Informatics in Education, 2015, Vol. 14, No. 2, 143–160

DOI: 10.15388/infedu.2015.09

Programming Language Use in US Academia and Industry

Latifa BEN ARFA RABAI[1], Barry COHEN[2], Ali MILI[2]
[1] *Institut Superieur de Gestion, Bardo, 2000, Tunisia*
[2] *CCS, NJIT, Newark NJ 07102-1982 USA*
e-mail: latifa.rabai@isg.rnu.tn, barry.cohen@njit.edu, ali.mili@njit.edu

Received: July 2014

Abstract. In the same way that natural languages influence and shape the way we think, programming languages have a profound impact on the way a programmer analyzes a problem and formulates its solution in the form of a program. To the extent that a first programming course is likely to determine the student's approach to program design, program analysis, and programming methodology, the choice of the programming language used in the first programming course is likely to be very important. In this paper, we report on a recent survey we conducted on programming language use in US academic institutions, and discuss the significance of our data by comparison with programming language use in industry.

Keywords: programming language use, academic institution, academic trends, programming language evolution, programming language adoption.

1. Introduction: Programming Language Adoption

The process by which organizations and individuals adopt technology trends is complex, as it involves many diverse factors; it is also paradoxical and counter-intuitive, hence difficult to model (Clements, 2006; Warren, 2006; John C, 2006; Leo and Rabkin, 2013; Geoffrey, 2002; Geoffrey, 2002a; Yi, Li and Mili, 2007; Stephen, 2006). This general observation applies to programming languages in particular, where many carefully designed languages that have superior technical attributes fail to be widely adopted, while languages that start with modest ambitions and limited scope go on to be widely used in industry and in academia. In (Dios, Mili, Wu and Wang, 2005) we used an empirical approach to build a statistical model that captures the evolution of programming language adoption by a variety of stakeholder classes (industry, academia, government, etc), and in (Bai and Mili, 2011; Ben Arfa Rabai, Bai and Mili, 2011; Ben Arfa Rabai, Bai and Mili, 2009) we generalize this model to a broader class of software technology trends.

In this paper, we present factual data on the adoption of programming languages in academia and industry, and attempt to identify trends over time, by comparing cur-

rent data against 2010 data; we also analyze possible cross-influences between adoption trends in academia and industry; we also analyze possible correlations between language adoption decisions in academia and institutional rankings. This information may be of interest to academic decision makers, as they may want to consider what languages are being used across academia, and may be of interest to industry decision makers and recruiters, as they contemplate what background graduating students have in terms of knowledge of programming languages and paradigms.

2. Programming Language Adoption in Industry

The *Tiobe Software* company (`http://www.tiobe.com`) offers one of the most comprehensive, and most timely, surveys of programming language use. This survey appears to use online resources to assess the use of programming languages in industrial practice worldwide, and updates its estimates on a monthly basis. For our purposes, we are interested to review the degree of usage of the most common programming languages as of April 2013; in order to analyze evolutionary trends, and to compare with the data we collected on the use of programming languages in academia, we also record usage data for April 2010. This data is shown in the Table 1:

Table1

Tiobe Programming Community Index, 2010–2013

Language	Rank 2013	Percentage 2013	Rank 2010	Percentage 2010	Evolution Percentage	Evolution Rank
C	1	17.86	1	18.06	–0.20	0
Java	2	17.68	2	18.05	–0.37	0
C++	3	9.71	3	9.71	0.01	0
Objective-C	4	9.60	11	2.29	7.31	7
C#	5	6.15	6	4.43	1.71	1
PHP	6	5.43	4	9.66	–4.23	–2
Visual Basic	7	4.70	5	6.39	–1.69	–2
Python	8	4.44	7	4.20	0.24	–1
Perl	9	2.33	8	3.55	–1.22	–1
Ruby	10	1.97	12	2.22	–0.25	2
JavaScript	11	1.51	10	2.47	–.096	–1
VB .NET	12	1.09				
Lisp	13	0.90				
Pascal	14	0.89	16	0.65	0.24	2
Delphi	15	0.84	9	2.71	–1.87	–6
Bash	16	0.84				
Transact-SQL	17	0.72				
PL/SQL	18	0.71	14	0.71	0.00	4
Assembly	19	0.71				
Lua	20	0.65	20	0.52	0.13	0

Interestingly, the three top contenders remain the same, and in the same order, namely C, Java then C++. The big winner, in terms of positive evolution over the three year period is Objective-C, which jumps forward a full seven ranks, thanks to an increase of 7.310 in its adoptive population. The biggest loser in terms of adoptive population is PHP, which loses 4.234 percent of the programmer population; and the biggest loser in terms of ranking is Delphi, which drops by six positions (from 9th to 15th). In the next section we explore the ranking of languages in academia.

