What Do Beginning CS Majors Know?

Michela Pedroni
Chair of Software Engineering
ETH Zurich
8092 Zurich, Switzerland
michela.pedroni@inf.ethz.ch

Bertrand Meyer
Chair of Software Engineering
ETH Zurich
8092 Zurich, Switzerland
bertrand.meyer@inf.ethz.ch

Manuel Oriol
Department of Computer Science
University of York
YO10 5DD, York, U.K.
manuel@cs.york.ac.uk

ABSTRACT
The standard "Introduction to Programming" or "CS1" course traditionally assumes that it will be, for most students, the first serious exposure to programming. For the past six years, we have queried our students, in the first weeks of class, about what they know. Results are compelling: virtually all beginning CS students have used computers for over two years, and many for ten years or more; on average, they know at least one programming language in depth; many have written significant systems. These and other measures of prior knowledge have been stable over the query period. This article analyzes both the results obtained and their pedagogical implications for courses and textbooks.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – *computer science education.*

General Terms
Human Factors.

Keywords
programming experience, student diversity.

1. INTRODUCTION
To "know your audience" is one of the fundamental rules of mass communication [13]. This particularly applies to an educational setting: understanding the backgrounds of students is essential for providing quality educational programs tailored to their interests and needs. Instructors of advanced courses for computer science majors are generally able to take a certain basic knowledge of computing and programming as granted. This is rarely the case for introductory programming courses where students start with a variety of backgrounds; "students are diverse in terms of their prior experiences, their pre-existing skills, their expectations and their motivations" [6]. While confirming this diversity, however, the present study suggests that teachers can actually rely on certain assumptions regarding computer literacy and programming experience.

For the past six years, we have carefully tracked to what extent our first-year undergraduate CS students are familiar with computers and how proficient they are at programming and programming languages. The questionnaire changed only slightly over time to include items that better capture students' answers. The study reports on 753 questionnaires filled in by students in the first weeks of the course. As expected, most students used computers for more than two years when they enter the course. A remarkable result is the average proficiency that students show at programming; what we think of as being an introductory programming course has actually fewer than 20% programming novices; and over half of the students know at least one programming language well before starting their studies. Tracking the evolution of answers shows an increase in popularity for web-related languages (Java, JavaScript and PHP) at the expense of more traditional languages such as Pascal and Basic.

Section 2 details the setup of the questionnaire and discusses specific aspects that may affect generalization of the results. Sections 3, 4, and 5 analyze these results, respectively on computer experience, programming experience, and programming languages; each such section first presents the raw results, then proposes some interpretations for the more surprising aspects, and immediate implications for teachers. Section 6 draws up some lessons from these results for the teaching of introductory programming. Section 7 shows related work. Section 8 presents conclusions and future work.

2. QUESTIONNAIRE SETUP
Since fall 2003, the first semester Computer Science majors at ETH attending the Introduction to Programming course fill in a questionnaire in the first weeks of the semester. Partial results of this questionnaire were already presented in [8].

"Introduction to Programming" is offered in the very first semester as the only computer science course and a required step for future computer science graduates on their way to a bachelor's and possibly a master's degree.

Most of the students that start a CS program at ETH come from one of the Swiss high schools where they graduated with the so-called "Maturity" degree. The Swiss high school system is decentralized: while a federal regulatory instrument sets general standards for the Maturity, each of the 26 cantons implements it with its own school laws. In the computing area, most high schools offer introductory courses on computer applications (text processing, table calculations, web surfing), but very few teach computer science, or programming using a higher-level language. Until 2007, computer science was not mentioned in the Swiss high school regulations; it recently became an optional

supplementary subject, with implementation starting fall 2008. It will be interesting to see how this affects the backgrounds of CS students.

The number of students that participated in the courses is approximately 1130 (250 in 2003, 180 in 2004, 170 in 2005, 160 in 2006, 170 in 2007, and 200 in 2008). In the first iteration, the questionnaire was handed out on paper in class; in the following years it was available online. The results, tracked over these six years, form the basis for the rest of the present discussion.

Threats to validity and limitations. While we believe that many of the conclusions apply to the teaching of computer science anywhere, a number of specifics may limit generalization.

