# ‘PROGRAM BY DESIGN’ AS PART OF A NEW MAJOR IN COMMUNITY INFORMATION SYSTEMS

Levy, Dalit, Zefat College of Education, Zefat, ISRAEL, dalitl@zefat.ac.il

## Abstract

*During the last decade, social information systems have gained significant popularity by providing the individual both access to knowledge and powerful means of communication. At the same time, social information systems have emerged as an empowering force for communities, organizations, and businesses. In parallel with these changes, a new interdisciplinary area of study has evolved, arguing that the social and the technological mutually shape each other. New academic programs have been proposed around the globe, aiming at establishing a framework within which students gain experience in the socio-technical process of designing information systems in business, libraries, health, government, education and beyond. These new interdisciplinary programs often regard computer science (CS) as one of their supporting pillars and therefore include some core CS courses, aiming at educating broad-minded practitioners rather than expert programmers in the field of information systems. This paper presents some thoughts on incorporating CS education in academic programs intended for non-CS majors and proposes an approach called ‘Program by Design’ for the first CS course in a new undergraduate program in community information systems at Zefat Academic College.*

*Keywords: Community Informatics, Design Recipes, Introductory Programming.*

# 1 INTRODUCTION

In the last two decades we have witnessed an invasion of homes, workplaces, public spaces, and both local and global organizations by information technology tools and systems. The advents of the World Wide Web, wireless communications, and miniaturized computing technology have considerably expanded this invasion into mobile devices and remote communities. The widespread everyday use of computers and information systems reflects a shift in conceptualizing the technology as more social than it was perceived before: 'the computer started as a totalitarian tool, but has now also been embraced as a social tool' (Kizza, 2003, p. 158). More recently, over the last several years, social information systems have gained significant popularity. Social networking sites, social sharing and tagging systems and social media attract several million users a day all over the globe. These kinds of information systems provide their individual users with increased social presence, much broader access to information and knowledge, and powerful means of communication. At the same time, social information systems emerge as an empowering force for both local and global communities, organizations, and businesses.

Following these radical changes (Carr, 2008), a new interdisciplinary area of study has evolved, arguing that the social and the technological mutually shape each other. Studies in this area touch several different fields, including CS, information systems, information science, and some social sciences (Kling, 1999). By examining the social aspects of computing, the fields of Social Informatics and Community Informatics aim to ensure that technical research agendas and information systems designs are relevant to the lives of people and organizations. Community Informatics aims further at empowering communities through the use of technology, especially those groups who are excluded from the mainstream communication systems (Gurstein, 2008).

The increasing interest among different communities of practice in integrating human and social considerations into traditional information systems curricula has led to the development of new academic programs around the globe. These are aimed at establishing a framework within which students develop analytical skills to identify and evaluate the social consequences of ICT-based systems, and gain experience in the socio-technical process of designing information systems in business, libraries, health, government, education and beyond. The latest model curricula for undergraduate degrees in information systems recommended in 2009 by a joint task force of the ACM and the AIS also supports reaching beyond the schools of management and business. While information systems curricula have been traditionally targeted to business schools, the current task force believes that the discipline provides expertise that is critically important for an increasing number of domains (Topi et. Al., 2010).

Currently, most undergraduate programs in information systems (IS) around the globe operate either as part of the faculty of engineering or within the context of the business environment and related activities. This is also the case in Israel, where the new community–oriented IS program here reported has only recently opened. Although other IS programs in the country might offer one or more courses dealing specifically with social aspects of IS and ICT, these courses are considered marginal.

The new undergraduate program in Community Information Systems has been developed in Zefat Academic College in light of the global trends discussed above and, in addition, as a response to the educational gap identified between various population sectors in Israel (Levy, 2010). The program seeks to be sensitive to the increasing demand for higher education of the population in a rural part of the country, by considering the multi-cultural facets of businesses, organizations, and communities, and to empower these developing communities by using advanced technologies and information systems. The curriculum combines theory and practice while emphasizing subjects that are relevant to the workforce and the organizations surrounding the college, thus creating “Practice of Relevance” for its students (Benbasat & Zmud, 1999). Section 2 further describes the program and its curriculum as an intersection of disciplines.

