questionnaire [35], for instance, uses a set of 27 multiple-choice questions about pseudo-code programs to measure students' knowledge of imperative programming. Such instruments, which directly target student knowledge, are potentially very valuable. However, taking such a test is time-consuming and can be stressful for students; moreover, heavyweight tests are poorly suited for repeated assessment within a single study module.

Other researchers have indirectly assessed students' computing skills through the lens of *self-efficacy*: the students' belief in their ability to perform certain tasks. Danielsiek et al. [12] validated an instrument for measuring self-efficacy in algorithms courses. Self-efficacy has been shown to correlate strongly with successful learning [4, 24, 40] and can be assessed rapidly. However, it is not a direct measure of students' conceptual understanding.

To measure students' prior knowledge, skills, and conceptual understanding, students could self-assess their own skills. *Self-assessment* is the evaluation or judgment of "the worth" of one's performance and the identification of strengths and weaknesses with a goal of improving one's learning outcomes [38]. In short, self-evaluation is particularly suited for assessing past or current work, and self-efficacy attempts to estimate the confidence and extrapolate future performance.

Ross [38] reviews evidence from previous research suggesting that self-assessment contributes to higher student achievement and improved behavior. Furthermore, there is evidence that supports the reliability of self-assessment in terms of internal consistency, consistency across tasks and items, and over short periods of time. However, age, experience, the expectancy of grading, and prior achievement impact on how students self-evaluate. Thus,

ICER '19, August 12–14, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6185-9/19/08…$15.00
https://doi.org/10.1145/3291279.3339407

self-assessments tend to correlate only moderately with teacher assessments. Miller [30] shows that ceiling effects in self-assessment studies are related to overly vague or difficult to understand scoring criteria. Instrument sensitivity can be increased by adding measurement levels and precisely defined criteria.

Self-assessment is also a key element of self-regulated learning [22] and closely related to self-efficacy. In literature, the terms self-assessment and self-evaluation are often used interchangeably; we will use the term *self-evaluation* when discussing our work.

This article continues the work of Duran et al. [13] to develop and validate a *self-evaluation instrument* (SEI). The SEI complements existing assessment tools by rapidly surveying students' understanding of introductory programming concepts.

We aim for the SEI to include the following characteristics: (1) It can be answered quickly. Students volunteer to answer the SEI when prompted and complete it; (2) It captures pedagogically useful information about students' prior programming knowledge and the growth of this knowledge as they learn, reflecting students' progress along a learning trajectory of code-comprehension skills; (3) It highlights differences between programming concepts (rather than being a single generic measure of programming ability or self-confidence); (4) Despite not being a rigorous test of ability, it provides some insight into the actual performance as measured by more heavyweight instruments; (5) It does not "feel like an exam" but promotes reflection and self-evaluation; (6) It is easy for students to understand without prior training: students need not answer questions about highly abstract concepts and generic learning objectives; (7) Instead, it is phrased in terms of learning objectives, programming language, and other contextual factors of a particular CS1 course; (8) Its generic structure can be adapted to different courses; (9) It is easy to administer at scale.

In practice, our SEI is an online survey that asks students to rate their own understanding of selected programming concepts. By administering the SEI repeatedly at different stages of an introductory programming MOOC, we explored the practicality of the SEI. Furthermore, while investigating the SEI, we also administered the SCS1 at the end of the MOOC and checked whether our student cohort is comparable in terms of SCS1 performance to other cohorts reported in the literature. This gave us an opportunity to replicate some of the results from the nascent body of research on the SCS1. Our research questions are as follows:

- **RQ1** *How do students answer the SEI and SCS1: do they volunteer to answer and complete them?*
- **RQ2** *Does the SEI capture CS1 students' growing confidence in their programming knowledge as they learn?*
- **RQ3** *Does the SEI reveal concept-specific differences in students' evaluations of their ability?*
- **RQ4** *Does the SEI have external validity: to what extent do students' self-evaluations agree with existing performance metrics, such as examination grades and the SCS1?*
- **RQ5** *Do our students find the same SCS1 questions difficult as the students in earlier studies did?*

2 RELATED WORK

The computing education research (CER) community has introduced numerous instruments for assessing students' knowledge and skills (e.g. [2, 12, 16, 17, 21, 27, 31, 33, 36, 37, 39, 41, 45, 55]). Margulieux et al. [28] provide a comprehensive list of the validated ones. In this section, we comment on selected studies that either provide useful benchmarks or are similar to the present work by exploring self-evaluation or self-efficacy in a computing context.

Some of the best-known validated tests of programming knowledge include the FCS1 [49] and SCS1 [6, 35]. The FCS1 is a multiple-choice test of concepts commonly covered in CS1 courses: students answer questions about programs written in imperative pseudocode. It was validated and shown to be reliable and positively correlated with course grades. The SCS1 [35] is a more easily available isomorphic version of the FCS1; it has a similar internal consistency and a strong correlation with the FCS1. Since their introduction, the FCS1 and SCS1 have been used in many other studies as performance benchmarks [23, 32, 50, 51, 58].

Danielsiek et al. [12] developed and validated a questionnaire for assessing self-efficacy in an algorithms course. This instrument prompts students to evaluate themselves by rating 21 short statements about their algorithm-related abilities. The authors administered it several times, demonstrating that students' self-efficacy increased as they learned.

Murphy and Tenenberg [31] examined upper-level CS students' ability to self-assess their knowledge of data structures. They compared a single question that asked students to self-assess their performance on a test with the students' actual test score. The results revealed a moderate correlation between students' (n=61) estimates and quiz performance; students with higher scores were more accurate. In another study, Kallia and Sentance [21] used a combination of two self-efficacy measures and a single self-assessment question to investigate relationships between function-related misconceptions and self-efficacy for programming. They found that students who hold misconceptions about programming demonstrate significantly lower levels of self-efficacy than students who have few or no misconceptions and overestimate their performance in the self-assessment.

Ngai et al. [33] sought to alleviate CS1 students' assessment anxiety by teaching them to self-evaluate. The students in the pilot study (n=13) used a given rubric to rate themselves on three programming skills — debugging, coding and programming — and even gave themselves an overall grade. Ngai et al. found that the self-evaluations correlated strongly with instructor-given grades (.78) and the grades from a final examination (.85).

Alaoutinen and Smolander [2] created a survey instrument intended for teachers to assess student knowledge as well as for students to self-evaluate their development. Based on Bloom's Taxonomy [3], the instrument asks students to evaluate their knowledge and skills in the context of introductory programming in the C language. Students were to estimate at which level they could successfully operate, selecting a different Bloom level for each concept. The students' self-evaluations correlated strongly with exam grades (.54), and the topics considered more difficult by teachers were also rated more difficult by the students.

While many studies above overlap with ours in one respect or another, none address the same combination of goals as we do. The study by Alaoutinen and Smolander [2] is arguably the most similar to ours; our work can be seen as an effort to refine theirs. Compared to their study, we aim to (1) phrase our SEI in more concrete terms

(as suggested by the literature on self-evaluation), thus relieving students from the burden to interpret Bloom levels, which is known to be difficult even for teachers [29]; and (2) connect our SEI to a learning trajectory of code-comprehension skills.

