Applying Software Design Methodology to Instructional Design

J. Philip East
Computer Science Department, University of Northern Iowa, IA, USA

ABSTRACT

The premise of this paper is that computer science has much to offer the endeavor of instructional improvement. Software design processes employed in computer science for developing software can be used for planning instruction and should improve instruction in much the same manner that design processes appear to have improved software. Techniques for examining the software development process can be applied to an examination of the instructional process. Furthermore, the computer science discipline is particularly well suited to these tasks. Thus, computer science can develop instructional design expertise for export to other disciplines to improve education in all disciplines and, eventually, at all levels.

1. INTRODUCTION

Every academic discipline develops procedures for use in research and practice. This is true of physics, chemistry, psychology, anthropology, history, and so forth. It is also true of computer science. However, procedures used to work within the discipline are seldom, if ever, applied to instructional development. This is understandable, as experts tend to focus on expertise in the discipline rather than on teaching or instructional development. This practice should be questioned. Surely if these specialized techniques can be utilized for finding scientific truth, making good interpretations of social phenomena, and designing effective systems, they might also be useful for finding instructional truth, making good interpretations of instructional activity, and designing effective instruction.

Address correspondence to: J. Philip East, Computer Science Department, University of Northern Iowa, 1227 West 27th Street, Cedar Falls, IA 50614, USA. Tel.: +1-319-273-2939. E-mail: east@cs.uni.edu

10.1080/0899340042000303438$16.00 © Taylor & Francis Ltd.
The next section explores the link between software development and curriculum development, reports on an examination of several texts on curriculum and instructional design, and considers the application of concepts from the area of curriculum and instruction to computer science education. The main part of the paper investigates two areas of possible transfer from software development to instructional development. The first major section illustrates applying the software development life cycle to both discipline-level curriculum design and course-level instructional design and provides an example of applying this approach in designing a CS1 course. The second major section suggests an analogy between a software process maturity framework and a maturity framework for the instruction process. The concluding sections argue that the computing field has many advantages over other fields for offering techniques that can be used to improve instruction and our understanding of it.

2. BACKGROUND

To date there has been little or no attempt at applying software development techniques to curriculum development in computer science. A search of the ACM Digital Library for items with the key word “instructional design” resulted in nine hits, none of which suggest using software design techniques for course or curriculum development. A search using “curriculum design”\(^1\) as a key word resulted in 20 hits. Only one of those articles addressed applying techniques from the computing field to instructional planning. In that paper, Ramírez and Fernández (2003) suggested using UML as a tool for organizing and communicating the elements of computing courses.

The notion that design is a generally applicable process is not a new one (Simon, 1996). However, it seems not to be used in teaching and curriculum

\(^1\)The following clarifies how terminology is used throughout this paper:

- **Instructional design** is a general term to mean the planning of instruction. It includes curriculum design and course design.
- **Program** will refer to computer programs and **programme** will refer to an area of study.
- **Curriculum design** will refer to the design of a programme of study or major developed by a professional organization or by college or department offering a degree programme.
- **Course design** will refer the design of a single course. **Design** will be equated with planning.
- **Instructional development** includes the production of teaching materials (e.g., assignments) and activities with some redesign as needed.
development. Simon describes a possible science of design and includes computer scientists among those who could practice this science. He mentions several other professionals likely to use it: architects, attorneys, civil engineers, and physicians. Later, he implies that managers of nearly any kind of organization or institution can make use of the science of design. Interestingly, Simon does not suggest that those teaching design should use design science to organize or plan their curricula and teaching.

When discussing the teaching of computing, computer science educators tend to omit discussion of curriculum and course design. Lapidot and Hazzan (2003) suggest a considerable number of areas of knowledge and expertise to be included in a course for teachers of high school computer science. Their list of topics is quite extensive and includes what they call “soft ideas”, such as how to teach design. However, they do not explicitly address course design in their discussion of the methods course nor, more critically, do they suggest course design is something that prospective teachers should learn.