The Swiss practice of selective high schools, which in effect screen our incoming students for us, may bias the sample of surveyed students towards higher competence.

The absence of computer science in Swiss high schools (as opposed to many other countries) may bias the results in the reverse direction.

Another threat to validity of the survey is that it does not measure students' prior experience objectively, but through their own self-appraisal. It is unclear if this introduces a bias in any direction.

Finally, the switch from a paper questionnaire filled out in class to a voluntary online form caused a decrease in participation and introduces the risk of self-selection, another possible limitation.

To minimize the risk of having an unrepresentative sample of students as participants to the questionnaire, we ask students to rate their prior programming expertise again in the official end-of-semester course evaluation questionnaire required and administered by the university and handed out on paper during class. The results of that second test essentially coincide with the initial results, with the exception of a punctual discrepancy (23% of novices from the university questionnaire vs. 13%) for one single year, 2007.

An obvious potential limitation of this work is that it is mainly based on results from one institution. Although we cannot authoritatively claim generalization to other universities and countries, we did perform a similar test in a second institution in a different country. The results from the student group at University of York are very similar to the results at ETH; in particular, they exhibit no significant differences concerning the computer literacy outcomes, prior programming knowledge, and the number of languages that an average student knows a little, well and very well. A comparison is available in a separate report [9].

3. COMPUTER LITERACY

Without knowing how to use a computer it is extremely difficult even to consider learning how to program. This is usually the first concern of a CS1 educator. In our setup, Figure 1 shows that this concern is no longer justified.

The class of 2003 seems to have been less exposed to computing than later ones. For 2006-2008, more than half of the students had used a computer for over ten years; with a median age of around 20, the computer has been part of their life for at least half of it.

In line with these findings, the percentage of students that have a desktop computer at home has risen from 87% in 2003 to percentages between 95% and 98% in the following years. Similarly, the percentages of students who own a laptop increased from 56% in 2003 to 75%-93% in the next years.

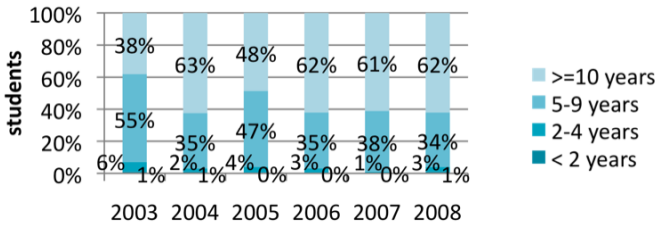

Figure 1: Time during which students have used computers

Interpretation. These findings come as no surprise, as today younger people typically use computers daily to read e-mails, surf the web and build up communities. Moreover, people attracted to computer science programs usually exhibit a strong interest in new media and technology.

Teaching implications. The survey does not indicate how deeply students understand the concepts behind computers and computer architecture. But the immediate lesson for CS1 instructors is that they do not need to fret about "computer literacy". Students are familiar with computers, and instructors can go straight into programming if this is the goal of the course.

4. PROGRAMMING EXPERIENCE

Table 1 shows the programming experience of students, broken down into the categories "no programming" (never programmed before), "no O-O" (programmed, but never with an object-oriented language), "small project" (worked on object-oriented projects consisting of less than a hundred classes) and "large project" (worked on O-O projects with hundreds of classes — a sizable experience for supposed novices).

Table 1: Programming experience

year	gender	number of students	no pro-gram-ming	some experience		
				no O-O	some O-O	
					small project	large pr. (> 100 classes)
2003	total	222	22%	39%	34%	5%
	male	203 (91%)	19%	39%	37%	5%
	female	19 (9%)	53%	42%	5%	0%
2004	total	127	14%	33%	43%	10%
	male	117 (92%)	11%	34%	44%	11%
	female	10 (8%)	50%	20%	30%	0%
2005	total	95	18%	25%	42%	15%
	male	81 (85%)	12%	26%	46%	16%
	female	14 (15%)	50%	22%	21%	7%
2006	total	97	19%	27%	43%	11%
	male	84 (87%)	18%	25%	44%	13%
	female	13 (13%)	23%	39%	38%	0%
2007	total	88	13%	20%	59%	8%
	male	84 (95%)	13%	19%	60%	8%
	female	4 (5%)	0%	50%	50%	0%
2008	total	124	18%	22%	43%	17%
	male	113 (91%)	16%	22%	45%	17%
	female	11 (9%)	46%	18%	18%	18%

The number of female students participating in the questionnaire varied between 5% and 15%, reflecting the actual ratio of female first semester CS students in our university. The table indicates

that the figures do not differ markedly between the genders, except for the higher number of total beginners among females; this measure may not, however, have any profound significance given the small size of the sample.