The next section summarizes the development and structure of our SEI, as originally presented by Duran et al. [13]. The subsequent sections present our study design and answer the five research questions listed above.

3 DEVELOPMENT OF THE INSTRUMENT

Our prototype SEI (available at https://goo.gl/nGR9Th) is intended for imperative object-oriented programming in Java but can be adapted to other contexts. The SEI is inspired by the self-evaluation rubric of the Common European Framework of Reference (CEFR) for languages [11, p. 26-27] and the theoretical and empirical works in CER that suggest a hierarchy of code-comprehension skills [9, 14, 25, 26, 47, 48, 59]. The framework levels are organized in a hierarchy of skills from comprehending the meaning, syntax, and semantics of a concept (A), to predict the effect of syntax on program behavior, i.e., tracing (B), and the ability to comprehend and summarize patterns (C). We introduce granularity to those levels by adding sub-levels of complexity as the number and quality of interacting elements [5, 14, 44] and students' familiarity with the code. Instructional designers can use as many levels as necessary in their adapted instrument. Each sub-level includes a statement reflecting the conceptual level of code comprehension and its associated skill, including the level of complexity.

As we expected students to have varied backgrounds and levels of expertise, we introduced three sub-levels to the first level (A): an unfamiliarity level (A0), a familiarity with the general meaning of a concept (A1), and comprehension of syntax and semantics of a concept (A2). By mastering level A, students can conceptually comprehend a concept (e.g loop) and recognize its syntax in the code (e.g. the *for* keyword) without being able to predict its functionality when applied to a concrete case.

After mastering level A, students can trace code on level B. Students at level B1 can trace code with few elements, predicting the code output when using concrete values and code with descriptive naming conventions (e.g. the code `"for count in range of 10"` denotes "repeat 10 times"). At level B2, students can achieve the same results as in level B1 but can also deal with several simultaneous interacting elements (e.g. code including many abutted loops with inner conditional structures). Recognizing patterns and summarizing code are the most advanced code comprehension skills. At level C1, students can recognize patterns using a concept even if code-naming conventions are non-standard and comprehend code with a variety of abstract inputs (e.g. recognize lower-level patterns, such as traversing a collection, checking for negative numbers, or adding elements to a list). Finally, at level C2, students should master all other levels and can generalize the purpose of the code, and summarize it in plain English (e.g. summarize code as "a filter function that takes a list as input and outputs all positive numbers").

We evaluated programming concepts typically found in imperative CS1 courses: *variables and assignment (var), input and output (io), expressions and arithmetic operators (exp), conditional statements (sel), loops and iteration (loops), data collections (dc), functions and methods (func) and classes and objects (objs)* [49]. The original instrument and questions were created in English and then translated into Finnish by an expert language instructor and an expert programming instructor.

The CEFR companion volume for new descriptors [34] describes three validation phases for new instrument scales: initial research and development (intuitive phase), improving the categories and quality of descriptors (qualitative phase) and calibrating the best descriptors to a mathematical scale, and confirming the cut-offs between levels (quantitative phase). Our intuitive phase consisted of exploratory work to define the number of levels, sub-levels, and phrasing of statements by the first and second authors. The qualitative phase iteration was conducted by the authors until a saturation point was reached and an overall agreement achieved.

Our prior work [13] explored the metrics of internal consistency in a pilot by analyzing the prototype SEI answers of 2196 students at the beginning of week 1. These results indicated that the instrument is highly reliable (Cronbach's $\alpha = .98$) and could distinguish itself from traditional metrics [15], while strongly correlating with them. In this article, we further explore the quantitative data.

4 STUDY SETTING AND DATA COLLECTION

This study was conducted during a programming online course (available at https://ohjelmointi-19.mooc.fi), offered as a MOOC by the University of Helsinki. The basic course lasts seven weeks (December to February) covering the principles of programming in Java. Each week covers the following concepts:

Week 1: Stdin/Stdio, variables, conditionals, while loops.
Week 2: Logical operators, methods and parameters, call stack.
Week 3: Lists and arrays, loops, strings, testing.
Week 4: Files, file system, classes and objects.
Week 5: Primitive and referential variables, method overloading, comparing objects, lists as objects.
Week 6: Classes and objects, methods, hashmaps.
Week 7: Sorting and searching, unit testing.

The course is taken by both degree and non-affiliated students at the University of Helsinki. When starting the course, students fill in a research consent form and background questionnaire (Duran et al. [13] explores the relationship between the SEI and background questions in more detail). Answering the questionnaire was voluntary but included an incentive raffle of movie tickets. Students could skip any question and choose not to give research consent. The mean age of respondents (n=3801) is 35.26 years, of which 54.35% reported themselves as males, 44.41% as females, 0.63% as not specified and 1.6% did not answer. Students' prior CS experience was heterogeneous (median = 10, mean = 370.17, std.dev = 1897.87 total hours programmed) as was CS prior education (median = 0, mean = 1.68, std.dev = 4.23 courses completed).

The SEI was an optional task. Data was collected using an online form that presents all levels simultaneously, and students could skip any concept. As typical with MOOCs, the dropout rate was high, and the number of respondents decreased over the weeks. The SEI was applied at the beginning of week 1 (n=3807) and week 4 (n=1379) in the course, and at the end of week 7 (n=867).

The SCS1 (27 questions) was translated into Finnish by an expert language instructor and an expert programming instructor and

then presented online as an optional task at the end of week 7 (simultaneously with the SEI). From the 640 students who answered the SCS1, 477 gave permission to conduct research with their data. We only considered test-takers those who answered at least 10 questions [58], yielding a total of 440 respondents. At the end of week 7, students (n=740) took an online exam (3 questions, weighted 20%, 30%, and 50%, respectively) to determine the final course grade. While only a part of the final grading, participating in the exam was voluntary. The course exam included three questions:

- Q1 was a modified version of the Rainfall problem [43] done with String lengths.
- Q2 asked students to create a set of static methods that create and handle lists and maps.
- Q3 asked students to create an object (e.g. a musician) and then another object that contains such objects (e.g. a band). Additionally, students were expected to construct a set of methods ranging from getters to methods that iterate over the nested objects, such as asking the musicians of the band to play and to output the average experience of the musicians.

Due to the voluntary nature of the tasks, there was much variation in the way students completed them. A total of 283 students completed the SCS1 and course exam, while 261 also completed the SEI on week 7. Considering the whole course, 205 students completed all tasks on week 7 and also answered the background questionnaire and the SEI on week 1.

5 RESULTS

RQ1: Student Responses to the SEI and SCS1

To investigate student answers to the prototype SEI and SCS1 (**RQ1**), we compared the completion rates of the SEI in Week 7 with those of the SCS1 and course exam. Of the 867 respondents who attempted the SEI in Week 7:

- 867 (100%) completed the SEI.
- 498 (57.43%) completed the exam.
- 376 (43.36%) answered > 10 questions of SCS1 (test-takers).
- 319 (36.79%) answered at least half of SCS1 questions.
- 84 (9.69%) answered all questions of SCS1.