Considering alternative sources of information, we have looked at data from the site `http://langpop.com/`, which dates back to the same period (Fall 2013). Specifically, we have focused on two metrics that this site is interested in, namely:

- **Programming language use.** In this metric, the authors attempt to gauge the level of use of programming languages by combining data from a variety of sources, including google search (a generic search for references to programming languages), github (a search that focuses on open source software), google files (a search of files with language-specific extensions), craigslist (a search of job postings on craigslist), Ohloh (which measures the number of programmers contributing code to open source projects). We ran the normalized computation on the basis of github and google search (assigning a weight of 0 to the other three), giving google search a weight of 2 and github a weight of one, because google search is more generic (whereas github is specific to open source). We give the other three a weight of zero: google files because it is biased (some languages generate more files per application than others), ohloh because it is redundant with github (which is more widely known and used), and craigslist because its data is incidental (it is a broad spectrum site, in which software job posting are only a small fraction, and is not the prime destination of software professionals). With these weights, we find the following twenty languages at the top: C, Java, C++, Objective-C, PHP, JavaScript, Python, Ruby, C#, Visual Basic, Perl, Shell, SQL, Delphi, ASP, Assembler, Scala, Cobol, Pascal, Lua. Out of these twenty languages, a full sixteen are in the Tiobe survey; and the four top languages (i.e. C, Java, C++, Objective-C) are in the same order in the two lists.
- **Programming language interest.** It has always been our belief, and our observation, that what makes a language popular is not necessarily its intrinsic quality attributes, but a host of incidental environmental and circumstantial extrinsic factors; so that we feel vindicated that the site `http://langpop.com/` finds it necessary to survey languages according to their level of interest, in addition to a survey based on language usage. To this effect, they collect data from sites that programmers visit to talk about programming languages; they argue that what languages programmers are interested in, and are experimenting with, are not necessarily the same as what languages programmers are paid to use. The site refers to three sources, namely: *Lambda the Ultimate*, which is rather academically oriented, and attracts programming language researchers; *programming.reddit.com*, which is a combined news site/ social networking site for programmers; and *slashdot.org,* which has a similar audience to reddit, but is smaller and less influential. We computed normalized results by giving reddit a weight of 2

and Lambda a weight of 1 (to lower its impact, since it is academically oriented and we are interested in industrial trends) and Slashdot a weight of 1 (due to its lower impact/ importance). The resulting table provides the following list as the twenty most interesting programming languages far the Fall 2013: Java, Java Script, Python, PHP, Perl, C++, Ruby, C, SQL, Lisp, Scheme, Haskell, C#, Shell, D, Erlang, Cobol, Assembler, Scala, Objective C. Out of these languages, only thirteen are part of the Tiobe survey, and many that are in both surveys are at widely different ranks.

Another source of programming language use in industry is RedMonk, which shows a table of language usage in two forums, namely *Stack Overflow* (an open forum for professional programmers) and *GitHub* (an open source forum). In the right hand corner of the chart, RedMonk shows the languages that are the quarter percentile of both rankings; these include Java, Java Script, PHP, Python, C++, Ruby, C#, C, CSS, Objective C, R, Perl, Shell, Scala, and Haskell. Of these, ten are among Tiobe's list of twenty top languages.

In a recent posting on `http://www.mashable.com`, Todd Wasserman lists the following languages as important languages that a modern programmer ought to know: Java, Java Script, C#, PHP, C++, Python, C, SQL, Ruby, Objective C, Perl, .NET, Visual Basic, R, Swift. These languages are selected and ordered on the basis of their importance for programmers at the high end of the pay scale, according to the online learning platform Lynda (`http://www.lynda.com/`). Out of these fifteen languages, no less than thirteen show up in Tiobe's list for April 2013 (whereas the mashable list is dated 2015, it must be noted).

Overall, it is fair to consider that the Tiobe list is a faithful indicator of the state of the practice in language usage in the software industry.

3. Programming Language Adoption in Academia

During the spring semester 2013 (January to April 2013) we have conducted a survey across US institutions of higher education, collecting data on programming language use for teaching; specifically, we collected the following data:

- What programming language is used for the first computing course; some institutions (such as NJIT, for example) have an introductory computing course that precedes the first programming course, and is a prerequisite thereof. Such a course is intended to expose incoming freshmen to general computing concepts, including (but not limited to) programming; hence the programming part of the course is covered using a user-friendly language that is not necessarily the language of their first programming course.
- What programming language is used for the first programming course? The focus of this course is to teach programming using a programming language as a medium, though it is not uncommon for this course to be geared towards teaching the

programming language as much as (or more than) it is geared towards teaching a programming discipline.

- What programming language is used for the first data structures course? Of course, this is most typically the same language as that used for the first programming course, but sometimes (more often than we thought) they are different.
- What languages are covered in the programming language course; this is typically a junior level course that explores general issues of programming languages, such as programming language analysis, programming language design, programming language processing, programming language compilers and interpreters, and programming paradigms, and exposes students to some programming languages for practical assignments.