The discipline of CS is often perceived by IS programs as one of their supporting pillars. CS methods and ideas, which are at the root of ICT innovations and information systems design processes, are

thought to have the potential to contribute to a greater understanding of those creations. Moreover, advancing students' understanding of computing has been thought of as critical to developing the needed workforce for the 21st century (The College Board, 2012). Therefore, students in IS programs in general, and in an interdisciplinary program such as Community IS in particular, should study fundamental CS courses in order to acquire the needed broad foundation in computing and consider its breadth of application. The first part of Section 3 therefore proposes a CS track tailored for non-CS majors in order to give them an understanding of the principles and practices of computing as well as its potential for transforming the world (Harvey and Garcia, 2010). The 'Program by Design' approach (Bloch et. al., 2010) found especially appropriate for the first course in this CS track is detailed in the second part of Section 3. As the new Community IS program has only recently opened, the last section of this paper offers early remarks on implementing 'Program by Design' at Zefat Academic College.

# 2 AN INTERDISCIPLINARY CURRICULUM IN COMMUNITY IS

## 2.1 Background

Zefat Academic College's undergraduate program in Community IS has been approved by the Israeli national council for higher education at the end of 2010 and the first students have started their course of study in the fall term of 2011. The program's main assumption is that the revolutionary development of information technologies in general and of information systems in particular, changes organizational structure and organizational practices. Therefore, the workforce as a whole will benefit from acquiring basic academic knowledge in information systems, not only the engineers or those in managerial positions (The College Board, 2012). The notion of "community" in Community Information Systems is broad, including business communities as well as non-profit organizations, global or local organizations, public communities, cultural communities, and rural communities.

Imagining information system as a junction connecting (i) human users, (ii) supporting technologies, and (iii) organizational environment, the new curriculum includes (i) psychological and sociological aspects, (ii) information technologies and systems, and (iii) issues of organizational culture (Levy, 2010). This interdisciplinary approach can be seen also in Community Informatics (CI) undergraduate and graduate programs in Canada, USA, Australia, Italy, and many more (Stillman and Linger, 2009) and in the emerging field of ICT and Development. The interdisciplinary nature of these fields calls for creating interdisciplinary academic programs that will support educating "more capable learners, more innovative teachers, more creative thinkers, more effective leaders and more engaged global citizens" (Bennett and Sterling, 2011, p.626). Such programs enable students' specialization both in the technical and the social aspects of information systems. They also expose learners to the breadth of human arenas and communities supported by information systems like public health, economic development, education, and many more.

## 2.2 Program Structure

The three years curriculum is structured around "Information Technologies and Systems" as a core area of study. Required core courses provide half of the program credits - 60 out of 120 credits, where one credit typically equals fifteen class hours.

| Area of study | | Year 1 | Year 2 | Year 3 | Sum of credits |
|---|---|---|---|---|---|
| Core: Information Technologies and Systems | Required credits | 21 | 22 | 17 | 60 |
| | Elective credits | | 4 | 6 | 10 |
| **Sum of core credits** | | **21** | **26** | **23** | **70** |
| Areas of specialization: (a) The Knowledge Society (b) Information in Organizations | Required credits in area (a) | 10 | 6 | 4 | 20 |
| | Required credits in area (b) | 10 | 6 | 4 | 20 |
| | Elective (a) OR (b) | | 4 | 6 | 10 |

| | | | | |
|---|---|---|---|---|
| **Sum of credits in areas of specialization** | **20** | **16** | **14** | **50** |

*Table 1. Distribution of Courses in Different Areas of Study*

Additional ten credits are offered through elective courses in the core area of study. The rest of the credits are equally divided between two supporting areas of study: (a) "The Knowledge Society" and (b) "Information in Organizations". As can be seen in Table 1, students are exposed to the interdisciplinary nature of the program from Year 1. In the second and third years, students elect either area (a) or (b) as an area of specialization.