Of the 740 respondents who attempted the course exam:

- 740 (100%) completed the course exam.
- 498 (67.3%) completed the SEI.
- 254 (34.32%) answered > 10 questions of SCS1 (test-takers).
- 216 (29.19%) answered at least half of SCS1 questions.
- 55 (7.43%) answered all questions of SCS1.

Of the 477 respondents who attempted at least one question of SCS1:

- 422 (88.47%) completed the SEI.
- 283 (59.33%) completed the exam.
- 440 (99.24%) answered > 10 questions of SCS1 (test-takers).
- 365 (76.52%) answered at least half of SCS1 questions.
- 92 (19.29%) answered all questions of SCS1.

Students answered both the SEI and SCS1 within the same survey system and links to the instruments were placed into the material next to each other. While expecting the shorter questionnaire to receive more answers than the lengthier SCS1, we were nevertheless surprised by the large difference in the numbers of respondents. To better understand this phenomenon, we considered the written feedback received by the course staff. There was no student feedback about the SEI. This was in stark contrast to the SCS1, on which the students volunteered to give a noticeable amount of negative feedback. For example, some students complained about the unclear purpose of the test; a mismatch between SCS1 questions and course content urged them to make assumptions about the constructs of the language. Some argued that the SCS1 is, perhaps, testing how to read other people's code, something not practiced in the course. Students also complained about the 1-hour time frame to complete the test, arguing that it requires very quick reasoning and good problem-solving skills.

Students also stated that fill-in-the-blanks exercises were good and tested their logic understanding, but tracing questions felt like "processing speed and memory tests". Questions that checked variable output after method calls were considered difficult since students felt that this behavior is particular to a given programming language and hard to assess in pseudocode. Some even noted that if the course exam difficulty was as high as the SCS1, they could even consider withdrawing from the course. In some cases, students removed their research consent after seeing the SCS1.

RQ2: Students' Growing Confidence

To investigate if students' confidence in their programming ability increased during the course (**RQ 2**), we examined the descriptive statistics of the SEI on weeks 1, 4 and 7 (Table 1). Table 2 shows that the differences between each week's means by concept (δ) are all significant at $p < .001$ using a Mann-Whitney U-Test. The effect size (Cohen's d [10]) of changes from week 1 to week 4 and week 1 to week 7 is large. The changes from week 4 to week 7 were much smaller, except for classes and objects with a large effect size. Figure 1 shows that these findings match the timeline of course contents, particularly the small effect size of Δ in all concepts from week 4 to 7, except for classes and objects, introduced in week 4, which could still be considered as a "new content" for students.

Figure 1: Content timeline and mean levels of SEI concepts.

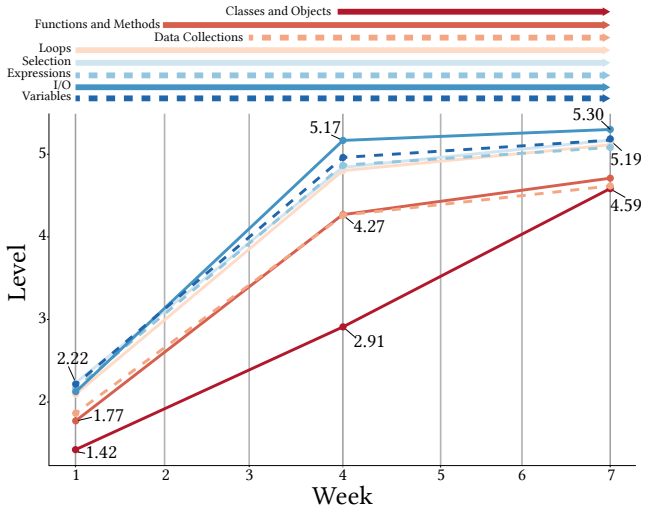


Table 1: Descriptive statistics of SEI concepts.

	Week 1								Week 4								Week 7							
	Var	I/O	Exp	Sel	Loops	DC	Funct	Obj	Var	I/O	Exp	Sel	Loops	DC	Funct	Obj	Var	I/O	Exp	Sel	Loops	DC	Funct	Obj
Mean	2.22	2.13	2.17	2.23	2.10	1.86	1.77	1.42	4.96	5.17	4.86	4.84	4.80	4.26	4.27	2.91	5.19	5.30	5.08	5.17	5.11	4.62	4.71	4.59
Std. Dev.	2.06	2.04	2.02	2.0	1.98	1.85	1.84	1.72	1.14	0.96	1.13	1.09	1.10	1.21	1.32	1.98	1.03	0.91	1.08	0.96	0.99	1.16	1.12	1.18
Median	2	2	2	2	2	1	1	1	5	5	5	5	5	4	4	3	5	6	5	5	5	5	5	5
Instrument Mean	1.99								4.51								4.97							

Table 2: Δ of means by SEI concept and effect size of Δ.

	Week 4 - Week 1		Week 7 - Week 4		Week 7 - Week 1	
	Δ	Cohen's d	Δ	Cohen's d	Δ	Cohen's d
Var.	2.75*	1.47†	.23*	.21‡	2.97*	1.55†
I/O	3.04*	1.67†	.13*	.14★	3.17*	1.69†
Exp.	2.7*	1.48†	.22*	.2★	2.92*	1.55†
Sel.	2.61*	1.45†	.33*	.32‡	2.94*	1.59†
Loops	2.71*	1.51†	.31*	.29‡	3.02*	1.64†
D.C.	2.4*	1.41†	.35*	.3‡	2.75*	1.58†
Func.	2.5*	1.46†	.44*	.35‡	2.94*	1.7†
Obj.	1.49*	.83†	1.68*	.98†	3.16*	1.94†

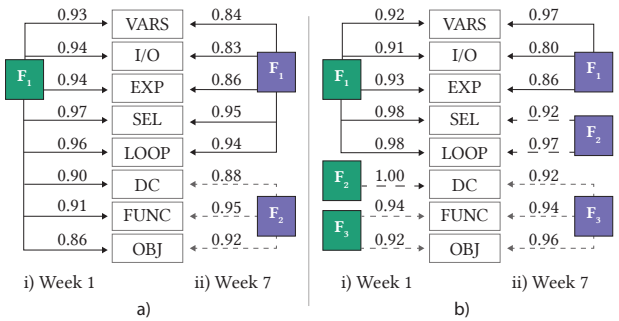
* = $p < .001$. † = large, ‡ = small and ★ = negligible effects.

RQ3: Concept-Specific Differences in Ability

To investigate if the instrument can distinguish between concept-specific differences in students' evaluations of their ability (**RQ3**), we examined if the evaluations could be better understood as single or multiple factor models. We conducted an Exploratory Factor Analysis (EFA) to examine possible factor structures that fit the SEI data. A parallel analysis using a minimum residual method suggested between 1 and 4 factors for week 1, and between 2 and 4 factors for weeks 4 and 7.