In order to record evolutionary trends, we have collected this data for the spring semester 2013 and the spring semester 2010. We have collected this data for 134 institutions across the US, ranked 1 to 134 in the latest *US News and World Report* Survey. For the Spring 2013 semester, this data is collected by merely inspecting relevant course catalogs, course schedules and (when available) course sites. For the Spring semester 2010, it is more difficult to collect this data, as it requires that we find three year old course sites, course catalogs, or course syllabi; occasionally we had to write individual emails to instructors and/or administrators, with limited success; hence we have fewer data points for 2010 than for 2013.

3.1. *First Programming Course*

Table 2 shows the data pertaining to the programming language used in the first programming course in the spring semester 2013 and the spring semester 2010.

Table 2
Programming Language Adoption in Academia, 2010–2013
First programming Course

Language	Rank 2013	Percentage 2013	Rank 2010	Percentage 2010	Evolution Percentage	Evolution Rank
Java	1	44.44	1	51.66	–7.22	0
C++	2	19.26	2	26.66	–7.41	0
Python	3	17.04	4	5.00	12.04	1
C	4	13.33	3	10.00	3.33	–1
MatLab	5	1.481	6	1.66	–0.18	1
C#	6	0.74	7	0.00	0.74	1
Haskell	6	0.74	7	0.00	0.74	1
PHP	6	0.74	7	0.00	0.74	1
JavaScript	6	0.74	7	0.00	0.74	1
Scheme	6	0.74	5	3.33	–2.59	–1
Racket	6	0.74	7	0.00	0.74	1
Ruby	7	0.00	6	1.66	–1.66	–1

Before we compare these results with the *Tiobe* data, we need to make the following observations:

- While the data in this table pertains exclusively to academic institutions, the data collected by *Tiobe Software* is based on "the number of skilled engineers worldwide, courses, and third party vendors". Assuming that "courses" refer to industrial courses, in addition, possibly to academic courses, we feel it is fair to consider that the *Tiobe* data reflects primarily the industrial trends of the moment.
- While our data pertains exclusively to US academic institutions, the *Tiobe* data reflects industrial practice worldwide. We see no compelling reason to believe that industrial practice in the US (in terms of programming language preferences) should be radically different from industrial practice elsewhere, but we need to be mindful of this qualification.

With these qualifications in mind, we make the following observations:

- C, Java and C++ are in the top four languages in academia and in industry, in 2010 and in 2013. But while C is ranked #1 in industry in 2001 and 2013 (perhaps due to the weight of legacy software), it is ranked 4th in academia in 2013, and 3rd in 2010. Academic institutions have more latitude in switching between languages than does industry.
- The distribution of languages in academia is less uniform than the distribution of languages in industry: Java is ranked first in academia with a whopping 44.44%, whereas C is ranked first with a mere 17.862%.
- Another language to watch, besides the three top languages cited above, is Python. With 17.037 % of the market share in academia in 2013, it is nearly as prevalent as the top languages in industry (17.862% for C, and 17.681% for Java). Perhaps more interestingly, its presence jumps from 5.00% in 2010 to 17.037% in 2013. In industry, this language garners 4.442% of the market in 2013, slightly up from its showing of 2010 (4.205 %).
- Among the languages that are used in industry but shunned in academia, it is worth pointing to Object-C, whose market share is a significant 9.598 %, and to C#, whose market share in industry is 6.150 %.
- Some of the languages that appear in academia but not industry include MatLab, Haskell, Scheme and Racket. The rationale for using a language that is not used in industry is that we want a language that best supports a programming discipline, and that once students acquire a sound discipline, migrating to another language is a simple matter (Yi, Li and Mili, 2007).

In order to get a clearer sense of which languages are gaining ground in academia (in a first programming course), and which languages are losing ground, we have considered the four top languages of the table above and recorded how universities have (or have not) changed their adopted language from 2010 to 2013. The results are summarized in the matrix below, where rows represent the languages adopted in 2010 and columns represent the languages adopted in 2013. The diagonal represents the number of institutions that have maintained their choice of language, and outside the diagonal we represent the number of institutions that have moved from the language represented

2013 2010	C++	Java	Python	C	Loss
C++	16	1		1	2
Java	1	29	4	1	6
Python			5	1	1
C		2	1	7	3
Gain	1	3	5	3	

in row to the language represented in column. From this table, it is clear that Python is showing the greatest positive evolution (loss of 1, gain of 5), even though it currently has the lowest adoption rate.

An interesting question that we want to explore is whether the choice of languages for the first programming course is correlated with institutional rank; to this effect, we divide our sample of 134 institutions into four quartiles according to their ranking in the latest *US News and World Report* survey (1 to 33, 34 to 66, 67 to 99, and finally 100 to 134). For completeness, we have also added a column for language adoption in MOOCs (Massive Open Online Course), including sites such as Coursera, edX, Udacity, Udemy, Codecademy, Lynda.com and Treehouse. The results, which we limit to the nine top languages of *Tiobe*'s survey for April 2013, are summarized in the Table 3:

The only trend that appears to be monotonic is the percentage of adoption of C++, which increases from 14.286 % for first tier institutions to 34.286 % for fourth tier institutions. From the first tier to the third tier, the adoption of Java drops precipitously, and is compensated almost perfectly by the adoption of Python. Except for the fact that it includes many languages (such as Ruby, JavaScript, CSS, HTML, HTML5) that are not part of the sample, the set of languages adopted by MOOCs looks closer to the column of top tier universities (ranks 1 to 33); many of the MOOCs are operated by top-tier institutions, which justifies this observation.

