Implementing IT0/CS0 with Scratch, App Inventor for Android, and Lego Mindstorms

Suleyman Uludag
University of Michigan - Flint
Dept of Comp Sci, Eng & Phy
uludag@umflint.edu

Murat Karakus
University of Michigan - Flint
Dept of Comp Sci, Eng & Phy
mkarakus@umflint.edu

Stephen W. Turner
University of Michigan - Flint
Dept of Comp Sci, Eng & Phy
swturner@umflint.edu

ABSTRACT

The trend of declining enrollment and interest in computing fields, combined with increased demand from the industry, challenges instructors to come up with new, fresh and appealing methodologies to attract and retain students. Further, with the diffusion of information and computing technologies into almost all fields of study, introductory computing courses for non-majors need approaches that motivate students to feel comfortable with the life-long learning of computing concepts and tools. The goal of this paper is to summarize our teaching experience blending the aforementioned two needs into one course that may be considered as a type of CS0/IT0 course. With the pedagogical underpinnings stemming from constructionist learning and contextualized computing education, we present our motivation and the details of a course that uses the Scratch programming language, App Inventor for Android, and Lego Mindstorms robotics.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: Computer Science Education; D.2.6 [Programming Environments]: Graphical environments; D.3.2 [Programming Languages]: Language Classifications—Specialized application languages

General Terms
Design, languages

Keywords
CS0, IT0, computer science for non-majors, introductory programming, novice programming environments, information literacy, IT education, CS education,

1. INTRODUCTION AND MOTIVATION

In recent years, student interest in information and computing technology (ICT) has declined to the extent that computer science departments face enrollment crises. Student motivation to enroll in computing fields is in decline, and the problem is not helped when early computing courses are found to be uninteresting by and/or not involving of the students. In contrast, demand for competent, trained information technology (IT) professionals continues to rise. The strategic position of IT at the crossroads of Science, Technology, Engineering, and Mathematics (STEM) education is equally notable. The National Science Foundation (NSF) Strategic Plan 2006-2011 [6] calls for building excellence in undergraduate STEM education in order to maintain the U.S. position at the forefront of discovery and innovation. It underlines the aging science and engineering workforce coupled with continuing strong demand for STEM-related fields. The National Research Council (NRC) [2] notes that providing engaging laboratory exercises is one of the challenges of effective undergraduate education in STEM fields [36].

![Figure 1: (a) Bachelor’s degrees awarded in 1000s in CS & CE from 1995 to 2009 [85]. (b) Computer and information sciences degrees conferred from 1971 to 2009 as reported by U.S. Dept of Ed., Higher Education General Information Survey (HEGIS) [10].](image)

In this context, this paper presents the description of CSC101, an introductory (IT0/CS0) course that aims to attack the declining enrollment problem while simultaneously providing more engaging laboratory exercises to potential undergraduate students in computing fields. The course utilizes extensive laboratory exercises that remain relevant to students’ real-world experiences, provide interesting and “cool” exercises, provides a gentle introduction to programming through graphical drag-and-drop programming languages such as Scratch, and uses familiar technology, such as the Android phone and LEGO Mindstorms robots.

1.1 The CS Enrollment Crisis

While there is a need to bring more students to STEM in general, and computing in particular, there has been an enrollment crisis in computing for the past decade or so [24].
Declining support both from industry and government, as well as its ever-deteriorating reputation and status among high-school students, have presented formidable challenges in attracting new or retaining existing students in computing fields [85]. Figure 1(a) shows the computer science and computer engineering bachelor’s degree production from 1993 to 2009 as reported by Computing Research Association (CRA) Taulbee Survey. Figure 1(b) is a plot of the computer and information sciences bachelor’s degrees conferred by degree-granting institutions in the US from 1971 to 2009 based on HEGIS survey [10]. Both graphs clearly depict an overall decline in interest for computing fields.