We performed a Factor Analysis (FA) to investigate the structure of concepts. Believing that factors were correlated and data distribution was not normal, we used an oblique rotation and Ordinary Least Squared/Minres factoring method. Week 1 analysis determined that only the *one-factor* model exhibited acceptable factor loadings (>.3), as shown in Figure 2(a)(i). Week 4 analysis showed acceptable factor loadings using 2, 3 and 4 factors and week 7 showed acceptable factor loadings using 2 and 3 factors.

Figure 2: Factor loadings on weeks 1 and 7 for overall data (a) and experienced students (n=51) (b).



We performed a Confirmatory Factor Analysis (CFA) fitting the data into the FA suggested models using Maximum Likelihood (ML) as an estimation method. Model parameters suggest that the one-factor model fits week 1 data well [20]. The Comparative Fit Index (CFI) of .92 shows that the one-factor model is a better fit than a more restricted baseline model (values > .9 are usually accepted). The Root Mean Square Error of Approximation (RMSEA), which measures how closely the model reproduces data patterns (penalizing more complex models), has a value of .11, which is very close to the suggested value of acceptance (≤ .1). A Standardized Root Mean Square Residual (SRMR) is an absolute measure of fit, defined as the standardized difference between the observed and predicted correlations. The one-factor model shows a good fit for the data, with an SRMR of .021 (≤ .08 is considered a good fit).

Since the models suggested for weeks 4 and 7 had different numbers of factors and factor structures, they could not be considered nested. To test which model more accurately describes the data, we performed a Vuong test [53] and analyzed estimators of fit when necessary. The test showed that the four-factor model was distinguishable from the other models with less factors and was considered a better fit for week 4 data ($z = -2.01, p = .02$), and model fit indicators can all be considered acceptable: CFI = .99 (>.9), RMSEA = .063 (≤ .1), SRMR = .016 (≤ .08) .*Factor one* concepts (loadings) are: variables (0.85), I/O (.8) and expressions (.88). *Factor two* concepts are selection (.95) and loops (.96). *Factor three* consists of data collections (1), while functions (.93), and classes and objects (.68) loaded into *factor four*.

Vuong tests further showed that three and two-factor models for week 7 could not be considered distinguishable ($w2 = .001, p = .54$), and parameter indicators were very similar for both models. In general, models with smaller Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) can be considered better fits to data. The three-factor model showed smaller (non-significant) values for AIC (-1.46) and BIC (-6.23). Overall, both models can be accepted. Since the only difference between model structures is a factor with a single component (variables) with very high loading (1) in the three-factor model, we suggest a simpler two-factor model fit, as presented in Figure 2(a)(ii).

To further examine how the SEI could reveal concept-specific differences, we investigated if the previously described factors and loadings would remain when focusing on expert students (measured by students' self-reported total hours programmed, i.e. HPT). Since HPT was collected in week 1, we considered in this analysis only students who completed all course tasks (n=205).

For experienced students (upper quartile), the EFA analysis recommended 1, 2 or 3 factors for week 1. The Vuong test showed that the three models are distinguishable, the three-factor model was a better fit ($z = -0.98, p = 0.16$), and its indicators were within the recommended parameters ($SRMR = .03, RMSEA = .16, CFI = .96, TLI = .94, \chi^2 = 40.78, p = 0$). For Week 7, the EFA analysis suggested 2 or 3 factors. The Vuong test showed that the two and three-factor models are not distinguishable, but the indicators of the three-factor model ($SRMR = .03, RMSEA = .08, CFI = .99, TLI =$

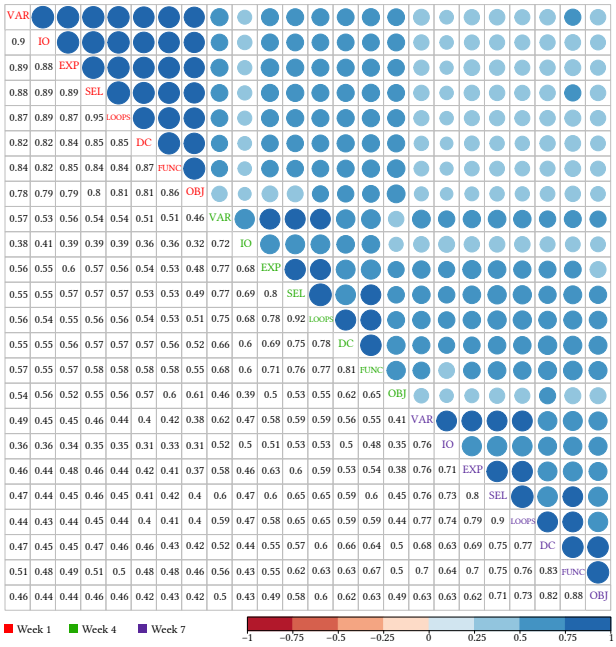
.98, $\chi^2 = 516.26, p = 0, AIC = 374.57, BIC = 426.72$) were a better fit than the two-factor model ($SRMR = .04, RMSEA = .15, CFI = .95, TLI = .93, \chi^2 = 516.26, p = 0, AIC = 388.99, BIC = 437.29$). Figure 2 (b) shows the factor loadings for the best fit models for experienced students on weeks 1 and 7.

RQ 4: External Validity

Many studies have investigated the student performance using the FCS1/SCS1 [23, 32, 50, 51, 58]. One main advantage of using a validated instrument is to produce comparable and generalizable results, providing meaningful information about the investigated context. To study the external validity of the SEI (**RQ4**), we first examine metrics reliability over the weeks and later show how the instrument correlates with the course final exam and SCS1.

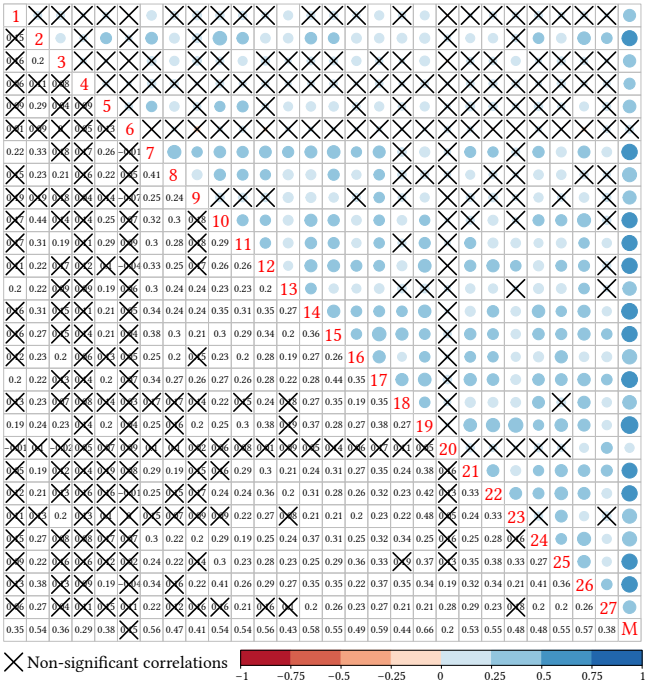
The metrics of internal consistency and construct validity show that the SEI is very reliable. Cronbach's α reliability test is very high: $\alpha = .98$ *on week 1,* $\alpha = .92$ *on week 4 and* $\alpha = .96$ *on week 7.* Removing items from the instrument does not increase its reliability, except for an increase of .03 when removing classes and objects on week 4. The composite reliability of the instrument is also very high: *.98 for week 1, .94 for week 4 and .95 for week 7.* Figure 3 shows Spearman's correlations (corrected with Bonferroni's method for multiple observations) between the concepts during the course.