Table 3
Programming Language Adoption vs. Institutional Ranking First Programming Course, 2013

Language	Institutional Ranking				MOOCS
	1 to 33	34 to 66	67 to 99	100 to 134	
C	14.29	16.67	11.76	11.43	7.77
Java	60.71	46.67	35.29	42.86	15.55
C++	14.29	16.66	23.53	34.29	7.77
Objective-C	0.00	0.00	0.00	0.00	0.00
C#	3.57	0.00	0.00	0.00	7.77
PHP	3.57	0.00	0.00	0.00	4.44
Visual Basic	0.00	0.00	0.00	0.00	0.00
Python	3.57	20.00	29.41	11.43	10.00
Perl	0.00	0.00	0.00	0.00	0.00

3.2. *First Data Structures Course*

Whereas, for the sake of convenience, it is natural to use the same programming language in the first programming course and the first data structures course, there is also some rationale for using different languages. Indeed, one may argue that these two courses deal with distinct/orthogonal programming disciplines (top down versus bottom up) and distinct design approaches (functional decomposition versus data modeling). Hence we were only moderately surprised, though surprised nevertheless, when we found that a full 32 % of institutions in our sample used different languages in the first programming course and the first data structures course. Table 4 shows, side by side, the percentage of languages used for the first programming course and the first data structures course in our sample.

The difference between the distribution of languages in the first programming course and the distribution of languages in the first data structures course is sufficiently large to indicate that in fact, institutions do not automatically adopt the same language for these two courses. The following table (Table 5) further elucidates this observation by showing how institutions are distributed in terms of language adoption for the first programming course (in rows) and for the first data structures course (in columns) – where we restrict our attention to the main languages cited in section 3.1.

3.3. *First Computing Course*

Most universities we have surveyed offer a first computing course distinct from the first programming course, though it includes a significant programming component. By contrast with the first programming course, which focuses specifically on teaching a programming discipline, the first computing course introduces students to a wide range of computing topics, and is usually used as a prerequisite to subsequent CS courses, and/

Table 4
First Programming Course, versus First Data Structures Course, 2013

Language	1st Programming Course		1st Data Structures Course	
	Rank	Percentage	Rank	Percentage
Java	1	44.44	1	46.73
C++	2	19.26	2	44.60
Python	3	17.04	4	2.80
C	4	13.33	3	4.67
MatLab	5	1.48	6	0.00
C#	6	0.74	6	0.00
Haskell	6	0.74	6	0.00
PHP	6	0.74	6	0.00
JavaScript	6	0.74	6	0.00
Scheme	6	0.74	5	0.93
Racket	6	0.74	6	0.00
Ruby	7	0.00	6	0.00

Table 5
Transitions from First Programming Course to First Data Structures Course

Data Structures / Programming	C	Java	C++	Python	Java Script	C#	MAT LAB	Haskell	PHP
C	3	6	8						
Java	3	42	15	1					
C++	2	5	21	1					
Python	2	7	9	1					
Java Script	1		1						
C#	1								
MATLAB	1	1	1						
Haskell		1							
PHP	1		1						

or as an introductory computing course for non CS majors. Programming languages for the first computing course have to meet a different set of requirements from those of programming courses; they are typically chosen for their user-friendliness, their ease of learning and their ease of use, rather their relevance in industry. Hence it is not surprising that very few universities (only 2 out of our sample of 134) use the same programming language for the first computing course and the first programming course. Our data is summarized in Table 6:

Two observations are striking: First, the choice of programming language for the first computing course appears to be taken without consideration for what is in vogue in industry; second, this decision appears to be in flux, in light of the broad swings that we find in adoption figures between the 2010 data and the 2013 data. It bears pointing out that we have far less data for 2010 than we have for 2013, due to the difficulty of collecting archival data. Table 7 shows the adoption pattern as a function of institutional ranking.