At some institutions, techniques for designing a course of instruction may be included in the education offerings for teachers rather than in the offerings within the discipline. At my own institution (a teaching university in the Midwestern USA with a strong teacher education program) this seems not to be the case. Prospective secondary teachers take content-specific courses similar to those taken by non-teaching majors. They also take one or two methods courses taught by education faculty in the subject area and similar to the one described by Lapidot and Hazzan (2003). Additionally, the prospective teachers must complete courses in human development, learning, classroom evaluation, schools in American society, human relations, diverse learners, instructional technology, field experience (in two separate extended contexts), and student teaching. In the education courses, no specific attention is paid to curriculum or course development or design. At my institution, these topics are reserved for graduate students (and this also was my experience 25 years ago as a graduate student). Thus, neither education nor methods courses appear to supply prospective teachers with guidance in course or curriculum development.

The remainder of this section reports on an examination of three texts on curriculum development.2 That analysis yields some ideas pertinent to

---

2Two of these texts were familiar from graduate school and were distinct in their approach to the topic. The third was selected because its approach to curriculum development provided a third distinct approach. Together the three provide comprehensive coverage in that a learning-theoretic perspective, a general philosophical view, and a practice-oriented approach are all included.
curriculum development within computer science. It also suggests that the transfer of ideas and techniques from a relatively complex specialty within one discipline (e.g., curriculum development) to another discipline (e.g., computer science education) may be non-trivial.

Gagné and Briggs (1979) introduce the following nine-step process for instructional design:

1. Investigate the needs for instruction including goals, resources, and constraints.
2. Develop an organizational framework with course descriptions and expectations.
3. Define objectives in terms of performance of the various learned capabilities (intellectual skills, cognitive strategies, information, attitudes, and motor skills).
4. Specify the conditions needed for attaining the desired performance.
5. Plan the sequence of instruction that takes into account prerequisite structure and progression from narrow to more broad scope.
6. Plan the units of instruction focusing on the performance expected at the end of the unit.
7. Plan detailed lessons of instruction.
8. Determine how student learning will be assessed.
9. Package the course planning for its delivery and assessment aimed at improving and refining instruction.

Gagné and Briggs provide a relatively detailed examination of the nature of the various types of learning including: a characterization of performance, what the student must bring to the learning, and what the instruction should provide. They also address the analysis of learning tasks and definition of performance objectives. Of particular note is the statement, “The question initially asked by the designer is not, ‘What will students be studying?’ but rather, ‘What will students be doing after they have learned?’” (p. 97). The middle section of the book provides a very detailed discussion of the various elements of instructional design. The latter portion of the book discusses group instruction, individual instruction, and evaluation.

Doll (1996) takes a much broader look at curriculum planning. He begins with a philosophical and historical discussion of curriculum development, including seven schools of thought about curriculum. He then examines the psychological bases of curriculum development by discussing learning, student characteristics, and the application of learning theory. The application
of learning theory addresses: providing variety in learning experiences, giving
direct instruction, teaching indirectly, encouraging mastery learning, using
time wisely, helping pupils learn to think, encouraging learning that is
authentic, responding to differing learning styles, recognizing but not
exaggerating sex differences, cooperative learning, and highlighting the
rewards of learning. Next, Doll discusses social and cultural forces impinging
on curriculum decisions. Finally, he examines the role of subject matter. It is
clear that Doll is working at a very broad level from the aspects of subject
matter organization that he uses, that is, organizing by: subjects, correlation,
broad fields, core programmes, persistent life situations, pupil experience, and
the placement and addition of new subject matter. Actual curriculum design is
discussed about one-third of the way through the book. The process Doll
suggests has many similarities to that proposed by Gagné and Briggs (1976).
Doll’s suggested process includes: determining need, stating goals, deter-
mining objectives (cognitive, affective, and psychomotor), identifying means
of evaluation, choosing a design approach, specifying learning content,
determining and organizing learning activities, and assessing the design. As
with Gagné and Briggs’ approach, objectives are to be stated in terms of
desired student performance. Doll provides some useful insights but also
includes material extraneous to the current discussion, such as philosophical
issues related to curriculum development and topics not closely related to
college teaching, e.g., teaching children of migrant workers, education about
environmental hazards, and multiculturalism.