At the same time, there appear to be some positive signals emerging recently [68]. There have been modest increases, both in computer science majors [85] and freshmen intentions, as published by National Science Board, Science and Engineering indicators [9], graphically shown in Figure 2. Yet, meeting the projected demand for graduates of computing-related programs (see for example [11]) looks likely to take more sustained effort. Another interesting observation concerning freshmen intentions is shown in Table 1. It reveals that even though the percentage of students intending to major in STEM fields has remained relatively static from 1993 to 2009 (varying from 33.3% to 34.7%), the number of freshmen displaying CS intentions has fluctuated widely over the years and has trended significantly lower since 2001. One way to interpret this observation is that more efforts need to be made to recast the image of CS as more appealing, even to STEM-inclined students. This is one aspect of our motivation for proposing our CSC101 course as we explain later.

1.2 Motivation for non-CS students

For the more general student body beyond STEM fields, in order to participate effectively and confidently, and contribute productively, in today’s digital world, undergraduate students must be proficient with Information and Computing Technologies (ICT), regardless of their areas of study [70]. The traditional term of computer literacy for exposing students to ICT has been replaced by the NRC by Information Technology (IT) fluency [55]. The former only encompasses skills that are likely to become obsolete with changing technology, whereas the latter includes not only the skills but also the foundational concepts and intellectual capabilities to adapt to changing technologies [4]. This theoretical framework, provided by NRC in [4], asserts that as students become more comfortable with information technology, they are able to express themselves creatively, to reveal knowledge in novel forms, and to synthesize new information. NRC has adopted the term FITness (Fluency in IT) to denote familiarity and comfort in the understanding and use of ICT. Even though students entering college today come with unprecedented levels of exposure to different manifestations of ICT in their daily lives, the growing concern and surprising reality, as concluded in many studies (e.g. [47] and [50]), is that FITness levels have been declining. While ICT holds the promise of increased productivity, this promise may not be realized due to the limited abilities of professionals to make efficient use of ICT [29]. A key reason is that FITness is dynamic and requires lifelong learning as technology changes.

Another motivation for our CSC101 course is very nicely summarized in [35] as follows:

In traditional programming-first introductory courses, generic problems are often used to introduce foundational programming skills and concepts that will become important to students as they progress to more advanced courses and attempts to solve more complex problems. Unfortunately, since the relevance of basic programming skills is often revealed in later courses, non-majors who are compelled or decide to take the introductory courses seldom experience the gratification of using their programming skills in a personally meaningful context. By failing to connect programming and CS concepts with students’ diverse interests and backgrounds, traditional introductory courses fail to motivate many students and may even discourage them from pursuing further CS learning.

All of the above narrative and evidence have led us to our CSC101, a CS0/IT0 course. Three important components of the course for Contextualized Computing Education as noted above will be (1) the Scratch programming language from MIT, (2) the Android operating system for smartphones, and (3) the Lego Mindstorms robotics programming platform. The latter two are especially what most, if not all, students are already familiar with. They enable us to relate the fundamental computing topics to tangible real-life phenomena.

The rest of the paper is organized as follows: Section 2 gives a comprehensive discussion of the related work. Pedagogical foundation of our course is presented in Section 3. The details of our CSC101 course are given in Section 4. Section 5 concludes the paper.

2. RELATED WORK

As ACM Turing Award Winner Perlis puts it, everyone should learn to program as part of a liberal education [71]. The terminology for the introductory computing courses dates back to ACM 1978 Curriculum Recommendations [16] where a two-course sequence of CS1 and CS2 were first stated. Even though so much has changed since then in terms of technology, philosophy and structure, the CS1/CS2

![Figure 2: Percentage of entering freshmen intending to major in computer science. Source: NSF National Science Board, Science and Engineering indicators [9].](image-url)
duo is widely used in the literature to refer to the introductory computing courses for majors. Significant growth in demand for computing graduates and the diffusion of computing fundamentals into most of the other fields of study have made apparent that an additional preparatory course to CS1/CS2 sequence is warranted. This raised the issue of preparing majors into CS1/CS2 sequence with additional content and the term coined for this course has become CS0. However, no ACM Curricula Recommendations has included any definitions or models for CS0.