3.4. *Programming Languages Course*

Whereas languages for the first computing course are chosen for their ease of use, whereas languages for the first programming course are chosen with an eye on the market, and whereas languages for the first data structures course are chosen to support data structure representation and manipulation, languages for the programming languages course are chosen for their educational value (if they embody a meaningful/ unique programming paradigm), their design attributes (if they capture meaningful design principles), or their historical significance (if they have influenced subsequent languages, or spawned many variations). Consequently, the list of languages chosen for the programming language course cover a broader range than the earlier lists, and include older languages, and less mainstream languages; also, because of the criteria used to select these languages, they tend to evolve more slowly from year to year, as they are not subject to market pressures. Our data is summarized in Table 8:

Table 6
Programming Language Adoption in Academia, 2010–2013
First Computing Course

Language	Rank 2013	Percentage 2013	Rank 2010	Percentage 2010	Evolution Percentage	Evolution Rank
MATLAB	1	26.15	12	0.00	26.15	11
Python	2	23.08	2	16.67	6.41	0
Visual Basic	3	12.31	4	5.56	6.75	1
Scratch	4	4.61	12	0.00	4.61	8
JavaScript	5	3.08	4	5.56	–2.48	–1
Alice	5	3.08	12	0.00	3.08	7
Fortran	5	3.08	4	5.56	–2.48	–1
HTML	5	3.08	4	5.56	–2.48	–1
Racket	10	1.54	12	0.00	1.54	2
Ruby	10	1.54	12	0.00	1.54	2
Scheme	10	1.54	4	5.56	–4.02	–6
Mathematica	10	1.54	12	0.00	1.54	2
C	10	1.54	4	5.56	–4.02	–6
Sway	10	1.54	12	0.00	1.54	2
Maple	10	1.54	4	5.56	–4.02	–6
Second Life	10	1.54	12	0.00	1.54	2
C++	10	1.54	3	11.11	–9.57	–7
Java	10	1.54	1	27.78	–26.24	–9
PHP	10	1.54	4	5.56	–4.02	–6
CSS	10	1.54	12	0.00	1.54	2

Table 7
Programming Language Adoption vs. Institutional Ranking
First Computing Course, 2013

Language	Institutional Ranking			
	1 to 33	34 to 66	67 to 99	100 to 134
MATLAB	27.78	38.46	50.00	26.67
Python	44.44	38.46	0.00	13.33
Visual Basic	0.00	15.38	33.33	26.67
Scratch	11.11	7.69	0.00	6.67
JavaScript	11.11	0.00	0.00	0.00
Alice	5.56	0.00	16.67	0.00
Fortran	0.00	0.00	0.00	13.33
HTML	0.00	0.00	0.00	13.33

Among the top fifteen languages, we find Prolog ranked very high, in second position, even though it is nowhere to be seen in the *Tiobe* survey, nor in the list of programming languages used in other courses; this language is used as a vehicle for discussing logic programming. Another impressive showing is the collective figure of functional

Table 8

Programming Language Adoption in Academia, 2010–2013
Programming Language Course

Language	Rank 2013	Percentage 2013	Rank 2010	Percentage 2010	Evolution Percentage	Evolution Rank
Java	1	13.02	2	11.76	1.26	1
Prolog	2	11.76	1	12.50	–0.73	–1
C++	3	10.92	3	10.29	0.63	0
Scheme	4	9.66	3	10.29	–0.63	–1
Python	5	7.56	5	8.82	–1.26	0
Haskell	6	7.14	6	6.62	0.52	0
ML	7	5.46	8	5.15	0.31	1
Lisp	8	4.20	9	3.68	0.52	1
Racket	9	3.78	17	0.73	3.05	8
Ada	10	3.36	13	2.20	1.15	3
C	10	3.36	7	5.88	–2.52	–3
OCAML	10	3.36	10	2.94	0.42	0
Perl	13	2.52	10	2.94	–0.42	–3
SML	13	2.52	10	2.94	–0.42	0
SmallTalk	15	2.10	13	2.20	–0.10	–3
Algol	16	1.26	17	0.73	0.52	–3
Scala	16	1.26	17	0.73	0.52	–2
Erlang	18	0.84	27	0.00	0.84	1
Pascal	18	0.84	13	2.20	–1.36	1
Lua	18	0.84	17	0.73	0.10	9
CAML	21	0.42	17	0.73	–0.31	–5
Coq	21	0.42	17	0.73	–0.31	–1
Simula	21	0.42	27	0.00	0.42	6
Cool	21	0.42	27	0.00	0.42	6
Modula2	21	0.42	27	0.00	0.42	6
Oz	21	0.42	27	0.00	0.42	6
Salsa	21	0.42	27	0.00	0.42	6
JavaScript	21	0.42	17	0.73	–0.31	–4
Squeak	21	0.42	17	0.73	–0.31	–4
Fortran	21	0.42	17	0.73	–0.31	–4
MatLab	31	0.00	17	0.73	–0.73	–14
Ruby	31	0.00	13	2.20	–2.20	–18

programming languages, which include Scheme (ranked 4th), Haskell (ranked 6th), ML (ranked 7th), Lisp (ranked 8th), OCAML (ranked 10th), SML (ranked 13th), and CAML (ranked 21); together, they account for a total of 32.772 %, and support the practice of functional programming. The interest of Ada (ranked 10th) is that it was developed through a worldwide competition, and that it embodies the state of the art in language design for its era (late seventies/ early eighties); it has many advanced features, that are not found in any of the languages that are currently in use. Smalltalk (ranked 15th), Simula (ranked 21st) and Modula (also ranked 21st) are languages that support modular programming by providing object oriented functionalities. As far as evolution between

2010 and 2013, the empirical data bears out our expectation that the distribution of the main languages remains relatively unchanged: the top eight languages have maintained the same rankings between 2010 and 2013, within a limit of 1.