Figure 2 visualizes the results of Table 1 for students of both genders. It indicates that an increasing subset of the students start with experience in O-O programming, while the percentage of those with non-OO language experience has dropped.

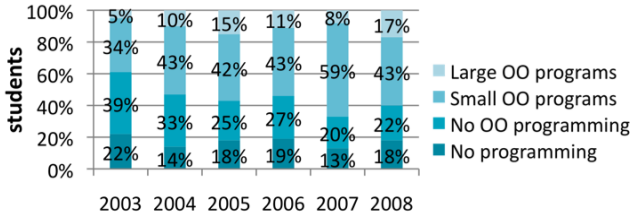

Figure 2: Programming experience

There is no immediately identifiable change trend over the years in the number of novices, which hovers around one-sixth to one-fifth of each class.

Another item on the questionnaires asked students where they have learnt to program. On average over the six years, 55% of all students stated that they learnt programming by themselves; 18% are novices; only 17% took a programming course at high school and the remaining 10% learnt it at university, at work or on another occasion (such as courses at an evening school).

Interpretation. A possible reason for the lower exposure to computing of the class surveyed in 2003, could be that this was still just "after the Internet bubble burst", after which more students have been attracted to computer science by genuine interest. Observations that would seem to support such a hypothesis include: the highest percentage of novice programmers (22% including both genders) for the year of 2003; the above-average numbers of CS enrollments in that year (although an alternative explanation for that particular phenomenon could be a change that occurred in the Swiss high school system); and our own informal observation that students in subsequent years seemed more genuinely interested in CS.

The increase in object-oriented language experience is probably due to the increasing spread of O-O languages such as Java (see also Section 5).

Teaching implications. The evidence on prior O-O language use has a consequence for teachers: while those of professor age may have first encountered object technology as a leading-edge development, possibly even still with a slightly sulfurous flavor, such qualms are irrelevant today. For students who have already programmed, O-O is given and needs no particular apology or justification.

Tempering this lesson coming from the questionnaire data is a more subjective observation from our informal interactions with students: many do not fully grasp the more sophisticated properties of object technology, such as polymorphism, dynamic binding and other architectural techniques. They realize this lack of solid theoretical understanding and are eager to correct it by attending the course. It seems more useful to explain these concepts in depth than to take pains to justify the use of objects.

Another important conclusion arises from studying the other end of the data: the persistence of the "no prior programming" 15%-20% minority. It raises significant challenges for teachers, especially when assessed against the only slightly lower percentage of those who have programmed fairly large object-oriented systems. It is hard to think of another academic field, which, at the start of studies, faces such heterogeneity. The variety of prior programming expertise is, in our experience, one of the largest obstacles facing introductory programming teaching today.

5. PROGRAMMING LANGUAGES

As part of the questionnaire, students were asked to rate 15 programming languages (ranging from Java, PHP and C++ to Fortran, Eiffel and Python; for a full list see Figure 4) whether they know it *not at all*, *a little*, *well* or *very well*. The answers to these questions (Table 2) reveal that an average student knows — in his or her self-evaluation — two to three of the languages a little and at least one of the languages well.

Table 2: Average (and median) number of languages known

	2003	2004	2005	2006	2007	2008
a little	1.8 (2)	3.2 (2)	3.2 (3)	2.8 (3)	2.6 (2)	2.4 (2)
well	1.0 (0)	1.1 (0)	1.4 (1)	1.2 (1)	1.3 (1)	1.2 (1)
very well	0.2 (0)	0.6 (0)	0.6 (0)	0.7 (0)	0.6 (0)	0.5 (0)

Considering the number of programming languages students know *well* or *very well*, Figure 3 confirms that almost half of the current students have sound proficiency in two or more languages and that at least one third of all students have not really mastered any of the languages (these numbers include the students that stated being novice programmers).