Even though the literature is full of references to CS0, CS1, and CS2 terms, there is no universally agreed upon delineation of these three courses in terms of their coverage. After ACM Computing Curricula 2001, the distinction between CS0 and CS1 has become even fuzzier. Further, some issues of disagreement for CS0 include how much, if any, programming to include in CS0, breadth-first versus depth-first approach, separate CS0 for computing majors [57], targeted majors [44, 78] and non-majors [31, 43, 23]. Benefits of CS0/IT0 courses are discussed in [49, 59] for majors and in [27, 30] for the non-majors. A good summary of CS0 history from 1968 to the 1990s is given in [58].

Starting from the 2005 recommendations, the ACM has separated Information Technology (IT) from the Computer Science degree. IT2008 [8] defines IT as follows: IT, as an academic discipline, is concerned with issues related to advocating for users and meeting their needs within an organizational and societal context through the selection, creation, application, integration and administration of computing technologies. One of the 5 pillars of IT, as given in IT2008, is programming. Thus, a CS0-type course for IT majors is also needed. With this, we consider our CSC101 course as a CS0/IT0-type course.

### 2.1 IT0/CS0 objectives

In the literature, the objective of IT0/CS0 courses fall into one or more of the following categories:

1. A service course that teaches some practical computing knowledge and/or meets a general education requirement at the institution [21],
2. To improve retention and success in subsequent CS courses, particularly CS1 [26, 22],
3. To provide a broad overview of the field, for prospective majors and non-majors alike. [20],
4. To increase diversity of students to the major [15],
5. To prepare students to analyze and address the current ethical, social and legal implications of the growing ubiquity of and increased reliance on technology [74],

### 2.2 Choice of programming language

Even though some CS0/IT0 courses are offered without any programming, the majority have some sort of programming in their content. A survey and taxonomy of languages and environments used to make programming easier to learn for novice programmers is presented in [51]. Another good survey and classification of different programming languages used in introductory programming courses is given in [69]. A very recent brief history of learning to program is given in [33]. Guzdial in [41] classifies entry-level programming into LOGO-based [1], graphical rule-based (aka microworld) and pedagogical (by means of the programming environment) approaches. Scratch is an example of the former. Alice is an example that uses rule-based microworlds, and Greenfoot is an example of the latter [33]. These three are compared and contrasted by their creators in [81]. The use of robotics (such as Karel [66] or Lego Mindstorms) in introductory programming courses are also reported in the literature, such as in [83, 67, 68].

A brief snapshot of the variety of programming languages used in IT0/CS0 courses is shown in Table 2 as collected from the published literature. Primary, secondary and tertiary designations indicate the order of the exposure of multiple languages in class.

### 3. PEDAGOGICAL PILLARS

The first pedagogical pillar of our CSC101 is on the Constructionist learning theory [65, 45, 64], by Papert, which is based on the constructivist theories of psychology by Jean Piaget. As noted in [13], Papert considers knowledge and learning as firmly grounded in contexts, and shaped by their active and engaged uses. In his landmark book [63], Papert describes his involvement with toy automobiles at the age of two and how it drove his curiosity and motivation in learning mathematics during his schooling later. He used Piaget’s theory and his own learning experience to develop his theory. His use of gear as transitional object is similar to our use of smart phones and Lego robotics. Constructionist learning is usually, albeit partially, mapped to the learn-by-making or learn-by-doing motto, as noted by the originator in [65]. From constructivist theories of psychology, we take a view of learning as a reconstruction rather than as a transmission of knowledge. Then we extend the idea of manipulative materials to the idea that learning is most effective when