Figure 3: Correlations of SEI concepts on weeks 1, 4 and 7. Size and color of circles correspond to size of correlation.



The scaled final score (min = 0, max = 100) of the 440 SCS1 test-takers showed a mean of 36.98 (1.06 standard error) points and a standard deviation $\sigma = 22.15$, similar to Parker et al. results (m =35.85, $\sigma = 13.1$) and higher than Timmermann et al. [50] (m = 22). Cronbach's α of our SCS1 data ($\alpha = .87$) was higher than Xie et al. [58]'s ($\alpha = .7$) and Parker et al. [35]'s ($\alpha = .59$) values. Examining α changes when removing items, question 6 was the only one considered problematic with a .01 increase in α (Xie et al. [58] found that items 20, 24 and 27 were problematic). By analyzing the factor loadings, we found that questions 6 (.08) and 20 (.18) had a poor loading while Xie et al. [58] reported questions 20, 24 and 27 with poor loadings. Figure 4 shows the Spearman's correlations (Bonferroni's corrected) of all 27 items of SCS1 and its mean score.

Figure 4: Correlations between SCS1 questions (1-27) and the test mean (M). Several questions showed non-significant correlations and could be considered problematic (e.g. 6, 20).



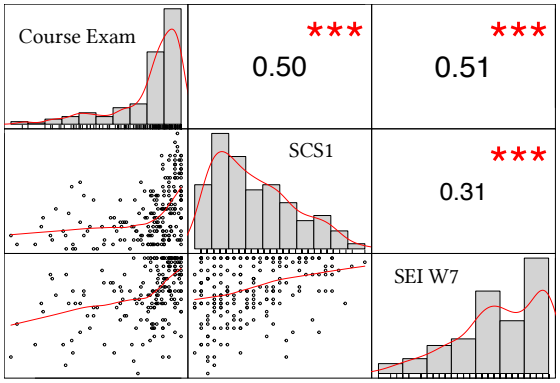
The final course exam test-takers (n=740) achieved a mean of 76.16 (.86 standard error) points (min = 3.33, max = 100) and a standard deviation $\sigma = 23.26$. The Cronbach's α reliability test showed that the course exam was reliable ($\alpha = .79$). A Spearman correlation test was performed to check how each programming question correlated with the final score and each other. Question 1 showed a moderate correlations with questions 2 (.46) and 3 (.47), and question 2 showed a strong correlation (.62) with question 3 (all at p < .001). Questions 1 (.54), 2 (.79) and 3 (.82) were strongly correlated with the course exam grade (p < .001).

Figure 5 presents the Spearman's correlation between the course exam, SCS1 and SEI on week 7 (n=205). The SEI on week 7 and SCS1 had a very similar correlation with course exam grades while being moderately correlated with each other (.31). Whereas Parker et al. [35] does not provide a correlation of SCS1 with course grades, Tew and Guzdial [49] reported the same correlation (.51) between the FSC1 and a final exam.

RQ5: Replication of SCS1 Results

As noted above, we found similar student performance on the SCS1 and similar metrics of the SCS1's internal reliability as earlier studies. We now seek to provide a more nuanced analysis to investigate if our data suggests difficulties in the same questions (**RQ5**) by

Figure 5: Distribution and Spearman's correlations between the SCS1, course exam, and SEI (* p < .001)**



conducting an Item Response Theory (IRT) analysis using previous research [35, 58] as benchmarks for methods and results.

As the test was provided online, one question at a time, we argue that the conditional independence of items is still accepted [58]. To test the unidimensionality of data, one requirement for IRT, we performed a *Confirmatory Factor Analysis* (CFA), fitting the data into a single latent factor model using Weighted Least Squares (WLSMV) as the estimation method [58]. The single latent model indicators (CFI = .931, RMSEA = .039 and SRMR = .048) suggest an acceptable fit, very close to Xie et al. [58]'s values (CFI = .98, RMSEA = .014 and SRMR = .079).

An ANOVA test confirmed that a three-parameter (3PL) model was a better fit than restricted models, with all items fitting our data. The SCS1 can be considered moderately difficult with a mean difficulty δj = .81 (Table 3). Students' ability (θ) ranged from $\theta = [-1.74, 2.76]$ with a mean value of .018. The first quartile students showed an ability estimation $\theta = -0.65$, while fourth quartile students showed an ability estimation $\theta = .66$. All discrimination values (αj) can be considered meaningful (>.4)[58]. Since our analysis used a 3PL (as opposed to a 2PL by Xie et al.) some questions showed a positive value in the third parameter, which can be interpreted as a pseudo-guessing value where students at lower levels of ability no longer show a 0 probability of having a correct answer, but instead a (γj) probability.

Table 3 shows that questions 6 and 20 were the most difficult (δj), but no question was considered too difficult ($\delta j > 3$), as opposed to Xie et al. [58] results (δk) where questions 5, 13, 15 and 18 were regarded as too difficult. Using Parker et al. [35] methodology (% of correct answers), questions 3 (.8), 11 (.51), 12 (.55), 19 (.61), and 23 (.64) could be considered moderately difficult and all other items hard. Parker et al. found that questions 1, 2, 3, 19 and 23 could be considered moderately difficult and all other questions hard.

6 DISCUSSION

Our RQ1 concerned student responses to the SEI and SCS1. We expected that students would more likely volunteer to answer the SEI than take a test, which was clearly the case. Even when compared to the course exam (which was optional but affected grading), the number of SEI answers is almost two times higher than for the

Table 3: Parameters of SCS1 questions using a 3PL model.

j	δj (SE)	δk	αj(SE)	γj	j	δj (SE)	δk	αj (SE)	γj
1	.89 (.23)	1.16	.64 (.13)	0	14	.52 (.12)	1.13	2.02 (.38)	.05
2	.62 (.16)	.89	2.03 (.52)	.15	15	.98 (..09)	3.11	2.81 (.52)	.05
3	-1.7 (.58)	.27	.92 (.18)	.01	16	.86 (.16)	1.28	1.37 (.33)	.03
4	1.76 (.37)	2.26	.87 (.49)	.16	17	.7 (.1)	1.68	2.33 (.44)	.05
5	1.25 (.14)	3.91	2.16 (.58)	.18	18	1.36 (.15)	5.07	1.39 (.2)	0
6	2.93 (1.16)	.99	1.29 (1.17)	.31	19	-0.33 (.08)	.74	2.51 (.33)	0
7	.83 (.11)	2.33	2.1 (.44)	.06	20	2 (.19)	NA	4.81 (4.06)	.09
8	1.02 (.13)	1.6	2.24 (.7)	.19	21	.77 (.12)	1.58	1.66 (.31)	.03
9	.78 (.32)	1.16	1.04 (.34)	.1	22	.45 (.09)	1.36	1.57 (.19)	0
10	.71 (.21)	1.27	3.24 (2.16)	.23	23	-0.55 (.11)	.08	1.28 (.17)	0
11	.41 (.15)	1.9	2.12 (.49)	.2	24	1.31 (.11)	NA	2.2 (.42)	.02
12	-.016 (.09)	0.63	1.58 (.19)	0	25	.49 (.09)	.78	1.56 (.19)	0
13	1.46 (.19)	3.99	1.18 (.36)	.01	26	.84 (.1)	1.77	2.55 (.52)	.05
					27	1.6 (.16)	NA	2.06 (.61)	.04

SCS1. Possible explanations for such behavior include that students generally dislike being tested, even when not graded [19]; a mismatch between the course content and what the SCS1 measures; low sensitivity of pseudocode; or even the perspective of having a low test score stored for future research. We suggest that the SEI may reduce test anxiety, providing more opportunities for students to self-reflect and improve their meta-cognition skills [38] as well as plan their learning.

