December 2006

An HCI-Oriented Approach to the Introductory IS Programming Course

E. Vance Wilson  
*University of Wisconsin-Milwaukee*

Matt Germonprez  
*University of Wisconsin-Eau Claire*

Follow this and additional works at: [http://aisel.aisnet.org/mwais2006](http://aisel.aisnet.org/mwais2006)

**Recommended Citation**

[http://aisel.aisnet.org/mwais2006/20](http://aisel.aisnet.org/mwais2006/20)

This material is brought to you by the Midwest (MWAIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in MWAIS 2006 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
An HCI-Oriented Approach to the Introductory IS Programming Course

E. Vance Wilson
University of Wisconsin-Milwaukee
wilsonv@uwm.edu

Matt Germonprez
University of Wisconsin-Eau Claire
germonr@uwec.edu

ABSTRACT

Despite increasing need for information systems (IS) students to graduate with “real world” business perspectives and strong problem-solving skills, introductory IS programming courses typically are taught in a manner that is disconnected from meaningful interaction with business problems or with needs of actual software users. Further, the syntax-oriented approach used in most courses raises sociological barriers to student success. This paper proposes a realignment of IS programming curriculum using an HCI-oriented approach. Potential benefits of this approach are increasing social support and motivational aspects of programming and expanding students’ exposure to business problems and user needs.

Keywords

INTRODUCTION

The IS profession has changed dramatically over the past decade due to implementation of new business strategies (e.g., offshore software development), adoption of transformational technologies (e.g., e-commerce), and economic flux (e.g., build-up and lay-off of IT staff related to the year 2000 problem and the dot-com bubble). IS academic leaders have responded by calling for curriculum revisions that will help develop students’ abilities in three key areas:

• broad business and real world perspective, including an understanding of the role IS can play in business strategy;
• strong analytical and critical thinking skills, with a focus on problem solving and goal achievement; and
• strong interpersonal communication and team skills, with a requirement for “curiosity, creativity, risk taking and a tolerance of these abilities in others” (Davis et al., 2001, p. 3).

Programming coursework is a part of this model curriculum. However, the typical introductory IS programming course focuses almost exclusively on training individual students in syntax-centered aspects of the programming language rather than the larger context of the program’s purpose and use, e.g., as a part of business strategy. Student effort is expended toward developing syntax skills and becoming familiarized with concepts and controls of the language. Authors of programming textbooks directed toward IS students typically present end-of-chapter exercises in terms of business applications, but this is largely window dressing. In our experience, students in introductory IS programming courses typically have little opportunity to understand how their small programs might apply to real world business, to apply problem-solving skills beyond getting their code to compile successfully, to collaborate meaningfully, or to exercise much curiosity or creativity.

Some may argue that a syntax-centered focus is essential to introductory programming coursework. However, this argument is contradicted by numerous empirical studies. Consider a study conducted to investigate why over half the business majors in a required introductory programming course at Utah State University failed to successfully complete the course and why the proportion of female majors in computer science (12% females) was so much lower than in related majors, such as math (49% females). The authors write,

[for many students in CS1, it appears that rather than learning the basic concepts of the field, their energies are devoted to learning syntax. Rather than learning real problem solving skills, they resort to trial-and-error. Rather than “getting the big picture” of computer science, they narrow their focus to “getting this program to run.” (Allan & Kolesar, 1996, p. 2)

In fact, a number of alternatives have proved to be more effective than the syntax-centered approach to introductory programming. These include game-based programming, e.g., “turtle” Logo (Papert, 1980), programming with robots (Williams, 2002), pair programming techniques coupled with project reporting requirements (VanDeGrift, 2004), and
completing preparatory coursework in problem solving techniques (Allan & Kolesar, 1996). Kelleher and Pausch (2005) write that it is essential to overcome sociological barriers in introductory programming courses by providing social support and motivational contexts that give students a reason to program. In this paper, we propose a human-computer interaction (HCI)-oriented approach to introductory programming coursework that meets both objectives by training IS students in a process of collaboratively developing software that can support business strategies and meet the operational needs of actual users.