Posner and Rudnitsky (1994) take a more teacher-oriented approach. They
quickly get to course planning and suggest starting by listing and analyzing
course content information: initial ideas, course title, preliminary outline,
intended learning outcomes (ILOs), central questions or big ideas, cognitive
map of terminology and concepts, flowchart of skills, and students’ existing
knowledge and skill. Their next step is to develop a rationale for the course
that addresses learners, society, subject matter, and educational goals. Once
the outline and rationale are finalized, the ILOs can be refined and categorized
as cognitions, cognitive skills, psychomotor-perceptual skills, or affects. Next
the ILOs are organized around instructional foci and formed into units of the
course that will be organized into a reasonable sequence using a wide variety
of considerations. The last two steps are to plan the instructional activities and
course evaluation.

Examining these texts was a challenging task. Instructional design occurs
at various levels – societal, discipline, course, unit, and lesson – and may be
expected to take into account seemingly unimportant philosophical and social matters. Analyzing the theoretical suggestions (as in Gagné & Briggs, 1979) for practical implications required familiarity with specialized terminology and a general understanding of learning, teaching, and assessment that is included in the study of education but not computer science. Even analyzing the practical suggestions of Posner and Rudnitsky (1994) was somewhat tedious as it required considerable educational context and the examples were in non-computer science areas at the K-12 level. These factors suggest that few computer science educators are likely to delve into this literature (though doing so would likely be useful).

3. APPLICATION IN COMPUTER SCIENCE EDUCATION

Professionals in computer science education can and should learn from curriculum specialists in education. Of particular note is the emphasis on performance objectives rather than knowledge or understanding. Ultimately, computer science students will be asked to perform as software developers, system designers, researchers, and so forth. It makes sense to have successful performance of those and related tasks as the goals of instruction.

It may also be useful to think of classifying the desired learning. Bloom (1987) indicated three broad categories (cognitive, affective, and psychomotor), with the cognitive area including the progressively more demanding levels of knowledge, comprehension, application, analysis, synthesis, and evaluation. Gagné and Briggs (1979) used the terms intellectual skills, cognitive strategies, information, attitudes, and motor skills. Within computing, Snyder et al. (1999) used similar terms of intellectual capabilities, concepts, and skills when addressing what everyone should learn about computing. The use of some categorization of learning seems appropriate when planning instruction.

A process of curriculum development for use in computer science needs to be adopted, adapted, or devised. However, rather than struggling to separate the wheat from the chaff in a process designed for all areas of K-12 instruction, computer science educators could use a more familiar design process. Computer scientists have developed processes that enhance the quality of the products they produce. Examples of such processes include stepwise refinement, structured design and programming, software development life cycles, unit testing, and code refactoring. While not perfect or universally accepted, each of these practices (as well as others) have enhanced the activity
of software development. As Simon (1996) suggests, design is general. It can be applied to natural or physical entities (bridges and buildings) and to artificial entities (software, medical diagnosis, and organizations). Surely it can be applied to instruction.

4. APPLYING THE SOFTWARE DEVELOPMENT PROCESS TO CURRICULUM DEVELOPMENT

Like professionals in other disciplines, computer scientists tend not to apply strategies from their discipline to instruction. They should. The following discussion is illustrative, not prescriptive, of how software development strategies in computer science can be applied to instruction. It indicates how the software development process of requirements analysis, design, implementation, testing, and maintenance can be applied to both curriculum and course design.

4.1. The Discipline and Programme Level

At a high level there are two separate stages of curriculum design. Professional organizations, such as the Association for Computing Machinery (ACM) and the IEEE Computer Society (IEEE-CS), provide general recommendations for institutions purporting to teach computer science (e.g., ACM/IEEE-CS, 2001). Based on these reports, specific institutions offer programmes or majors that comply, to varied extents, with the discipline’s definition. Software development methodology can be applied similarly to both of these tasks.