Table 9 shows the distribution of the top twelve languages (those with a percentage of use greater than 3.00) divided according to institutional ranking.

Third tier institutions (ranked 67 to 99) use Java and C++ the least, and use Prolog, Scheme, Haskell and Lisp the most. First tier institutions use OCAML the most, and their use decreases with institutional ranking. The use of C increases monotonically from first tier to fourth tier.

4. Cross Influences

In (Ben Arfa Rabai, Bai and Mili, 2011) we had speculated on whether and to what extent language choices in academia and industry influence each other: Industries may take the lead in adopting a language, forcing universities to follow in a bid to better prepare their students for the job market; conversely, universities may take the lead in adopting a language, producing generations of students who are proficient in this language, who in time may propagate the language in industry. To test whether our data bears out one hypothesis or the other, we compute statistical correlations between language adoption in 2013 by one stakeholder (academia or industry) and language adoption in 2010 by the other stakeholder; we do so for the most common languages in our sample, namely those that have a significant following in both academia and industry in 2013 and 2010. For academic courses, we consider the first programming course, because it is the course that is most likely to be influenced by industry trends, and is most likely to influence industry trends.

Table 9
Programming Language Adoption vs. Institutional Ranking
Programming Language Course, 2013

Language	Institutional Ranking			
	1 to 33	34 to 66	67 to 99	100 to 134
Java	17.02	16.67	10.42	20.00
Prolog	12.77	14.58	16.67	13.33
C++	19.15	14.58	10.42	13.33
Scheme	8.51	10.42	20.83	8.89
Python	8.51	6.25	6.25	13.33
Haskell	6.38	10.42	12.50	4.44
ML	8.51	4.17	6.25	6.67
Lisp	4.25	4.17	20.83	8.89
Racket	4.25	4.17	6.25	4.44
Ada	0.00	6.25	4.17	2.22
C	2.13	4.17	4.17	4.44
OCAML	8.51	4.17	0.00	0.00

Table 10 shows the adoption figures for relevant languages in 2010 and 2013, for academia and industry; and Table 11 shows statistical correlations between these columns. The correlations between academia 2010 and industry 2013, as well as the correlation between industry 2010 and academia 2013 appear to be both moderate, and virtually identical; this precludes any claim of a significant influence one way or the other (which does not mean there is no influence, only that our data does not reveal any). What is also possible is that while one stakeholder influences the other, it takes more than 3 years for the effect to show.

5. Conclusion

This paper presents some factual data about the adoption of programming languages in academia and industry, for years 2013 and 2010. Among the most striking results that came out of our survey, we cite the following:

- C, C++ and Java occupy top places in the ranking of language use in industry, and in the ranking of language use in the first programming course in academia.
- Virtually all of the languages that were developed in academia with the express goal of supporting education are uniformly shunned by academic institutions, and rarely used outside their home institution.

Table 10
Cross Influences, Academia and Industry 2010–2013

Languages	Academia		Industry	
	2013	2010	2013	2010
Java	44.44	51.66	17.68	18.05
C	13.33	10.00	17.86	18.06
C++	19.26	26.66	9.71	9.71
C#	0.74	0.00	6.15	4.43
Python	17.04	12.04	4.44	4.20
Java Script	0.74	0.74	1.51	2.47
PHP	0.74	0.74	5.43	9.66
Ruby	0.00	1.66	1.97	2.22

Table 11
Correlation between Adoption Figures 2010–2013

	Academia		Industry	
	2013	2010	2013	2010
Academia 2013	1			
Academia 2010	0,977	1		
Industry 2013	0,739	0,700	1	
Industry 2010	0,693	0,662	0,966	1

- There is no measurable cross-influence of industry and academia in terms of programming language adoption, i.e. none appears to directly influence the adoption decision of the other, at least not within the three-year lead time that we have considered for our data collection.

A question that our data elicits is: why does industry keep using programming languages that date back to the late sixties/ early seventies (C), as well as variations thereof (C++, Java), at the expense of more modern languages, that represent modern ideas of language design, and feature interesting attributes such as support for modularity, exception handling, genericity, information hiding, etc. The answer to this question lies in two orthogonal premises:

- First, our investigation of software technology trends in general (Rabai *et al.*, 2011), and of programming language adoption trends in particular (YaoFei *et al.*, 2005) shows that intrinsic quality attributes of software artifacts play a minor role in adoption decisions, in favor of extrinsic factors pertaining to the circumstances in which the artifacts arose and evolved. Indeed, (YaoFei et al, 2005) analyze the correlations of eleven intrinsic factors to the adoption of languages by practicing programmers, and find that out of the eleven factors, only three have a correlation greater than 0.5, and six have a correlation less than 0.1; this is further borne out by (Meyerovitch and Rabkin, 2013) who have a section titled *Extrinsic Properties Dominate Intrinsic Ones*, in which they discuss how environmental considerations far outweigh language attributes in determining language adoption decisions. The relative insignificance of intrinsic factors in adoption decisions is actually plain to see even for the casual observer: how else can we explain that a language such as C, which was developed by two lone systems programmers to help them develop an operating system (Unix) has achieved worldwide success and has influenced so many subsequent languages, whereas a language such as Ada, which was designed by a team of experts selected through a worldwide competition, and embodied state of the art ideas about language design and modular programming, would fare so poorly as to disappear completely from the scene.
- Second, adoption of programming languages in industry is subject to many constraints that are not applicable in academia; these include, for example,
  - The cost of training programmers and analysts on a new programming language, along with possibly new programming environments and new software development processes.
  - The cost that stems from lower staff productivity and lower product quality resulting from adopting a new programming language, until such time as the software personnel gets up to speed on the new language.
  - The need to maintain staff expertise in languages that are used for legacy software, so as to support software maintenance; companies will find it much easier to manage their human resources if maintenance and new development depended on the same expertise, than if they were compartmentalized.
  - Market pressures, short-term business goals, and risk aversion limit the latitude that industry has to experiment with new languages or new paradigms, even if these could be justified in the long run.

In (Meyerovitch and Rabkin, 2013) Meyrovitch and Rabkin conduct a detailed survey of the factors that determine programming language adoption in academia and industry, and conclude that industry finds that "*existing code, existing expertise, and open source libraries are the main drivers of adoption*". Interestingly, they also find that older programmers are more resistant to adopt new languages than younger programmers; given that university students are, by definition, younger than the average industry programmer, existing expertise is a much bigger constraint in industry than it is in academia.

By contrast, the adherence of academia to such languages in the absence of the constraints above is rather puzzling, especially in light of the following observations:

- These languages, especially C, are woefully inadequate for the purposes of programmer education: they are too complex, have too many quirks, and are too implementation-dependent (expose the underlying machinery) to serve as models of computation for first year programming students. First year programming textbooks often make matters worse by shifting the focus of the course from teaching a discipline of programming using a programming language to teaching the programming language instead, including all its obscure, esoteric, quirky details.
- Academia has the latitude to lead: The debate of whether academic trends should lead or follow industrial trends applies to programming language choice as much as to any technology trend. Yet, the fact that industrial developers decide on what programming language to use based, at least in part, on their education (according to [Meyerovitch and Rabkin, 2013]) means that academic choices do affect industrial choices.
- Academia has the means to lead: Because developers learn new languages frequently and rapidly, a student can learn to program in one language and later practice software development in another language with minimal cost/ effort/ disruption; hence academia does not have to select languages according to industry choices, but ought to define and follow its own selection criteria. This supports the view that academia should select programming languages according to purely education criteria, rather than the myopic concern of preparing students to be immediately operational on the job.
- Academia has the incentive to lead: It is all the more critical for academia to follow its own selection criteria that they appear to differ significantly from industrial criteria: an ideal language for education is one that favors simplicity over computing power, and supports language-enforced correctness rather than expressive constructs; yet Meyerovitch and Rabkin find that industrial developers use the exact opposite criteria.
- Academia has ample opportunity to lead: many dedicated educators and scholars have gone to the effort and trouble of creating small programming languages dedicated specifically for programmer education. The include: Alice [Dann *et al.*, 2012]; BlueJ [Koelling *et al.*, 2003]; Haskell [Hudak, 2000]; Racket [Felleisen, 2000]; Ruby [Flanagan and Matsumoto, 2008]; Scratch [McManus, 2013], Squeak [Ducasse, 2005]; Oz [Van Roy and Haridi, 2004]. Unfortunately, most of these languages are barely used for the purpose of programmer education.

In conclusion, we argue that academic decision makers ought to take the lead in setting the agenda of programmer education, through the judicious selection of programming languages that are designed for this purpose, that help the student develop a sound discipline of programming, and that ultimately help raise the level of software engineering education and the level of software practice. Understandably, the ACM/IEEE taskforce on computing curricula stays clear of making any recommendations on the choice of programming languages, because it views them as means to an educational end, rather than an end; as a result, it reasons exclusively in terms of programming paradigms, and makes recommendations regarding object oriented programming, functional programming, reactive programming, logic programming, and concurrency and parallelism, leaving academic decision-makers all the latitude they need to choose the languages that best convey these paradigms.

Acknowledgements

The authors acknowledge the assistance of Kevin Lin, Rutgers University and Islem Souid, ISG, Tunis University in collecting and compiling the data. They are also very grateful to the anonymous reviewers for their valuable comments and suggestions, which have greatly improved the contents of the paper.

References

ACM (2013). *Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Programs in Computer Science. December 20, 2013*. ACM/IEEE Computer Society.

Agresti, A. (2002). *Categorical Data Analysis* (2d edition). Wiley Interscience, Florida.