<table>
<thead>
<tr>
<th>Language</th>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>[67, 83, 78, 28, 60, 49, 25]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td>[67, 46, 39]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lego Mindstorms</td>
<td>[21]</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>Visual Basic</td>
<td>[21]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scratch</td>
<td>[56, 76, 25]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>[44, 80, 14]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VPython</td>
<td>[44]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MatPlotLib</td>
<td>[44]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>[56]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash</td>
<td>[78]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GameMaker</td>
<td>[62, 75]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaScript</td>
<td>[57, 20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greenfoot</td>
<td>[31]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>[57]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td>[62]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robotics</td>
<td>[68, 67]</td>
<td>[83]</td>
<td></td>
</tr>
<tr>
<td>JPie</td>
<td>[38]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATLAB</td>
<td>[54]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StarLogo TNG</td>
<td>[82]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>karel</td>
<td>[72]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drizzle</td>
<td>[32]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual Logic</td>
<td>[40]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sibelius</td>
<td>[78]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jeroo</td>
<td>[74]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OOPIC</td>
<td>[74]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Variety of languages used in introductory programming classes with the classification based on the order in which they are introduced their respective classes.
part of an activity the learner experiences as constructing a meaningful product. Second pedagogical foundation is active learning as coined by Bonwell and Eison in [19]. Another pedagogical pillar is grounded in the Project-based learning methodology by Blumenfeld [18]. We believe that fostering student engagement and longer lasting learning are achieved by combining student interest with a variety of challenging, authentic and real-world problem-solving tasks. Another objective is to shift the students from the state of passive learners to the level of active learning by engaging them with intensive lab exercises. The intensive labs facilitate the students’ learning at comprehension, application, synthesis and evaluation levels of cognition based on the classical taxonomy of cognitive levels by Bloom [17]. As a result of the learn-by-doing method, the learning outcome for the students is exposure to the real-world problems and acquisition of skills sought by the current, tight and highly competitive job market, especially under the positive job outlook for computing graduates, as outlined in [11].

The NRC also encourages eliciting and working with students’ preexisting knowledge in [34, 61]. The Report on Reports, by Project Kaleidoscope [5, 7] emphasizes connecting learning to the world beyond the classroom in order to highlight the relevance. It highlights the significance of applications of class material to the experience of students. The positive utility of the role of prior knowledge in learning and bringing real-world problems into classrooms is pointed out in [48]. The NSF report Shaping the Future, Volume II [3] (as noted in [5]), indicates that the incorporation of knowledge of what STEM practitioners do in their daily jobs is very important to the education of students in these fields, as relevant knowledge accelerates learning by facilitating a better organization of the information. Another observation, from offering an introductory computing course like ours, is that such courses must keep the students engaged by showing them how their everyday activities, both online and offline, are influenced by ICT [53]. Finally, Georgia Institute of Technology’s offering [42], and conclusion from years of offering similar courses, indicate that CS0-like courses must be taught in terms of practical domains or contexts that students recognize as important. They call this approach as Contextualized Computing Education, which is what our philosophy in designing and developing CSC101 in this manner with labs is guided by.

4. OUR CS0/IT0 COURSE

The goal of our CS0/IT0 (CSC101) is three-fold:

1. Educate students with varied backgrounds. Our school has added a new technology general education requirement for all entering students as of Fall of 2009. Therefore, many non-majors will be taking our course.

2. Educate ICT students who lack preparation. Some computer science and computer information systems majors lack the proper background to begin their studies. Therefore, many majors will be taking this course.

3. Educate high-school students who may enroll at our university. High-school students in a pre-engineering program, offered in conjunction with a local school district, will also be taking this course.