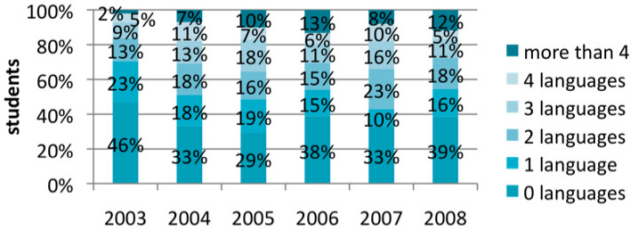

Figure 3: Number of languages known well or very well

Questionnaire items on the level of familiarity of the 15 programming languages help answer additional questions: (1) What are the most known languages among them? (2) Are there languages with growing or dropping popularity with this particular population?

Figure 4 shows the 15 programming languages and the percentages of students with the four levels of familiarity (knowing the language in question *not at all*, *a little*, *well* and *very well*). Some of the languages, marked *, were only included in the survey after the first iterations. The analysis takes into consideration the answers from all students (including programming novices) and does not distinguish between years.

The web scripting language PHP is the most popular, having both the highest number of students who state they know it *very well* and the fewest students who don't know it at all. Other popular languages are C/C++, Java/JavaScript, and Basic/VisualBasic. The top three languages (i.e. the languages where the least students state that they don't know it at all), discriminated by year, include most of the languages rated as most known totaled over the years (see Table 3). C++ is an evergreen - it appears almost every year in the list of the three top languages. Since 2005, Java,

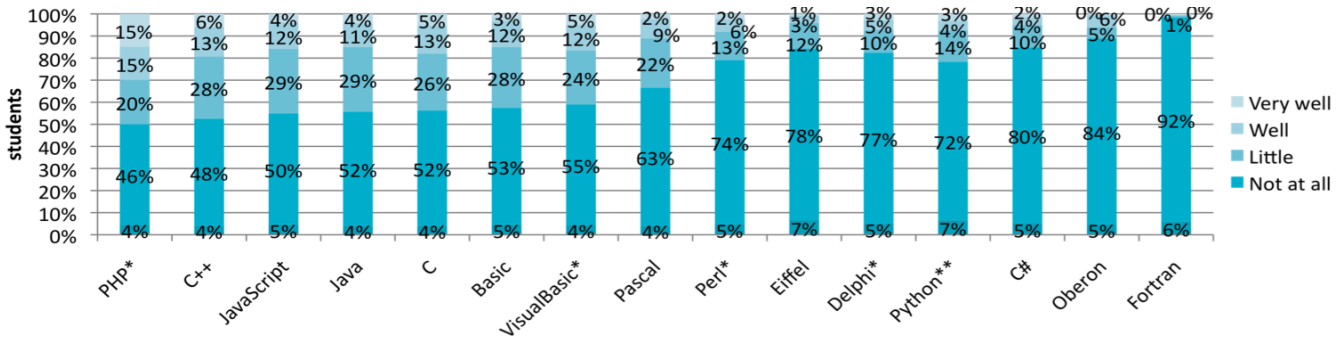

Figure 4: Overall popularity of programming languages

JavaScript and PHP also strengthened their position and for the last two years around 50% of all students have worked with PHP, JavaScript and/or Java before starting to study CS.

Table 3: Programming languages rankings (JS: JavaScript)

ranking	2003	2004	2005	2006	2007	2008
1st	Basic	Eiffel	C++	PHP	PHP	PHP
2nd	Pascal	C++	Java	JS	Java	C++
3rd	C++	JS	PHP	Java	JS/C++	C

These languages also belong to the list of programming languages with increasing popularity amongst the students. Figure 5 shows the most popular programming languages that have a rising tendency in the percentage of students that state to know it *a little*, *well*, or *very well*, i.e. the percentage of students that do not know the programming language at all has decreased since 2003. The programming languages Basic, Pascal and VisualBasic exhibit a decreasing trend in students' level of familiarity.