Bai, Yanzhi., Mili, A. (2007). Monitoring software technology evolution, one trend at a time. In: *SEDE 2007*, 349–355.

Barry, B. (1981). *Software Engineering Economics*. Prentice Hall.

Ben Arfa Rabai, L., Bai, Y.Z., and Mili, A. (2009). Modeling the evolution of software engineering trends – a bottom up approach. In: *ICSOFT 2009*, 1, 47–54.

Ben Arfa Rabai, L., Bai, Y.Z., and Mili, A. (2011). A quantitative model for software engineering trends. I*nformation Sciences*, 181(22), 4993–5009.

Clements, PC. (2006). Future trends of software technology and applications: software architecture. In: *Thirtieth Annual Conference on Computer Software and Applications.*

Dann, W., Cooper, S., Pausch, R. (2012). *Learning to Program with Alice.* Prentice-Hall.

Ducasse, S. (2005). *Squeak: Learn Programming with Robots. TIA: Technology in Action*. Apress.

Felleisen, M. *et al.* (2001). *How to Design Programs: An Introduction to Programming and Computing*. MIT Press.

Flanagan, D., Matsumoto, Y. (2008) *The Ruby Programming Language*. O'Reilly.

Geoffrey, A.M. (2002). *Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers*. Harper Collins Publishers.

Geoffrey, A.M. (2002a). *Living on the Fault Line: Managing for Shareholder Value in any Economy*. Harper Collins Publishers.

Hudak, P. (2000). *The Haskell School of Expression: Learning Functional Programming through Multimedia*. Cambridge University Press, New York.

Jerry, L. (2000). Statistics in reliability. *Journal of the American Statistical Association*, 451(95), 989–992.

John C.K. (2006). Future trends of software technology and applications: model based development. In: *Thirtieth Annual Conference on Computer Software and Applications*.

Kölling, M., Quig, B., Patterson, A., Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education* (Special issue on Learning and Teaching Object Technology), 13(4), 249–268.

McManus, S. (2013). Scratch programming in easy steps: covers Versions 2.0 and 1.4. In: *Easy Steps Limited*.

Meyerovitch, L.A., Rabkin, A. (2013). Empirical analysis of programming language adoption. In: *OOPSLA'13, Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, New York, 1–18.

Richard, A,J. (2002). *Applied Multivariate Statistical Analysis*. Prentice Hall.

Roy, V., Haridi, P., Haridi, S. (2004). *Concepts, Techniques and Models of Computer Programming*. MIT Press.

Stephen, S.Y. (2006). Future trends of software technology and applications. In: T*hirtieth Annual Conference on Computer Software and Applications*.

Warren, H. (2006). Future trends of software technology and applications: the phone is the computer. In: *Thirtieth Annual Conference on Computer Software and Applications*.

Yaofei, C., Dios, R., Mili, A., Wu, L., Wang, K. (2005). An empirical study of programming language trends. *IEEE Software*, 22(3), 72–78.

Yi, P., Li, F., Mili, A. (2007). Modeling the evolution of operating systems: an empirical study. *Journal of Systems and Software*, 80(1): 1–15.

L. Ben Arfa Rabai is a University associate professor in the Department of Computer Science at the Tunis University in the Higher Institute of Management (ISG). She received the computer science Engineering diploma in 1989 from the sciences faculty of Tunis and the PhD, from the sciences faculty of Tunis in 1992. Her research interest includes software engineering trends quantification, quality assessment in education and e-learning, and security measurement and quantification. She has published in information sciences Journal, Computers in Human Behavior Journal, IEEE Technology and Engineering Education magazine... She has participated in several international conferences covering topics related to the computer science, E-learning, quality assessment in education, cyber security, quantifying security...

B. Cohen is the associate dean of the College of Computing Sciences at the New Jersey Institute of Technology (NJIT). He coordinates and is one of the teachers of the first programming class for computing majors at NJIT (taught in Python). His primary field of research is bioinformatics.

A. Mili holds a PhD in computer science from the University of Illinois in IL, USA and a doctort es-siences d'etat from the Joseph Fourier University of Grenoble, France. He is on the faculty of NJIT in Newark, NJ, USA; his research interests are in software engineering and software engineering education.

Programavimo kalbų naudojimas JAV aukštosiose mokyklose ir įmonėse

Latifa BEN ARFA RABAI, Barry COHEN, Ali MILI

Kaip natūralios kalbos veikia ir formuoja mūsų mąstymą, taip ir programavimo kalbos turi didelį poveikį tam, kaip programuotojas analizuoja problemą ir formuluoja sprendimą. Tikėtina, kad pirmas programavimo kursas lemia studento požiūrį į programos kūrimą, programos analizę ir programavimo metodologiją, todėl programavimo kalbos pasirinkimas pirmam programavimo kursui yra labai svarbus. Šiame straipsnyje pristatomas tyrimas, kuris leidžia palyginti, kokių programavimo kalbų mokoma Jungtinių Amerikos Valstijų aukštosiose mokyklose ir kokios vartojamos įmonėse.