While we did not specifically collect information regarding the SEI application (e.g. feedback and time on task), students voluntarily provided (negative) feedback on the SCS1. One obvious weakness of testing is that, while potentially more accurate, it can be a time-consuming and strenuous task (only 19.29% of those who attempted the SCS1 completed it in full). Students particularly commented on the short time limit of the SCS1 (an opinion shared by the authors). Recent research shows that extensive mental effort can interfere with learning by depleting cognitive resources [8]. Our results suggest that the SCS1 may put off students; thus, we encourage new approaches that alleviate this behavior, such as self-evaluations or lighter versions of the SCS1 [6]. Given the simplified nature of the SEI, based on the extremely high completion rates (i.e. everyone who attempted the SEI also completed it) and comparable correlations with course grades, we suggest that the SEI can be an efficient additional tool in the assessment process of learning.

Students' growing confidence as measured by the SEI matches the course structure (e.g. very limited effect sizes from week 4 to 7 in all concepts, except for the still "new" ones. See Figure 1). Future research should investigate if such increase and discrimination between the concepts would be even more evident in a course timeline that introduces concepts at a slower pace. The results suggest that the instrument can distinguish between concepts and that this distinction improves with repeated measurements, which corroborates the findings that self-evaluation can improve with practice [38]. While week 1 showed similar loadings for all concepts (except for classes and objects. See Figure 2), the distinction between the concepts became more pronounced and the factor structure changed as students became more knowledgeable and experienced with the contents. Factor loadings and structure of experienced students on week 1 are similar to the entire cohort at the end of the course. This implies that the SEI discriminates concepts early and not only follow the course content timeline.

Metrics of internal consistency showed that repeated measures of the SEI do not decrease its reliability and remain higher than those of the tests (Figure 3). In terms of performance, the SCS1 results show that our context is comparable to previous research (Table 3). However, we know little about self-evaluation ability of these students. Thus, it is difficult to determine their accuracy and compare them with other cohorts, and we can offer only limited transferability claims.

While the SEI correlation with the exam was marginally strong, so was the case with the SCS1 (Figure 5). Ross [38] suggests that such comparisons must be taken with caution since course exams can be biased. Teachers and students may have different expectations about the suitability of an exam to evaluate the learning of course contents. In our case, we stress that while both the SCS1 and prototype SEI focus on code (or concept) comprehension skills, the final course exam was a program writing test.

Due to poor internal correlations and student feedback (see Figure 4), the SCS1 results can be difficult to interpret, and many questions with several interacting concepts make it difficult to measure student knowledge. Other approaches have provided higher correlations with course exams (e.g. [1]), but they require costly and extensive knowledge of student data. Some students also argued that the SCS1 aligns poorly with course contents. Moreover, validated tests are less flexible and difficult to adapt to other contexts, whereas the SEI instrument is designed to be flexible and allow customization. We further discuss possibilities to further improve the SEI in our conclusions.

While the SCS1 performance of our group matches with previous research, the same could not be observed when analyzing question-by-question responses. The IRT analysis showed that not only the difficulties of the questions were ranked differently, but also some questions showed very dissimilar difficulties and performance (Table 3). For example, Q3 can be considered very easy for our group, but moderately difficult to those of Xie et al. [58]. Conversely, we observed the opposite phenomenon in Q6. Q18 was difficult for our group but very difficult for students of Xie et al. (the same applies to Q13 and Q15). While a performance and difficulty analysis could elicit differences between groups, this analysis is incomplete without discussing the test questions themselves. What makes Q6 so hard for one group and not for another? Is this a case of different course structure and content, or perhaps a different phenomenon? For this reason, we suggest that in future the CER community should not only examine the test performance but also analyze the test questions to understand this phenomenon.

7 CONCLUSION

When designing the SEI, our aim was to create a lightweight tool that provides meaningful information to students and instructors, while functioning as a self-evaluation tool for students to support self-regulated learning. Our current findings support the notion that the SEI could be a useful and efficient tool, even as a prototype. It provides fair discrimination of concepts, captures students' increasing confidence in their code comprehension ability, is internally reliable and coherent, allows repeated measures without losing its validity, and can be publicly available. Our tentative framework shows potential to develop other SEIs that can be easy to scale and apply. Furthermore, it allows students from diverse backgrounds to be assessed with relative accuracy and provides students an opportunity to self-reflect without the burden of feeling tested.

However, the instrument and our results have some limitations. The validation of the instrument shows that the prototype SEI achieved some of these goals partially. Measures of divergent validity (e.g. comparing the SEI with self-efficacy instruments) could provide further support of a distinguishable construct. There is certainly an overlap between self-efficacy and self-evaluation. Our results show that, like Danielsiek et al. [12], there is an increase in students' perception of programming related ability and distinguishable factors in the instrument. Whereas Danielsiek et al. [12] evaluates skills such as algorithm design, runtime analysis, and pseudocode writing and tracing, our instrument evaluates the ability *to comprehend programming concepts*.

The SEI still requires stronger evidence of external validation. While our results were comparable in this respect to those of the SCS1, we believe that both the SEI and tests could be improved to provide more reliable and accurate information. The final exam used code writing questions while the SEI evaluates code comprehension ability, which could explain the correlation levels observed in our results. Future research should also aim to compare the SEI and SCS1 with code comprehension course exams.

In addition to the SEI conceptualization and evaluation, we provided a comprehensive analysis of an SCS1 replication. This analysis supports the findings of previous research, provides a detailed comparison between very distinct contexts used in previous works and raises important questions regarding the current CER validated instruments and how the community interprets their results, offering constructive feedback to instrument designers. We propose that future replication studies using validated tools as benchmarks should carefully examine *question-level* performances and dissimilarities in performance as well as investigate the question contents to present argumentation to such dissimilarities.

This study of the prototype SEI shows its potential to outline a *context independent framework that instructors can use to create comparable SEIs tailored for specific needs*. Following Ross [38] guidelines to improve self-evaluation and prior research results [30], we recommend future work to improve reliability and validity of instruments aimed at a particular context (language and behavior).