*The core* of the curriculum contains required foundation courses in three tracks: ICT, IS, and CS. The latter track is further described in Section 3 below. *Area (a)* "The Knowledge Society" includes required courses like digital culture, sociology of the internet, and evaluating digital communities. *Area (b)* "Information in Organizations" includes required courses like knowledge management and organizational behaviour. As Table 1 demonstrates, while students elect only one area of specialization, they also take some required courses in the other area. That way the program provides the multidisciplinary knowledge required for entry-level positions in a wide spectrum of organizations.

The structure of the Community Information Systems program separates the core of the curriculum from the electives with the intent of supporting the creation of a sound knowledge base of information systems, at a level appropriate for undergraduate students. At the same time, the courses in both areas of specializations mark the social, cultural, organizational, and human aspects as central to the knowledge base of information systems, thus can support the conceptual development of a multi-faceted body of knowledge by those who will study according to the new curriculum in Community Information Systems. As a result of such integrated curriculum, we vision graduates who are both information-technology-oriented and social-oriented, and thus can empower the communities within they live and work.

# 3 THE COMPUTER SCIENCE TRACK

## 3.1 Computer Science Education for non-CS Majors

As has been noted above, the new curriculum builds upon three parallel core tracks, one of which focuses on computing principles and practices. This computer science track has been designed to suit the characteristics of the students in the program on the one hand, and to make computer science discourse and culture understandable by those students on the other hand. In other words, it is this track's goal to enable the graduates' participation in the professional discourse used among programmers, software designers, and software development teams. Towards this goal, four successive courses are offered at the main computer science track, as is illustrated in Figure 1.

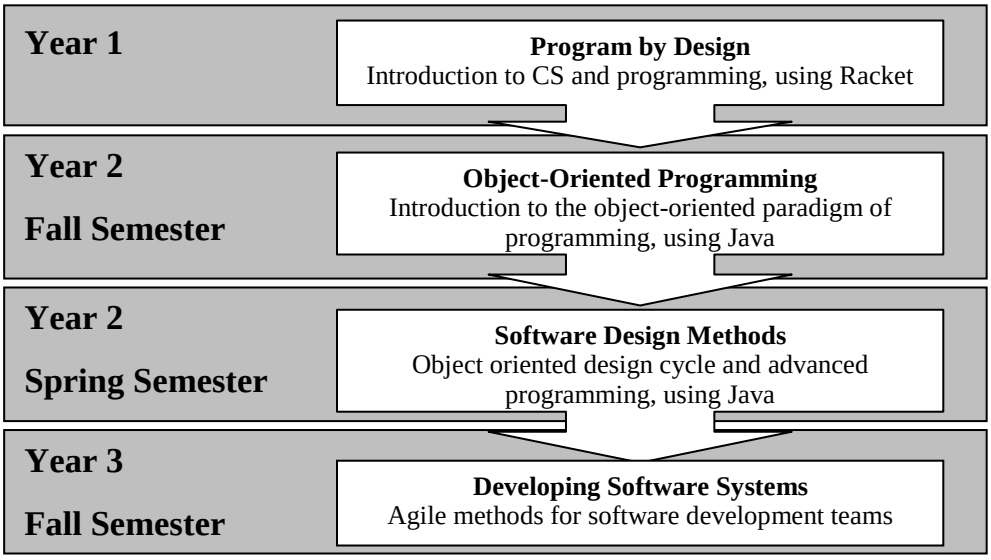

*Figure 1. The Flow of the Required Courses in the Computer Science Track*

Together with few more CS courses (required courses such as data structures and electives such as applications programming), the CS track integrates theory and practice while engaging students in the full cycle of software design. Since the students are not majoring in CS, and thus might not have an advanced mathematical and scientific background, the CS track starts with an educational approach called 'Program by Design' (Bloch and Prolux, 2007; Bloch et al., 2010) which is based on Racket programming (Flatt and PLT, 2010). As is detailed in the next section, this approach seems especially appropriate for a first programming course for non-CS majors, by focusing on developing design practices and desired programming habits from day one, by immersing learners into the profession's jargon, and by experiencing real coding with pictures instead of the conventional arithmetic jumpstart (Bloch, 2010).