4.1.1. Requirements Analysis and Specification

When developing software, practitioners start with explication of requirements of the system being developed: What should the system accomplish? Practitioners seek information from a variety of stakeholders in the system: users, managers, (perhaps) clients for/on whom the system will be used, and so forth. Analogously, when developing an instructional system, the stakeholders of the instructional system must become part of the dialog underlying the development process. Presumably this includes individuals in industry and research, (perhaps) government, (perhaps) students, and, of course, academics. In developing such a system, the goal should be to identify the abilities expected of graduates, not the body of knowledge they should
possess. Having knowledge does not mean one is able to apply it, either specifically or generally. That is why performance goals or objectives are important.

An analysis of the performance goals and objectives produces a draft set of desired abilities for graduates. The next step is to have the various stakeholders review the ability set and provide suggestions for improvement. This may be an iterative process. At some point the ability set is finalized, resulting in the specifications for the instructional system.

This process could occur at the discipline level or at the programme level. It would be reasonable for the next ACM/IEEE-CS curriculum process to begin in this way. Additionally, as departments review existing programmes or propose new ones, similar requirements analyses and specifications could be performed. Departments would begin with the specifications provided by the discipline, but because each department has a local constituency, their programmes will differ slightly from the discipline’s more general statement. In both cases, the specifications are a necessary first step to guide curriculum planning or design.

4.1.2. Design

The instructional system specification has defined a set of desired student abilities. The next step is to examine each of those abilities to determine constituent sub-abilities and how these might be developed. Design is not an algorithmic process. There are heuristics one can apply, however, such as bottom-up thinking, top-down development, and stepwise refinement. For example, one general specification in an early course might be “the ability to develop a good program”. That ability is quite complex. It involves at least the related abilities of problem analysis, data representation, algorithm design, algorithm implementation or coding, syntax debugging, test data design, testing and debugging, user interface design, and documentation. It also may include the ability to communicate with clients and co-workers and the ability to work cooperatively within a team. Each sub-ability might consist of another level of sub-abilities. Eventually, however, the skill refinement ends, perhaps with some simple task, such as the recall of knowledge or the ability to adapt familiar knowledge to an unfamiliar context.

How these abilities and sub-abilities might be organized into an overarching curriculum is a subject for further consideration and study. The notion that skills and expertise are context-sensitive may well mean that curriculum will continue to be organized as areas within the discipline.
(architecture, algorithms, etc.). Alternatively, the organization might be a set of mastery modules that must be completed in some sequence, after which students might choose (or programmes specify) units that address areas of expertise designed to develop abilities in elective contexts.

In this process, curriculum design will have occurred at both the discipline and programme levels. The discipline will have produced a template for curricula that can be used by local institutions. The template will explicate abilities and their components and organize them using an understanding of their relationships and of educational theory and practice. Departments will have started with the general discipline template and adjusted it to fit local specifications, needs, and understandings.

Good software design requires that all assumptions be documented. The same is true for curriculum design. Assumptions about learner abilities, the learning process, content or ability interdependencies, and so forth must be made explicit so design decisions can be examined in context during later reviews. Additionally, to advance computer science education as a discipline requires a public and shared understanding of practice within the discipline. Such an understanding requires that all assumptions be explicit.

4.1.3. Implementation
Implementation does not occur at the macro (or discipline) level. In reality, while departmental majors or programmes are implemented in some sense, the actual implementation occurs as courses are developed and taught – at the micro-level.

4.1.4. Testing
While implementation does not occur at the level of disciplines or programmes, testing of a new curriculum must be addressed at this level. The actual testing will occur in the context of courses, with those results fed back to the programme and discipline levels to allow for corrections in the specification or design. Without an explicit mechanism designed to transfer test results back to the more general levels, curriculum development will remain an ad hoc process.

4.1.5. Maintenance
The maintenance process provides the mechanism for correcting flaws arising from inappropriate specification, design, or implementation and adjusting to
new developments that dictate change in the specifications. The ever-changing nature of computing has necessitated periodic review and revamping of curriculum recommendations. The fact that the computer science education community conducts a relatively regular curriculum re-examination (e.g., ACM/IEEE-CS, 2001) is a bonus for instructional developers in computer science education in that the need for revisiting the curriculum is widely accepted. However, there is a need for a more formal mechanism for feeding data into curriculum maintenance.