**ESSENTIAL ASPECTS OF HCI**

HCI is “a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them” (Hewett et al., 1996). The study of human computer interaction has ranged from early work on computer interfaces (Dickson et al., 1977) to user constructed visualizations (North and Shneiderman, 2000). Aimed at helping users reduce information overload, human computer interaction has largely focused in the development of computer-based tools and design principles in support of cognitive mechanisms (Jarvenpaa, 1989; Vessey and Galleta, 1991). Prior to the past decade, relatively little attention was paid to HCI issues in IS education and research, however, this area recently has seen strong growth, as evidenced by the advent of an active special interest group (SIGHCI) of the Association for Information Systems (AIS) and numerous HCI tracks at IS conferences. This increased attention toward HCI issues reflects the growth of end-user computing in business settings. The programming duties of IS professionals increasingly emphasize developing, adapting, and evaluating applications that provide user interface (UI) support for end users rather than creating reports and coding efficient algorithms for large transactional systems, e.g., to handle payroll, billing, or point of sale transactions. Correspondingly, there is increasing need to provide HCI training to IS students so that they can handle UI-development duties effectively.

One approach that is recommended in the literature is to create an HCI “minor” in IS academic programs by substituting two or three HCI courses for other IS electives (Hewett et al., 1996). This approach is problematic, however, as it preempts IS electives that are important gateways to professional specializations and may themselves require UI development, e.g., data warehousing and web development. In recognition of the increasing need for UI development throughout the range of IS software projects, we propose that key HCI principles and methods should be integrated into IS programming coursework, including the introductory course.

To frame the aspects of HCI that we feel are essential for IS students, we draw from the SALVO method for UI development (Wilson & Connolly, 2000). SALVO is a mnemonic acronym that prescribes five stages in the development process, based upon HCI principles. The first three of these are planning stages in which developers Specify user needs, Adopt a technology-specific design guide, and Leverage appropriate matches between technology and user needs to enhance user abilities and support user disabilities. Once planning stages are initially completed, developers perform Visualization techniques including mock-ups to elicit user feedback early in the development cycle. Visualization is followed by an Observation stage in which developers observe and evaluate attempts by users to perform representative tasks using the mock-up or completed software. SALVO describes an iterative development model in which performance is based on achieving a goal, in the manner that a salvo of applause continues until a performer is adequately acknowledged or a salvo of artillery fire continues until a target is destroyed. In the following section, we present a model curriculum for an HCI-oriented introductory IS programming course based on the SALVO method.

**MODEL CURRICULUM**

The model curriculum we propose is directed toward teaching a visual language, such as Visual Basic, Java, or C++, within an integrated development environment (IDE), such as Visual Studio .Net. The curriculum is organized into lecture and hands-on (lab) portions. However, the proposed program does not require a visual language/environment. By not using a visual environment, instructors are able to put implicit emphasis on the programming syntax without addressing it specifically. A non-visual environment moves coding to the background of systems development by not focusing on syntax and its associated environment as a means to an ends. Instead, the language and environment becomes a simple tool enabling the students to focus on the larger goal of finishing a project.

The lecture portion of the course addresses three instructional components. The first component develops the concept of integrated business systems, in which business functions are supported by interconnected software applications. This instruction should emphasize input, process, and storage requirements of business data, as well as processes by which data flows between functions. At completion, students should be able to conceptualize the needs they would have as users of each software application. The second component presents HCI principles using the SALVO method. At completion, students should understand the process of planning and visualizing the program prior to developing functional code and be capable of...
observing user interaction with the program at one or more stages of completion. We anticipate these two components can be completed during the first three weeks of a semester-length course. The third component teaches programming methods with instructional examples directed toward integrated business systems. Because the time allocated for this component is necessarily reduced from a typical syntax-centered course structure, in-depth coverage should be given selectively to topics that are central to business software development. For example, we argue it is not essential in an introductory course to present in-depth coverage of sorting algorithms when teaching use of arrays or teaching varied file input/output options when teaching basic database connectivity (see Table 1).

<table>
<thead>
<tr>
<th>Week</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Integrated Business Systems Design</td>
</tr>
<tr>
<td>2 - 3</td>
<td>Planning and Visualizing System Design</td>
</tr>
<tr>
<td>4 - 16</td>
<td>Programming Methods, Instructional Examples, and Hands-On Exercises</td>
</tr>
</tbody>
</table>

Table 1. Overview of Course Layout

In the hands-on portion of the course, students are formed into project teams of two or three members in which they play dual roles. Each student is responsible for independently programming a portion of the overall system. In the role of programmer, students interact with a user to understand specific program requirements. The role of user is played in turn by another member of the team and—for final usability testing—by members of other teams.