### 3.2 The First Course in the Computer Science Track

'Program by Design' (PBD) is a longitudinal educational effort whose mission is 'to turn computing and programming into an indispensable part of the liberal arts curriculum'[1] both in high schools and undergraduate colleges. Led by Matthias Felleisen, the 2011 SIGCSE award winner for outstanding contribution to CS education and the 2012 SIGPLAN award winner for significant contribution to the field of programming languages, a group of CS professors and their students from several universities developed PBD as an innovative curricula and outreach program. PBD addresses some of the most well-known issues in introductory programming courses, like the "blank page" syndrome, program diagnostics, visualizing programming and thinking processes (Lapidot, Levy and Paz 2000), emphasizing testing (Felleisen, Findle, Flatt, and Krishnamurthi, 2004), and tailoring IDEs for learners' needs and prior knowledge by offering a series of pedagogic language subsets, in which at every level the error messages never depend on knowledge that the student does not yet have.

Rooted in the paradigm of functional programing (Felleisen, Findle, Flatt, and Krishnamurthi, 2004), PBD is a functions-first approach to teaching introductory programming and problem-solving emphasizing good software engineering practices such as early testing from the beginning (Tuttle, 2011). This is combined with the Racket IDE (Flatt and PLT, 2010), featuring different language levels, simple syntax, customized error messages, and support for 'algebra of images' that enables students to write code for graphic- and animation-rich computing problems (Bloch, 2010). Most importantly, this educational approach offers design recipes to lead beginner students through a sequence of steps to obtain an understanding of the problem's nature and the solution program's behavior, hence the approach title. Testing is an integral part of the design recipe enabling test-first development (Crestany and Sperber, 2010). Thus, the multi-step design recipe is useful not only to write programs but also to diagnose them. The design recipe scales naturally to the design of more and more complex systems of functions. This in turn empowers students to design programs for deep and interesting problems after just a minimum of introduction to the language, the environment, and the design recipes (Bloch et. al. 2010; Felleisen, Findler, Flatt, and Krishnamurthi, 2001).

The features of PBD make the approach especially suitable as a starter for a computer science track of courses in the liberal-arts-oriented undergraduate curriculum in Community Information Systems. Since PBD also includes support for transitioning to object-oriented programming in Java and a method for using design recipes as part of an Object Oriented Programming IDE, the second course in the computer science track builds naturally on the first while further focusing on the systematic design of programs and classes as a preparation for the more advanced computer science courses (see the courses of Year 2 and Year 3 in Figure 1).

An overview of the main computer science concepts in the first course is provided in Table 2. The course meets once a week for a four-hour lab session. The sessions are freely organized as a blend of short lecturer presentations, individual lab work, pair programming, and reflective class discussions.

[1] http://www.programbydesign.org/overview

During the first weeks of the course, the students focus on fundamentals of programming by dealing with tasks involving pictures of their own choice, like overlaying one picture upon another, cropping certain parts of a picture, and drawing "the big picture" by combining picture parts (Bloch, 2010). Such focus is made available by the special library of functions that supports "algebra of images" included in the "beginner student" dialect of Racket IDE. Programming with pictures also enables students to creatively develop a small-scale programming project at an early stage of the course.

| **Week** | **Content** |
|---|---|
| 1-2 | Built-in functions for manipulating pictures |
| 3 | Global variables as names for complex functional expressions |
| 4 | Contracts and error tracking |
| 5 | First mini-project: flags |
| 6-7 | Defining new functions, Parameters as local variables, scope |
| 8 | Design recipes and test cases |
| 9 | Second mini-project: simple animation |
| 10-11 | Conditionals |
| 12 | Structures |
| 13 | Final project |

*Table 2. Course Structure*

One recurring pedagogical pattern is weaved into many of the learning experiences and curriculum materials. In coordination with the test-first approach (Crestany and Sperber, 2010) and with Racket's unique function called check-expect[2] (Bloch and Proulx, 2007) that play an essential role in developing the culture of 'Program by Design', students are constantly asked to predict what they would expect to happen in a given situation before running the program, and write it down; to carefully observe and check the results of running the program; and finally to explain the results (which may or may not be what they predicted). This learning pattern follows the very well-known Predict-Observe-Explain structure (Linn and Eylon, 2006) found successful in science education (Millar, 2004). By using such pattern within the context of learning to program and by being able to embed it within the actual code, non-CS majors have the opportunity to experience one of the key components of the professional process of program design even at very early stages.