Tailored instruments could include customizing the number of levels (e.g. an A0 level could be meaningful to K-8 but not to CS1 instructors); the number of sub-levels based on the expected complexity of code that students usually work with; and providing contextualized examples. The number and types of concepts in the SEI should also reflect course contents and contexts; for instance, an SEI for a functional CS1 would focus on different concepts.

We only analyzed a single course in a MOOC context, which reduces generalization claims. In future validations, we aim to improve the clarity of descriptors by collecting student feedback, conduct qualitative investigations of student reasoning while answering the instrument, and replicate this work in other contexts.

8 ACKNOWLEDGEMENTS

We thank the SCS1 authors [35] for making it available for our use and Benjamin Xie for the discussions regarding the methods used in his IRT analysis and SCS1 results [58].

REFERENCES

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 121–130. https://doi.org/10.1145/2787622.2787717

[2] Satu Alaoutinen and Kari Smolander. 2010. Student Self-assessment in a Programming Course Using Bloom's Revised Taxonomy. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '10)*. ACM, New York, NY, USA, 155–159. https://doi.org/10.1145/1822090.1822135

[3] Lorin W Anderson, David R Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Raths, and Merlin C. Wittrock. 2001. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives* (abridged ed.). Addison Wesley Longman. 302 pages.

[4] Albert Bandura. 1977. Self-efficacy: Toward a Unifying Theory of Behavioral Change. *Pyschological Review* 84, 2 (1977), 191–215. https://doi.org/10.1016/0146-6402(78)90002-4

[5] Jens F. Beckmann. 2010. Taming a beast of burden - On some issues with the conceptualisation and operationalisation of cognitive load. *Learning and Instruction* 20, 3 (2010), 250 – 264. https://doi.org/10.1016/j.learninstruc.2009.02.024

[6] Ryan Bockmon, Stephen Cooper, Jonathan Gratch, and Mohsen Dorodchi. 2019. (Re)Validating Cognitive Introductory Computing Instruments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 552–557. https://doi.org/10.1145/3287324.3287372

[7] Janet Carter, Su White, Karen Fraser, Stanislav Kurkovsky, Colette McCreesh, and Malcolm Wieck. 2010. ITiCSE 2010 Working Group Report Motivating Our Top Students. In *Proceedings of the 2010 ITiCSE Working Group Reports (ITiCSE-WGR '10)*. ACM, New York, NY, USA, 29–47. https://doi.org/10.1145/1971681.1971685

[8] Ouhao Chen, Juan C. Castro-Alonso, Fred Paas, and John Sweller. 2018. Extending Cognitive Load Theory to Incorporate Working Memory Resource Depletion: Evidence from the Spacing Effect. *Educational Psychology Review* 30, 2 (01 Jun 2018), 483–501. https://doi.org/10.1007/s10648-017-9426-2

[9] Tony Clear, Anne Philpott, Phil Robbins, and Simon. 2009. Report on the Eighth BRACElet Workshop: BRACElet Technical Report 01/08. *Bulletin of Applied Computing and Information Technology* 7, 1 (2009).

[10] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.

[11] Council of Europe. 2001. *The Common European Framework of Reference for Languages: Learning, teaching, assessment*. Strasbourg. https://rm.coe.int/1680459f97

[12] Holger Danielsiek, Laura Toma, and Jan Vahrenhold. 2017. An Instrument to Assess Self-Efficacy in Introductory Algorithms Courses. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 217–225. https://doi.org/10.1145/3105726.3106171

[13] Rodrigo Duran, Jan-Mikael Rybicki, Arto Hellas, and Sanna Suoranta. 2019. Towards a Common Instrument for Measuring Prior Programming Knowledge. In *Proceedings of the 24th Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2019)*. ACM, New York, NY, USA, 6. https://doi.org/10.1145/3304221.3319755

[14] Rodrigo Duran, Juha Sorva, and Sofia Leite. 2018. Towards an Analysis of Program Complexity From a Cognitive Perspective. In *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18)*. ACM, New York, NY, USA, 21–30. https://doi.org/10.1145/3230977.3230986

[15] J. Feigenspan, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg. 2012. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. 73–82. https://doi.org/10.1109/ICPC.2012.6240511

[16] 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

[17] Shuchi Grover, Stephen Cooper, and Roy Pea. 2014. Assessing Computational Learning in K-12. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, NY, USA, 57–62. https://doi.org/10.1145/2591708.2591713

[18] Dianne Hagan and Selby Markham. 2000. Does It Help to Have Some Programming Experience Before Beginning a Computing Degree Program?. In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE conference on Innovation and Technology in Computer Science Education (ITiCSE '00)*. ACM, New York, NY, USA, 25–28. https://doi.org/10.1145/343048.343063

[19] Ray Hembree. 1988. Correlates, causes, effects, and treatment of test anxiety. *Review of educational research* 58, 1 (1988), 47–77.

[20] Li-tze Hu and Peter M Bentler. 1999. Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural equation modeling: a multidisciplinary journal* 6, 1 (1999), 1–55.

[21] Maria Kallia and Sue Sentance. 2019. Learning to Use Functions: The Relationship Between Misconceptions and Self-Efficacy. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 752–758. https://doi.org/10.1145/3287324.3287377

[22] Danny Kostons, Tamara van Gog, and Fred Paas. 2012. Training self-assessment and task-selection skills: A cognitive approach to improving self-regulated learning. *Learning and Instruction* 22, 2 (2012), 121–132. https://doi.org/10.1016/j.learninstruc.2011.08.004

[23] Michael J. Lee and Andrew J. Ko. 2015. Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 237–246. https://doi.org/10.1145/2787622.2787709

[24] Elizabeth A. Linnenbrink and Paul R. Pintrich. 2003. The Role of Self-Efficacy Beliefs in Student Engagement and Learning in the Classroom. *Reading & Writing Quarterly* 19, 2 (2003), 119–137. https://doi.org/10.1080/10573560308223 arXiv:https://doi.org/10.1080/10573560308223

[25] Raymond Lister. 2016. Toward a Developmental Epistemology of Computer Programming. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education (WiPSCE '16)*. ACM, New York, NY, USA, 5–16. https://doi.org/10.1145/2978249.2978251

[26] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships Between Reading, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the Fourth International Workshop on Computing Education Research (ICER '08)*. ACM, New York, NY, USA, 101–112. https://doi.org/10.1145/1404520.1404531

[27] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2017. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports (ITiCSE-WGR '17)*. ACM, New York, NY, USA, 47–69. https://doi.org/10.1145/3174781.3174784

[28] Lauren Margulieux, Tuba Ayer Ketenci, and Adrienne Decker. 2019. Review of measurements used in computing education research and suggestions for increasing standardization. *Computer Science Education* 29, 1 (2019), 49–78. https://doi.org/10.1080/08993408.2018.1562145 arXiv:https://doi.org/10.1080/08993408.2018.1562145

[29] Susana Masapanta-Carrión and J. Ángel Velázquez-Iturbide. 2018. A Systematic Review of the Use of Bloom's Taxonomy in Computer Science Education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 441–446. https://doi.org/10.1145/3159450.3159491