Currently, curriculum revisions focus on examining current technology and professional practice to revise the elements of the discipline’s knowledge base and the relative attention paid to the various elements. This would need to change if the proposed focus on student capabilities were to be adopted. The performance goals outlined in the specification phase would need to be analyzed in order to identify data that could indicate success or failure. A process for collecting that data would need to be designed and incorporated into the implementation and testing phases. Then the maintenance process could have a formal, evaluative, data-based mechanism to provide feedback for the next re-examination of the curriculum for possible revision.

4.2. The Course Level
A new curriculum development process might well generate a curriculum model other than one subdivided into courses. Abilities or areas of expertise could become the organizing theme. However, there must still be some unit of planning smaller than a programme. That unit, referred to here as a course, is the focus of this portion of the discussion.

To some extent the course-level activities described below are equivalent to the implementation of the macro-level described above – a design problem. It is, however, a different design problem. At the higher levels, that is, for a major or a programme, the goal is to design the system. At this level, the goal is to design the course.

4.2.1. Requirements Analysis and Specification
At the course or unit level, requirements analysis and specification start with pre- and post-conditions, with the post-conditions being the instructional goals. The pre-conditions indicate expertise in the form of performance (and related knowledge) that students must bring to the planned instruction. The specifications for a course are refinements of course goals and should indicate capabilities students are expected to possess when they complete it.
4.2.2. Design
Once the desired abilities and performances have been captured via the specifications, curriculum design for the course involves selecting or designing activities expected to enhance the development of the desired abilities. The activities for students may include readings, exercises, programming assignments, small group discussions, and so forth. The design will probably also include teaching or instructional activities such as questioning, providing feedback, and demonstrating quality products and procedures. Designers will need to remember Gagné and Briggs’ (1979) exhortation to consider what students should be able to do after they have learned, rather than what students should study. Because the goals are specified as abilities or skills rather than knowledge, the activities may be somewhat different than those currently used. There will likely be some disequilibrium in initial attempts to design instruction from this different perspective.

4.2.3. Implementation
Implementation in this context refers to teaching. Similar to the implementation of software, there will also be some redesign of instruction during the implementation phase. As with course planning, teaching that is oriented toward performance rather than knowledge may require additional time or effort.

4.2.4. Testing
Testing in this context does not refer to testing students or testing programs, but rather to testing the results of the planned curriculum unit or course. This step must involve assessing students’ ability to accomplish the desired tasks. However, assessing student ability is only the beginning, rather than the end, of the testing phase. Student success or failure must be analyzed to determine which specifications were met and which were not. An important step is to determine whether the design or the implementation is at fault; such a determination may be difficult. In fact, a precise determination may be less important than identifying particular instructional problems that must be fed into the maintenance process for possible changes in specification or design.

4.2.5. Maintenance
The maintenance task uses the testing data as information to feed back into a new cycle of the steps completed earlier. Each future implementation of
the course will likely be somewhat different from the present one. It may be necessary to reconsider the specifications for the course and then to modify the design. Alternatively, the specifications might remain the same and the design still change as a result of teaching experience. During the maintenance phase, data also must be analyzed for any feedback into the more general curriculum processes of programme and discipline curriculum development.

4.3. A Course Example
This section illustrates using software development strategies to design a course. The goal is two-fold. First, the example shows how performance objectives can be used effectively in instructional design in computing. Second, the example provides insight into the suggested procedure for curriculum development. Only the first two steps of requirements analysis and design are addressed in detail.

4.3.1. Requirements Analysis
In a CS1 course, a realistic goal is that students should leave with the ability to develop good programs of relatively minor size and complexity. What does that mean? The various aspects of the goal must be specified in more detail to be useful in instructional design.

That programs are of relatively minor size and complexity could mean using two or three interacting classes, each with three to five methods. At least one of the classes might involve reading data from a file and organizing it within some collection in RAM. When done, data will be read back out to the file. The selection and iteration control structures will be used in the program.

The meaning of to develop programs might be that the student is given a rough specification of general capabilities with only the behavior specified. Determining specific classes and the methods within each class is left to the students.

Finally, what does it mean for a program to be good? A good program must be correct. Other characteristics of a good program can include that it is reasonably designed with respect to cohesion, coupling, and data and procedural abstraction; that it is well documented; and that it is formatted according to course standards.