# 4 CONCLUDING REMARKS

A first course in computer science is in some sense an almost impossible task (Bloch, 2010), in which students with various backgrounds need to learn (a) the grammar and the components of a programming language and (b) how to analyze a problem and design a program to solve it using that language. It is considered quite easy to get caught up in the details of (a) at the expense of (b), but the language itself might be obsolete by the time the students finish their course of study. The much more lasting knowledge is constructed through dealing with how to design a program that is both correct and easy to write, read, modify, and repair.

This might be true for any first computer science course, whether in middle school, high school, or college. It is certainly true for an introductory course for non-CS majors who are not intended to become professional programmers, as is the case described above. The focus on design patterns, which are step-by-step "recipes" for getting from a vague description of a problem to a working computer

[2] Racket has a check-expect function which can be used inline with other Racket code to compare the result of two expressions.

program, gives such students the "taste" of how professional programmers work in real-life contexts. The emphasis on test-first tools and pair programming helps the students experience a culture of programming and thus better understand the professional discourse among teams of programmers.

As the new program in Community IS has only recently opened at our college, the implementation of 'Program by Design' with the first students has been in its early trial phase and the following conclusions are thus preliminary. However, even at this early stage, three central principles have proven viable for non-CS majors with variable background in programming. First, using pedagogically-grounded language tools and IDEs can indeed support novice learners' focus on the design process rather on the specifics of the language. Second, initiating the programming venture with tasks within the world of pictures, graphics, and animations, while building on an "algebra of images" rather than on arithmetic, opens up a whole new and unprejudiced context for both novices and those who have had some previous programming experience (or even a lot of experience). Within such a context, creativity often shows itself both at the level of the design process and at the level of the product. And third, applying test-first design and strict documentation requirements as early as possible (the third or four week) enforces disciplined programming while illustrating in a concrete way the program's desired behavior. Within the context of pictorial functional programming, students can combine small pieces of code into quite a complex program at an early stage. From the very beginning of the course they practice writing test cases before writing each function definition, and gain experience in phrasing each piece's contract (Felleisen, Findler, Flatt, and Krishnamurthi, 2001), expected result, and test cases.

In summary, although the students in the Community IS program are not expected to become professional programmers, their exposure to these basic features of software engineering makes them more able to talk to computer scientists, understand these professionals' concerns, collaborate with them in developing and maintaining organizational and communal IT projects, and at the same time to develop their own interdisciplinary career on a proper foundation. In the specific case here described, it is too early to tell how successful the CS introductory course will be in these regards. However, considering other cases, it is clear that one key to the success will be the distilling of well-known functional principles of programming into generally applicable design recipes that work also in other paradigms like that of object-oriented programming. In light of current trends that call for programming for all[3] and regard coding as the literacy of the 21st century, the proposed CS track tailored for non-CS majors presents a valuable alternative to consider. Together with its 'Program by Design'-based introductory course, such a CS track has the potential to give students majoring in any field an understanding of the principles of computing and knowledge about the practices of computing professionals. That knowledge will undoubtedly support students' ability to make the choice so much needed "In the emergent, highly programmed landscape ahead… - Program or Be Programmed" (Rushkoff, 2010).

## References


Benbasat, I., and Zmud, R. W. (1999). Empirical Research in Information Systems: The Practice of Relevance. *MIS Quarterly* 23 (1), 3-16.

Bennett, J. and Sterling, J. (2011). Computer Science is not enough. tripleC 9(2) Special Issue: ICTs and Society - A New Transdiscipline?, 624-631. http://www.triple-c.at/index.php/tripleC/article/view/190 .

Bloch, S. and Proulx, V. K. (2007). TeachScheme, ReachJava: Introducing OOP without drowning in syntax. Journal of Computing Sciences in Colleges 22(4).


[3] As Audrey Watters posted in *Hack (Higher) Education* blog http://www.insidehighered.com/blogs/should-all-majors-not-just-computer-science-majors-learn-code (Jan 10, 2012). See also http://codeyear.com/ .