Note that we do not argue that this is an appropriate model curriculum for a human computer interaction program nor a replacement for an HCI course. Instead, we propose that components of HCI can and should be integrated into IS-related course curricula, thus creating the foundation for an HCI-oriented approach.

**Demonstration**

We demonstrate the HCI-oriented approach with an integrated business system that can handle customer management, inventory management, and billing (see Figure 1). Each programming team (PT) produces a complete system with each team member producing one of the components, including at least one formatted report and database table. Most teams in this design will comprise three members; however, two-member teams can be formed by dropping the customer management or inventory management component.

The design of each project is guided by two sources. General system functionality is defined by guidelines provided by the instructor. Appearance is guided by designs provided by an assigned user team (UT). These designs are created by each team as part of the lecture instruction in the concept of integrated business systems prior to being handed off to the PT for production.

Four modules are used to incrementally develop the demonstration business system as shown in Table 2. Each module implements a complete experiential learning cycle (Kolb, 1984), comprising four stages:

1. the learner’s conceptual framework is tested and applied in a novel situation;
2. the learner does something concrete or has a specific experience;
3. the learner observes and reflects on the experience and their response to it; and
4. observations are assimilated into the learner’s conceptual framework.

Each module is initiated by user input or by observation from a previous module. This new conceptual framework drives creation and/or revision of the programs, including reflection on the results of prior decisions. At the end of each module, observations are conducted to evaluate the outcome of actions taken within the current module.

**DISCUSSION**

The HCI-oriented approach to introductory programming that we propose requires substantial changes from the syntax-centered approach that typifies current instruction. We conclude this paper by describing the major costs and benefits that are anticipated with this approach and arguing that benefits exceed costs in this case.
Costs accrue in three principle areas. First, increased emphasis on business processes and user needs will reduce the time students have to learn syntax and program functions. One way of mitigating this cost would be to fully integrate the introductory programming and system analysis and design courses into a two-course HCI-oriented sequence. However, discussion of this approach is outside the scope of the present paper. Second, the instructional design is substantially more complex than typical syntax-centered courses. This requires more initial organization on the part of instructors; however, we argue that increased effort in this area is counter-balanced by reduced labor in other areas, such as grading. Third, students who drop out of the course can undermine productivity of team members.

We believe that the HCI-oriented approach achieves several goals. Social support for students, feeling of team effort without compromising personal accountability is achieved. Students participate in a meaningful business project which provides real world experiences in a classroom setting. Students are motivated to complete programs promptly in order to meet schedules involving their peer and to test and debug adequately to avoid social stigma. Students have less opportunity and motivation to cheat by copying others’ code and learn much more if they do copy code by the effort necessary to modify it through diverse project appearance and control names. Finally, students are better supported and motivated to participate actively, drop-out rates will be reduced, and courses may become more attractive to students who are uncomfortable with the syntax-oriented approaches that are currently being applied.

Figure 1. Integrated Customer, Inventory, and Billing Business System.
<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Programming Techniques</th>
<th>User Input</th>
</tr>
</thead>
</table>
| 1 | Based on user input, create the UI for each component of the integrated business system; all events that will produce functionality in the final application must be represented by message dialogs | Introduction to SALVO method, programming language IDE, visual controls, and the programming code necessary to create message dialogs representing functionality | • Week 3: Programming team (PT) receives initial design and function input from user team (UT)  
• Week 5: PT observes use of UI for each application by members of UT |
| 2 | Refine UI based on observation conducted in Week 5; create functionality necessary for calculation and logic in each component | Constant and variable declarations, built-in functions, decision structures, repetition structures, and error-handling | • Week 8: PT observes use of software component by the PT member assigned as internal user for the component |
| 3 | Refine component code based on observation conducted in Week 8; create formatted reports from each component (e.g., customer list, in-stock report, and customer invoice) | Arrays, file input and output, and report formatting | • Week 11: PT observes use of software component by the PT member assigned as internal user for the component |
| 4 | Refine component code and report appearance based on observation conducted in Week 11; create database tables for each component and augment code with the ability to store and retrieve data; conduct internal functionality testing | Database design and development, data manipulation, methods of functionality testing | • Week 12: Coordinate database schemas within PT  
• Week 14: PT observes use of integrated system by at least two recruited test users external to the team  
• Week 15: PT observes use of UI for each application by members of UT and members of other teams as time permits |

Table 2. Detailed View of Instructional Modules (Based on 15-Week Semester)

REFERENCES