4.3.2. Design
To help students reach the instructional goals discussed above, a course must include activities that ultimately lead students to design, implement, test, and
assess their programs. At the same time, students must also learn the syntax and semantics necessary to express programs of the expected size and complexity. Thus, instruction must address syntax and semantics (though perhaps not through lecture). Additionally, successful programming involves successful debugging. Instruction also must address the debugging task. Table 1 shows an example first-draft course design, which includes assignments of student work, but no teaching activity. The example provides insight into what is required when designing a course directed at student performance rather than knowledge.

Table 1. Sample Design of CS1 Course Targeted at Student Performance Rather than Knowledge.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity 1</td>
<td>Enter, compile, and run code consisting of a simple class (perhaps a constructor, accessor, and mutator) and a tester(^a) for the class.</td>
</tr>
<tr>
<td>Activity 2</td>
<td>Enter, compile and debug, and test and debug code with syntax and logic errors. Log the errors for discussion in class.</td>
</tr>
<tr>
<td>Activity 3</td>
<td>Revise the class and tester code as explicitly specified. Evaluate code correctness.</td>
</tr>
<tr>
<td>Activity 4</td>
<td>Develop code for a new class given tester code and detailed specifications (a model design is provided only after students have had an opportunity to attempt their own design). Test and evaluate code correctness.</td>
</tr>
<tr>
<td>Activity 5</td>
<td>Complete multiple exercises concerning syntax and semantics of language elements, for example, assignment, output, and selection. Evaluate correctness.</td>
</tr>
<tr>
<td>Activity 6</td>
<td>Complete multiple exercises concerning syntax and semantics of language elements, for example, looping. Evaluate correctness.</td>
</tr>
<tr>
<td>Activity 7</td>
<td>Given the tester code (or given an explicit specification for the tester class), design and develop method code for a new class (or expand an existing class) in ways that involve selection and iteration. Test and evaluate correctness.</td>
</tr>
<tr>
<td>Activity 8</td>
<td>Given a relatively detailed description of the new components, design and develop method code for a new class (or expand an existing class) in ways that involve selection and iteration. Provide appropriate tester code and evaluate code correctness.</td>
</tr>
<tr>
<td>Activity 9</td>
<td>Complete multiple exercises concerning syntax, semantics, and utilization of arrays or vectors. Test and evaluate correctness.</td>
</tr>
<tr>
<td>Activity 10</td>
<td>Given a relatively detailed description of the class behavior, design and implement a subclass of an existing class in ways that involve arrays or vectors. Provide appropriate tester code and evaluate code correctness.</td>
</tr>
</tbody>
</table>

(continued)
4.3.3. Implementation
A good implementation (teaching) requires detailed planning and production of specific assignments and lessons. The teaching also entails carrying out the plans, adapting them as necessary, and collecting data for assessment of student performance and of the instruction.

4.3.4. Testing
Plans for how to test the course itself are developed during the design phase and partially conducted during implementation (teaching). The final assessment activity for the course is to analyze the collected data.

4.3.5. Maintenance
The goal during the maintenance phase is to understand the implications of the data generated during the process of testing or evaluating the course itself. This information feeds into the planning for the next iteration of the instruction.

4.4. Summary
Software development is a design activity. Instructional development is also a design activity and the procedure used with software can be applied to

Table 1. (continued).

<table>
<thead>
<tr>
<th>Activity</th>
<th>Given a relatively detailed description of the class behavior, develop code for a subclass of an existing class in ways that involve selection, iteration, and arrays or vectors. Provide appropriate tester code and evaluate code correctness. Adhere to documentation and layout standards.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>Given a relatively general description of the class behavior, develop code for a subclass of an existing class in ways that involve selection, iteration, and arrays or vectors. Provide appropriate tester code and evaluate code correctness. Adhere to documentation and layout standards.</td>
</tr>
</tbody>
</table>

*The term “tester” refers to code that will invoke and test various class methods. Some IDEs can test class code without an explicit class containing a “main” method. Another testing alternative is JUnit.