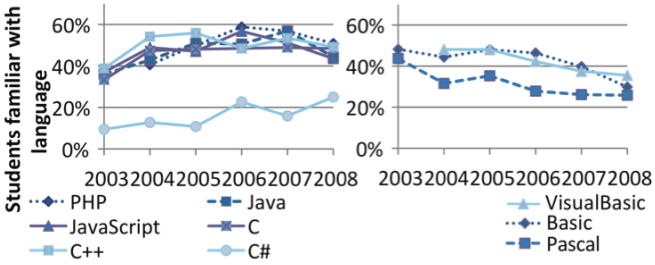

Figure 5: Evolution of popularity of programming languages

Interpretation. The popularity of languages such as JavaScript and PHP most likely reflects that many students' prior experience has been with web applications.

Note that our results are limited to the 15 languages itemized in the questionnaire: a student may know additional languages.

Teaching implications. These results show that when teaching introductory programming we need to take into account that the number of students who need to learn programming almost from scratch is higher than the 10% to 20% who are total beginners.

In particular it may well be that students whose programming has mostly been with Web applications in PHP or JavaScript are adept at writing user interface operations but only have superficial experience with loops, recursion, data structures and other standard computer science techniques. While our questionnaire does not test this conjecture, it is definitely supported by informal observations. If correct, we should not consider that proficiency at GUI and Web programming implies proficiency at concepts and skills of professional software development, meaning that we need to take extra care with the teaching of fundamental topics.

6. SUMMARY AND EFFECT ON TEACHING

The most prominent outcome of the questionnaire is a confirmation that the introductory programming course at our institution has been and - given the mostly stable situation – will be faced with a very diverse student body.

At one end, a considerable fraction of students have no prior programming experience at all (between 13% and 22%) or only moderate knowledge of some of the cited languages (around 30%). At the present stage the evidence does not suggest a decrease in either of these phenomena.

At the other end, the course faces a large portion of students with expertise in multiple programming languages (around 30% know more than three languages in depth). In fact, many have worked in a job where programming was a substantial part (24% in 2003, 30% in 2004, 26% in 2005, 35% in 2006, 31% in 2007 and 2008).

An increasing percentage of students who have programming experience used an object-oriented language; correspondingly, fewer students take the course without prior O-O exposure.

If we try to picture the "average" student taking Introduction to Programming at ETH, he knows one programming language in depth and another two to three languages slightly. His favorite programming languages are Java, JavaScript, PHP, and C++. He has learnt his first programming language in self-study.

The rest of this section presents measures proposed to adapt to such students.

Adapting the course material. As a first and simple option, if we want students with prior knowledge to understand courses better, we must connect to that knowledge. This can help adapt the course to students' needs; when introducing a concept, for example, instructors can provide references to its counterpart in the most known programming languages. They may consider going further and organizing special exercise groups for students with in-depth experience with a specific programming language.

Adapting the teaching methodology. Because the majority of CS1 students already know a programming language, it seems more natural to offer access to the whole libraries and to a complex development environment, thus letting the more curious students explore a richer environment. This is the technique used in the Inverted Curriculum approach [8]. While more novice students content themselves with the library's APIs, their advanced colleagues may explore the library's internals, discover

the more advanced aspects, and enhance their competence through imitation and inspiration.

Making student groups. Students who had learned a programming language prior to the CS1 course are, overall, more successful than novices [4, 11]. It is likely that the extra experience with other programming languages provides intellectual preparation for mastering the intricacies of software development, for which novices enjoy no counterpart. To redress this imbalance, it may be interesting to allow novice students to take extra lessons on programming either before the semester starts (such as in a CS0 course [1]) or during the semester.

Emphasizing concepts. One of the most important lessons is an answer (or at least elements of an answer) to perennial questions of introductory CS education: concepts vs. skills. On whether to teach concepts or skills, the easy answer — teach both — does not suffice, since the question is really about emphasis. Our view is that we should teach *selected skills illustrating important concepts*. The study results support the view that we can indeed be selective: it is pointless to teach PHP or JavaScript as many students know these technologies already, and the others will pick them up when they need them. We should teach skills (otherwise we train pure theoreticians), but the ones we select should illustrate important computer science concepts that will continue to help students when the technology has changed and they need to learn the new buzz du jour.