[30] Peter J. Miller. 2003. The Effect of Scoring Criteria Specificity on Peer and Self-assessment. *Assessment & Evaluation in Higher Education* 28, 4 (2003), 383–394. https://doi.org/10.1080/0260293032000066218 arXiv:https://doi.org/10.1080/0260293032000066218

[31] Laurie Murphy and Josh Tenenberg. 2005. Do Computer Science Students Know What They Know?: A Calibration Study of Data Structure Knowledge. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*. ACM, New York, NY, USA, 148–152. https://doi.org/10.1145/1067445.1067488

[32] Greg L. Nelson, Benjamin Xie, and Andrew J. Ko. 2017. Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/3105726.3106178

[33] Grace Ngai, Winnie W.Y. Lau, Stephen C.F. Chan, and Hong-va Leong. 2010. On the Implementation of Self-assessment in an Introductory Programming Course. *SIGCSE Bull.* 41, 4 (Jan. 2010), 85–89. https://doi.org/10.1145/1709424.1709453

[34] Council of Europe. 2018 (accessed April 1, 2019). *Common European Framework of Reference for Languages: Learning, Teaching, Assessment. Companion Volume with New Descriptors*. https://rm.coe.int/cefr-companion-volume-with-new-descriptors-2018/1680787989

[35] 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

[36] Leo Porter, Cynthia Taylor, and Kevin C. Webb. 2014. Leveraging Open Source Principles for Flexible Concept Inventory Development. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. ACM, New York, NY, USA, 243–248. https://doi.org/10.1145/2591708.2591722

[37] Marcos Román-González, Juan-Carlos Pérez-González, and Carmen Jiménez-Fernández. 2017. Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior* 72 (2017), 678 – 691. https://doi.org/10.1016/j.chb.2016.08.047

[38] John A. Ross. 2006. The Reliability , Validity , and Utility of Self-Assessment. *Practical assessment, research and evaluation* 10 (2006), 1–13. https://doi.org/10.1016/j.aspen.2014.06.014

[39] Sam Saarinen, Shriram Krishnamurthi, Kathi Fisler, and Preston Tunnell Wilson. 2019. Harnessing the Wisdom of the Classes: Classsourcing and Machine Learning for Assessment Instrument Generation. In *Proceedings of the 50th ACM Technical*

Symposium on Computer Science Education (SIGCSE '19). ACM, New York, NY, USA, 606–612. https://doi.org/10.1145/3287324.3287504

[40] Dale H Schunk. 1981. Modeling and attributional effects on children's achievement: A self-efficacy analysis. *Journal of educational psychology* 73, 1 (1981), 93. https://doi.org/10.1037/0022-0663.73.1.93

[41] Michael James Scott and Gheorghita Ghinea. 2014. Measuring Enrichment: The Assembly and Validation of an Instrument to Assess Student Self-beliefs in CS1. In *Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14)*. ACM, New York, NY, USA, 123–130. https://doi.org/10.1145/2632320.2632350

[42] Judy Sheard, Angela Carbone, Selby Markham, A J Hurst, Des Casey, and Chris Avram. 2008. Performance and Progression of First Year ICT Students. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78 (ACE '08)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 119–127. http://dl.acm.org/citation.cfm?id=1379249.1379261

[43] E. Soloway. 1986. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (Sept. 1986), 850–858. https://doi.org/10.1145/6592.6594

[44] John Sweller. 2010. Element Interactivity and Intrinsic, Extraneous, and Germane Cognitive Load. *Educational Psychology Review* 22, 2 (01 Jun 2010), 123–138. https://doi.org/10.1007/s10648-010-9128-5

[45] C. Taylor, D. Zingaro, L. Porter, K.C. Webb, C.B. Lee, and M. Clancy. 2014. Computer science concept inventories: past and future. *Computer Science Education* 24, 4 (2014), 253–276. https://doi.org/10.1080/08993408.2014.970779 arXiv:https://doi.org/10.1080/08993408.2014.970779

[46] Harriet G. Taylor and Luegina C. Mounfield. 1994. Exploration of the Relationship between Prior Computing Experience and Gender on Success in College Computer Science. *Journal of Educational Computing Research* 11, 4 (1994), 291–306. https://doi.org/10.2190/4U0A-36XP-EU5K-H4KV arXiv:https://doi.org/10.2190/4U0A-36XP-EU5K-H4KV

[47] Donna Teague. 2015. *Neo-Piagetian theory and the novice programmer*. Ph.D. Dissertation. Queensland University of Technology.

[48] Donna Teague, Malcolm Corney, Alireza Ahadi, and Raymond Lister. 2013. A Qualitative Think Aloud Study of the Early Neo-piagetian Stages of Reasoning in Novice Programmers. In *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136 (ACE '13)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 87–95. http://dl.acm.org/citation.cfm?id=2667199.2667209

[49] 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

[50] D. Timmermann, C. Kautz, and V. Skwarek. 2016. Evidence-based re-design of an introductory "course programming in C". In *2016 IEEE Frontiers in Education Conference (FIE)*. 1–5. https://doi.org/10.1109/FIE.2016.7757492

[51] Ian Utting, Allison Elliott Tew, Mike McCracken, Lynda Thomas, Dennis Bouvier, Roger Frye, James Paterson, Michael Caspersen, Yifat Ben-David Kolikant, Juha Sorva, and Tadeusz Wilusz. 2013. A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations. In *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports (ITiCSE -WGR '13)*. ACM, New York, NY, USA, 15–32. https://doi.org/10.1145/2543882.2543884

[52] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success. In *Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14)*. ACM, New York, NY, USA, 19–26. https://doi.org/10.1145/2632320.2632349

[53] Quang H. Vuong. 1989. Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses. *Econometrica* 57, 2 (1989), 307–333. https://doi.org/10.2307/1912557

[54] Christopher O. Walker, Barbara A. Greene, and Robert A. Mansell. 2006. Identification with academics, intrinsic/extrinsic motivation, and self-efficacy as predictors of cognitive engagement. *Learning and Individual Differences* 16, 1 (2006), 1 – 12. https://doi.org/10.1016/j.lindif.2005.06.004

[55] David Weintrop and Uri Wilensky. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 101–110. https://doi.org/10.1145/2787622.2787721

[56] Susan Wiedenbeck, Deborah Labelle, and Vennila NR Kain. 2004. Factors affecting course outcomes in introductory programming. In *16th Annual Workshop of the Psychology of Programming Interest Group*.

[57] Brenda Cantwell Wilson and Sharon Shrock. 2001. Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. In *Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education (SIGCSE '01)*. ACM, New York, NY, USA, 184–188. https://doi.org/10.1145/364447.364581

[58] Benjamin Xie, Matthew J. Davidson, Min Li, and Andrew J. Ko. 2019. An Item Response Theory Evaluation of a Language-Independent CS1 Knowledge Assessment. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 699–705. https://doi.org/10.1145/3287324.3287370

[59] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Andrew J. Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2-3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235 arXiv:https://doi.org/10.1080/08993408.2019.1565235

[60] Barry J Zimmerman. 2002. Becoming a Self-Regulated Learner: An Overview. *Theory into practice* 41, 2 (2002), 64–70. https://doi.org/10.1207/s15430421tip4102