Notes.
- Table includes major assignments only, no teaching activity.
- If students are expected to be able to do something, they must do and practice that thing. If students are to learn design, have them design. If they are to learn testing, have them test. If students are to learn what good code is, have them assess code. It also is necessary to provide feedback on those skills.
instruction at the discipline and programme levels as well as at the course level. Instruction should be designed to develop desired student skills. Data gathered to assess the outcomes of course instruction can be fed back into course redesign and implementation for future versions of courses. Additionally, the outcomes data of instruction can be collected and used in future curriculum recommendations at the discipline and programme levels. Computer science educators interested in instructional improvement can and should apply their knowledge of software development to instructional development. Computer science educators have access to a design process that, if applied to instruction, should enhance computer science education much as software design processes have enhanced software development.

5. DEVELOPING AN INSTRUCTIONAL PROCESS MANAGEMENT SYSTEM

Many universities now mandate that departments develop procedures for assessing their degree programmes. Meeting that requirement suggests another example of how computer science educators can apply a technique from their discipline to instruction – the software process maturity framework. Just as it is reasonable for software development organizations or entities to examine their software process, so is it reasonable for instructional organizations or entities to examine their instructional process. Indeed, without such an examination, it is likely that instructional processes will remain largely ad hoc. The software process maturity framework for process improvement provides another example of directing a specialized technique from the computing discipline toward the improvement of instruction.

One model of the improvement process for software development defines five levels of maturity: initial, repeatable, defined, managed, and optimizing (Humphrey, 1990). Each level is defined by certain characteristics. An organization whose process is at a higher level of maturity is expected to produce consistently better products than an organization at a lower level of maturity. The levels of maturity are determined to a great extent by the attention paid to an organization’s process.

The process of software development and delivery is both similar to and different from the process of planning and delivering instruction. However, a process known to improve the one might also be useful for improving the
other. The following discussion hypothesizes five levels of maturity of instructional process as an illustration of applying this discipline-specific tool to instruction. Again, the goal is to illustrate, not to prescribe.

## 5.1. The Model

### 5.1.1. Initial

The initial level of maturity relative to software development is characterized as ad hoc. Little or no attention is paid to the process used and until the process is under control, “orderly progress in process improvement is not possible” (Humphrey, 1990, p. 8). Presumably, there is no formal guidance exerted over how individuals or groups develop their products. For instruction, this might equate to instructors being provided a brief course description and perhaps little else (a reasonable assumption given the lack of explication of instructional process in the literature).

### 5.1.2. Repeatable

The repeatable process provides control over the way plans and commitments are established. An analog of the software development enterprise in instruction might be a formal procedure for developing, publishing, and enforcing adherence to syllabi within a department. With respect to instructional design, this might equate to the situation where, regardless of who teaches a course, all sections of the course look quite similar.

### 5.1.3. Defined

The defined process has: a process control group, an explicit procedure for process examination, and explicit methodologies for software development. A defined instructional process might be one in which: various programme and course goals are published, an outcomes assessment committee is functional, and a procedure is in place for feeding outcomes data back into instructional design.

### 5.1.4. Managed

The managed process collects and maintains data about quality and cost and has professionals analyze the data to provide feedback to practitioners and management. The analog in instruction might be characterized as a process in which only data-based recommendations are used to make instructional decisions.
5.1.5. Optimizing
The optimizing process examines, improves, and perhaps automates data collection; it also uses data to analyze and improve the process. An optimizing instructional process might be one in which the data from the managed process is effectively used to evaluate and revise the various aspects of instruction – requirements, specification, design, implementation, testing, and maintenance.

5.2. Summary
Clearly, there is much more to the software process maturity framework and its use than is supplied in this brief discussion. There also would be much more to any instructional process maturity framework than a mere characterization of the levels. The point is not to have levels of maturity but rather to study the process of instructional development and delivery in a manner similar to study of the process of software development and delivery. Again, it is reasonable and potentially very useful to apply tools extracted from the discipline of computing to instruction within the field.

6. COMPUTER SCIENCE AS AN EXPORTER OF INSTRUCTIONAL EXPERTISE
I have long thought that computer science is positioned uniquely to contribute to an understanding of the instructional enterprise. Several aspects of the field’s content and history contribute to that conclusion.