7. RELATED WORK

A number of studies have provided information on students' prior computing and programming knowledge. They yield some important insights for the present work, but in most cases the issue of prior experience is subsidiary to the authors' main interest rather than focus of attention as in the present work. Sometimes the main issue is gender differences, as in [2, 7, 12] for CS majors and [5] for general students. In other cases the focus is on prediction of success in introductory programming courses as in [3, 4, 10, 11, 14]. None of these studies provide data to investigate the stability of the situation concerning prior computing and programming knowledge of CS majors.

8. FUTURE WORK AND CONCLUSIONS

We will continue to track students' prior experience, which we view as an indispensable tool for tuning courses to the real students of the 21st century. The questionnaires have proved extremely useful in this endeavor, but we clearly need to continue refining them, if only to get some objective data on the conjectures raised in this article. In particular, we plan to address the issue of generalization by collecting data at other technical universities (as already done for the University of York). Also we would like to extend the questionnaire to query students about standard computer science knowledge, such as data structures, algorithms, and design patterns.

That one can be even thinking of asking such questions of 19- or 20-year-olds (and, based on informal probes so far, expecting to be positively surprised) shows how broadly some part of CS concepts have reached some of the world at large, including the younger segments of the population.

As this article has shown, none of this makes CS1 any easier to teach. Traditionally, the difficulty had been that students do not have any prior knowledge of the material. In the case of the introductory programming course for CS majors at our institution, this assumption does not hold anymore for the average student of the course. The "average student" already knows one programming language *well* and two or three programming languages *a little*. He most probably has learnt these languages on his own and thus might not have good design principles or a clear idea of how things should be coded. But he knows more than what we think, and we need to adapt to this new generation of students. The challenge, for those who teach introductory programming, is that we can neither ignore this background, as we would disconnect from a majority of the students; nor rely on it, as we would disconnect from a significant minority who still does not possess it. We have to build on it where present, make up for it where absent, and tackle it throughout our teaching as one of the many challenges defining this exciting pedagogical endeavor.

9. REFERENCES

[1] C. Dierbach, B. Taylor, H. Zhou and I. Zimand. Experiences with a CS0 course targeted for CS1 success. *SIGCSE Bull.* 37(1):317-320, 2005.

[2] A. Fisher, J. Margolis and F. Miller. Undergraduate women in computer science: experience, motivation and culture. *SIGCSE Bull.* 29(1):106-110, 1997.

[3] A. Gomes and A. Mendes. A study on student's characteristics and programming learning. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*, 2895–2904, Vienna, AT, 2008.

[4] D. Hagan and S. Markham. Does it help to have some programming experience before beginning a computing degree program? *SIGCSE Bull*. 32(1):25–28, 2000.

[5] M. E. Hoffman and D. R. Vance. Computer literacy: what students know and from whom they learned it. *SIGCSE Bull.* 37(1):356-360, 2005.

[6] T. Jenkins and J. Davy. Diversity and motivation in introductory programming. Innovation in Teaching and Learning in Information and Computer Sciences, 1(1), 2002.

[7] E. M. Madigan, M. Goodfellow and J. A. Stone. Gender, perceptions, and reality: technological literacy among first-year students. *SIGCSE Bull.* 39(1):410-414, 2007.

[8] M. Pedroni and B. Meyer. The inverted curriculum in practice. *SIGCSE Bull.* 38(1):481–485, 2006.

[9] M. Pedroni and Manuel Oriol. A comparison of CS student backgrounds at two technical universities. Technical Report 613, ETH Zurich, Chair of Software Engineering, 2009.

[10] P. Ventura and B. Ramamurthy. Wanted: CS1 students. No experience required. *SIGCSE Bull.* 36(1):240-244, 2004.

[11] J. S. Rosenschein, T. Vilner, and E. Zur. Work in progress: Programming knowledge - does it affect success in the course "Introduction to Computer Science Using Java". In *FIE'04: 34th Frontiers in Education*, pp.T2H/3-T2H/4, 2004.

[12] M. G. Sackrowitz and A. P. Parelius. An unlevel playing field: women in the introductory computer science courses *SIGCSE Bull.* 28(1):37—41, 1996.

[13] W. Schramm, How communication works, Mass Media & Society, Ablex Pub. Corp., 1997.

[14] B. Wilson. A Study of Factors Promoting Success in Computer Science Including Gender Differences. *Computer Science Education,* 12, 141-164, 2002.