### 4.1 App Inventor for Android

The Android OS for smart phones has recently positioned itself to dominate the market. Figure 3(a) shows the market share for smart phones as of March 2011 while Figure 3(b) gives a better view of the trends by means of the market share of the recent smart phone activations: Half of those surveyed had chosen the Android device. According to the Nielsen surveys, the market penetration of Android is even bigger among the younger consumers. That means, more than half of the college students will have used an Android device before taking our CSC101. Android opensource runs a modified Linux kernel with Java-based applications through a Dalvik virtual machine framework. In December of 2010, Google released Google App Inventor for Android (AIA) for developing applications by using a graphical interface that is very similar to MIT’s Scratch. The goal is to give a head-start on Android platform application development for non-programmers. This is accomplished using the more intuitive drag-and-drop features commonly available within web browsers. A number of colleges have already started using AIA in different courses for non-majors as reported in a recent panel [79] and a workshop [12], but the details are not published. The approach closest to our work is a course offered at the University of San Francisco [84], in which Android phones are used as robots by the students to develop applications. Our approach uses AIA to develop real applications for the Android, which gives the students a greater sense of involvement, due to their ability to create applications they might use outside of the classroom. Additionally, we make use of Lego Mindstorms to take advantage of the use of robot kits [68, 66] with which many students will already be familiar.

#### 4.2 AIA and Mindstorms

In order to deal with such a challenging and somewhat heterogeneous student mix, we have decided, based on the pedagogical justifications and motivations as outlined in Sections 1 and 3, to make use of Scratch, Google AIA and Lego Mindstorms. Scratch serves as a soft introductory programming language. Then, we aim at contextualizing the topics to raise the interest and motivation of the students by means of assignments using AIA for Android apps. We hypothesize with theoretical underpinnings given in Section 3 that relating the curriculum to the students’ immediate life in some meaningful ways we can pave the way for a more successful education and career for majors and facilitate non-majors life-long learning processes to make more effective use of technology in their educational, personal and professional lives.
We have piloted the first offering of our course in the Winter semester of 2011. We are still in the process of developing more labs using AIA. We have just obtained some internal funding for more smart phones and LEGO Mindstorms robotic kits.

### 4.2.1 AIA Lab Exercise

In an introductory lab exercise with AIA, we relate to another popular topic among student, i.e. school athletics. Students develop an audio player app to play a marching song (*Hail to the Victors* for our case) with some university logo as shown in Figure 4. Then, they use USB connection to an Android phone to upload the app to the phone. There were many who had their own Android phones and who enthusiastically used their own phones to upload the app they have created.

### 4.2.2 AIA and Lego Mindstorms Exercise

Another interesting part of AIA is its support for the Lego Mindstorms robotics interface. This opens many more interesting venues with which to engage students in the class content, and it potentially makes a more lasting impact in their careers. Figure 5 shows a screenshot of a Lab exercise where an Android phone is used to get connected to a Lego Mindstorms robotics device and start/stop its servomotor. Figure 6 shows the graphical, drag-and-drop type code of the aforementioned app in block editor in a browser.

### 4.3 Assessment

With the addition of the new equipment, we will be conducting surveys to measure the effectiveness of the labs in this Summer term. We are also planning to make use of 1-minute, anonymous feedback papers from the students to assess how well the class in general and AIA and robotics labs in specific are received by the students.

### 4.4 Future Work

In future offerings of our course, we are planning to include open-ended group projects to give students more opportunities to develop other relevant Android labs that are relevant to their majors, professional careers and/or personal lives.

5. CONCLUSION

In this paper, we have summarized our teaching experience in CSC 101, a IT0/CS0 course. We have elaborated on the objectives of the course for majors and non-majors, as well as the pedagogical motivation and foundations. An extensive survey of related work from the literature is also provided. To the best of the authors’ knowledge, our combination of Scratch, App Inventor for Android, and Lego Mindstorms robotics in an IT0/CS0 course is unique and holds some promise as an appealing course for both majors and non-majors. The main thrust of our approach is on contextualizing the curriculum delivery, in terms of phenomena that the students are previously familiar with, and leveraging it to raise their interest and involvement in the subject matter. We have piloted the course for the first time in the
Winter semester of 2011, and we are in the process of developing more lab exercises for future offerings, as well as setting up a formal assessment of its effectiveness.

6. REFERENCES