Bloch, S., Clements, J., Felleisen, M., Findler, R.B., Fisler, K., Flatt, M., Proulx, V. and Krishnamurthi, S. (2010). Program by Design Project Overview. http://www.programbydesign.org/overview .

Bloch, S. (2010). Picturing Programs: An Introduction to Computer Programming. King's College London: College Publications.

Carr, N. (20080. The Big Switch. WW Norton & Company, New York.

Crestany, M. and Sperber, M. (2010). Experience report: growing programming languages for beginning students. In proceeding of ICFP '10 - the 15th ACM SIGPLAN international conference on Functional programming. Baltimore, MD.

The College Board (2012). Proposed New Course and Exam—AP® Computer Science: Principles. http://www.collegeboard.com/html/computerscience/index.html#backrational

Felleisen, M., Findler, R.B., Flatt, M. and Krishnamurthi, S. (2001). How to Design Programs. MIT Press.

Felleisen, M., Findler, R.B., Flatt, M. and Krishnamurthi, S. (2004). The TeachScheme! project: computing and programming for every student. Computer Science Education 14, 1 (March 2004), 55-77.

Felleisen, M., Findler, R.B., Flatt, M. and Krishnamurthi, S. (2004). The structure and interpretation of the computer science curriculum. Journal of Functional Programming, 14, 4 (July 2004), 365-378. DOI=10.1017/S0956796804005076

Flatt, M. and PLT. (2010). Reference: Racket. Technical Report PLT-TR-2010-1, PLT Inc. http://racket-lang.org/tr1/.

Gurstein, M. (2008). What is community informatics (and why does it matter)?, Polimetrica, Milan, Italy.

Harvey, B. and Garcia, D.D. (2011). CS10 : The beauty and joy of computing. UC Berkeley course site http://inst.eecs.berkeley.edu/~cs10/fa11/

Kizza, J.M. (2003). Ethical and Social Issues in the Information Age. Springer-Verlag Inc. 2nd edition, New York, NY.

Kling, R. (1999). What is social informatics and why does it matter? D-Lib Magazine, 5, 1 (Jan. 1999). DOI= http://www.dlib.org:80/dlib/january99/kling/01kling.html

Lapidot T., Levy D. and Paz T. (2000). Teaching Functional Programming to High School Students. In Robson R. (ed.), Proceedings of the international conference on mathematics/science education and technology *(M/SET)*, 245- 249, San Diego, CA.

Levy, D. (2010). Information Systems curriculum goes social: An Israeli proposal for undergraduate curriculum in Community Information Systems. The 5th Mediterranean Conference on Information Systems – MCIS2010, Tel Aviv, Israel.

Linn, M.C., and Eylon, B.-S. (2006). Science education: Integrating views of learning and instruction. In P.A. Alexander & P.H. Winne (Eds.), Handbook of educational psychology (2d ed.) pp. 511-544. Mahwah, NJ: Erlbaum.

Millar, R. (2004). The role of practical work in the teaching and learning of science. Paper presented at the meeting: High School Science Laboratories: Role and Vision, National Academy of Sciences, Washington, DC. http://www7.nationalacademies.org/bose/millar_draftpaper_jun_04.pdf

Rushkoff, D. (2010). Program or Be Programmed: Ten Commands for a Digital Age. Introduction, Retrieved 1.12.12 from http://www.amazon.com/Program-Be-Programmed-Commands-Digital/dp/1935928155#reader_B004ELAPME .

Stillman, L., and Linger, H. (2009). Community informatics and information systems: how can they be better connected? The Information Society 25 (4) (Jul. 2009), 255-264.

Topi, H., Valacich, J.S., Wright, R.T., Kaiser, K., Nunamaker, Jr., J.F., Sipior, J.C., and de Vreede, G. (2010). IS 2010: Curriculum guidelines for undergraduate degree programs in Information Systems. Communications of the Association for Information Systems Vol. 26, Article 18.

Tuttle, S.T. (2011). Introducing programming in a functions-first manner, using the "Program by Design" approach. The Journal of Computing Sciences in Colleges, 27, 1, p. 101.