Computer science is a new discipline. On the one hand, that might be seen as leading to rather chaotic methodologies and problem selection (Kuhn, 1970). However, as a young field, computing brings a fresh perspective to other areas it touches. In particular, the computing field can bring a fresh perspective to instructional design.

Secondly, the nature of the content of computer science clearly incorporates both knowledge and skill aspects of expertise. In many other disciplines, either knowledge or skill will typically overshadow the other. Good instructional practice will entail addressing both needs. Thus, computer science educators should be particularly able to contribute to instructional improvement.

Thirdly, software development processes are explicit. Thus, they are readily amenable to examination and possible exportation as procedures for curriculum design. Such processes can be readily shared. Additionally, design processes in computer science appear simpler and more direct than those suggested in the
curriculum development literature. The abstracted software development process described here is adequate to guide curriculum development. That abstracted process is relatively free of jargon, philosophical or political concerns, and managerial aspects. In contrast, the curriculum literature has much that, while important for K-12 educational leaders, seems of marginal use to those actually designing and implementing instruction in college level computing.

Finally, the history of expert systems suggests that if anyone could come to understand the instructional process, computer scientists could. Expert system development requires collaboration of knowledge engineers and domain experts and often “neither the problem nor the knowledge required to solve it is precisely specified” (Buchanan & Smith, 1989, p. 179). Additionally, “commercial developers of expert systems [often] report that one major benefit of building a system has been that they...better understand the problem” (p. 179). Bringing the techniques used in expert system development to bear on instruction could enhance the understanding (and performance) of instruction and learning.

7. CONCLUDING THOUGHTS

This paper has suggested that the methods used in computer science for software development should be applied to the practice of computer science education. This claim was supported in several ways:

- Showing that the software development life cycle for software systems could be applied both to general curriculum design at the discipline and programme levels and to instructional design at the course level.
- Applying the proposed development system to a CS1 course in order to illustrate the process and the use of performance objectives instead of knowledge specification.
- Suggesting that the software process maturity framework could be analogous to an instruction process maturity framework to be developed by the computer science education community.
- Noting potential advantages of the computer science discipline over other fields for improving the instructional process.

These efforts illustrate how using the tools of the computing discipline might enhance instruction. The discussion here is meant to initiate further discussion. These ideas are even richer than this brief presentation might
imply – education/learning is complex and any serious examination of it will be complex. Education is as pervasive as computing. It makes sense for computer science educators to pay as much attention to instructional practice as computer science practitioners pay to software practice.

Progress in the practice and understanding of instructional delivery demands an approach to instructional design, delivery, and evaluation that is as disciplined as that used for software process. Computer science is uniquely suited to achieve substantial progress in understanding and improving instruction. It would make sense to first show how this can be accomplished within computer science education and then generalize that success to other subject areas.

Finally, the better one understands an activity the better it can be performed. This is true for software development. It is true for instructional development. Perhaps the greatest reason to spend more time and effort in designing computer science curricula and courses can be inferred from Simon (1996):

One can envision a future, however, in which our main interest in both science and design will lie in what they teach us about the world and not in what they allow us to do to the world. Design like science is a tool for understanding as well as for acting. (p. 164)

By designing instruction more explicitly and examining the design process, instructors will better understand instruction and, thus, instruct better. Practitioners in computer science are well positioned to apply their knowledge and skill in design to instruction and, as a result, to enhance instruction for computer science students. That same design practice or expertise could be exported to those in other disciplines who want to improve the practice and understanding of instruction in their fields and, as a result, to enhance instruction for their students.

Teaching is a design process. Computer scientists are design specialists. Computer science educators can perfect, for instruction, a design process similar to software design. That instructional design process can be exported to other disciplines to improve instruction for all students at all levels.

ACKNOWLEDGMENTS

I wish to thank my wife, Katheryn, the anonymous reviewers, and the guest editors of this special issue who saw something useful in a very rough submission and made excellent suggestions for improvement. I hope the final paper did not disappoint them.
REFERENCES


Copyright of Computer Science Education is the property of Swets & Zeitlinger, BV and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.