Instructional Design is to Teaching as Software Engineering is to Programming

AUSTIN CORY BART, Virginia Tech
JAVIER TIBAU, Virginia Tech
ELI TILEVICH, Virginia Tech
DENNIS KAFURA, Virginia Tech
CLIFFORD A. SHAFFER, Virginia Tech

Instructional Design is the systematic design of instruction, a pragmatic but principled approach to an age-old problem of how to optimally promote measurable learning. There are striking parallels between Instructional Design practices and Software Engineering practices, such as Test-Driven Development, Requirements Engineering, and Iterative Development. In fact, Instructional Design has been influenced by Software Engineering, and yet Software Engineering Education does not seem to be influenced by Instructional Design.

Instructors in Computer Science concerned with excellent teaching are often aware of educational theories of `learning` such as Constructivism and Situated Learning, but may not be familiar with the more practical theories of `teaching`.

This paper introduces a popular Instructional Design model (the Dick & Carey Model). We attempt to distill the core ideas and principles of Instructional Design into a set of simple practical strategies that instructors can immediately start using. The paper also walks through all 10 phases of the process, and is meant to be a meaningful introduction to the technique. Although there are many possible variations and alternate models possible when designing instruction, the core ideas of the process can be helpful for novice and expert designers alike.

CCS Concepts: •**Social and professional topics** → **Model curricula;**

General Terms: Design, Human Factors

Additional Key Words and Phrases: Instructional Design, Teaching, Software Engineering, Dick & Carey

ACM Reference Format:
Austin Cory Bart, Ishita Ganotra, Eli Tilevich, Dennis Kafura, and Clifford A. Shaffer, 2015. Instructional Design is to Teaching as Software Engineering is to Programming. *ACM Trans. Comput. Educ.* 1, 1, Article 1 (January 2015), 21 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Instructional Design is the systematic design of effective Learning Experiences. Its history stretches back into the 1940s. However, there has been few detailed, reported cases of its usage in Computer Science Education. This paper explains Instructional Design by relating it to Software Engineering.

It is our hope that readers will adopt or be influenced by the model in order to approach their courses with the same rigor they apply to software development.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship, Grant No. DGE 0822220
Author's addresses: Austin Cory Bart, Computer Science Department, Virginia Tech.

DOI: 0000001.0000001

1.1. Target audience of this Paper

The primary audience of this paper are educators and educational researchers specifically in Computer Science. We believe that instruction meant for students of any age group, both in the classroom and in industry, can benefit from the formal methods of Instructional Design. This is true whether these methods are applied strictly or just used as guiding principles. Although many approaches exist, each can suffer from different problems. Computer Science is a dynamic field that is still evolving. In some cases, instructors will be called to teach completely new material or develop a course completely novel to an institution. Whether designing a learning experience from scratch, or improving an existing experience, Instructional Design can provide a systematic process.

1.2. History of Instructional Design, comparison of models

Instructional Design has a rich history that stretches further back than Software Engineering. In the early decades of the twentieth century, it became clear that more systematic and formalized methods were needed if teaching was to ever become a modern field, mirroring the need in Computer Science for formal methods and rigor. In the 1950s, the concept of spelling out exactly what learners were going got be able to do after instruction was popularized by Mager, known as “Learning Objectives”, “Learning Outcomes”, or “Learning Goals”. [Mager 1984] This objectives could classified into different domains or levels, as was done by Bloom in his Taxonomy or Gagné in his Learning Domains. The 60s saw the rise of principled assessment, tied firmly to learning objectives. Glaser published papers relating learning objectives to “criterion-referenced Measures”. Finally, in the 1970s, the first formal Instructional Design models appeared. These included the ADDIE model, the Dick & Carey model, and the Guaranteed Learning model. Instructional Design models would continue to evolve and grow over the next few days, especially with the introduction of new technology and delivery mechanisms. In modern times, Instructional Design is most formally used in industrial training settings, although its principles pervade education.

1.3. Models of Instructional Design

In this paper, we largely refer to the Dick & Carey model of Instructional Design, originally published in 1978 but continually updated and refined since then [Dick et al. 2005]. Figure 1 gives an overview of this model. The advantage of the Dick & Carey model is that it is a teaching model – steps that are natural and implicit for an expert instructional designer are spelled out at each step. In some ways, the Dick & Carey model might be considered comparable to the Waterfall, with such a strong emphasis on the design up-front and the importance of planning; however, it is more comparable to modified forms of the Waterfall model that allow discontinuous phase transitions and iterative development, acknowledging the realities of the engineering process. There are many other popular instructional design models that vary in their dispositions and affordances. For example, the ADDIE model is less exhaustive, and has its roots in military training. The AGILE model is taken from the Software Engineering model of the same name, and shares its philosophy of evolutionary development and rapid response to change. An interested reader is encouraged to read further on these models and find one that suits their style.

1.4. An Instructional Designer

An Instructional Designer is expected to be the advocate for the learner. They are, first and foremost, dedicated to providing an optimal learning experience. The instructional designer may or may not be responsible for actually physically teaching the learners

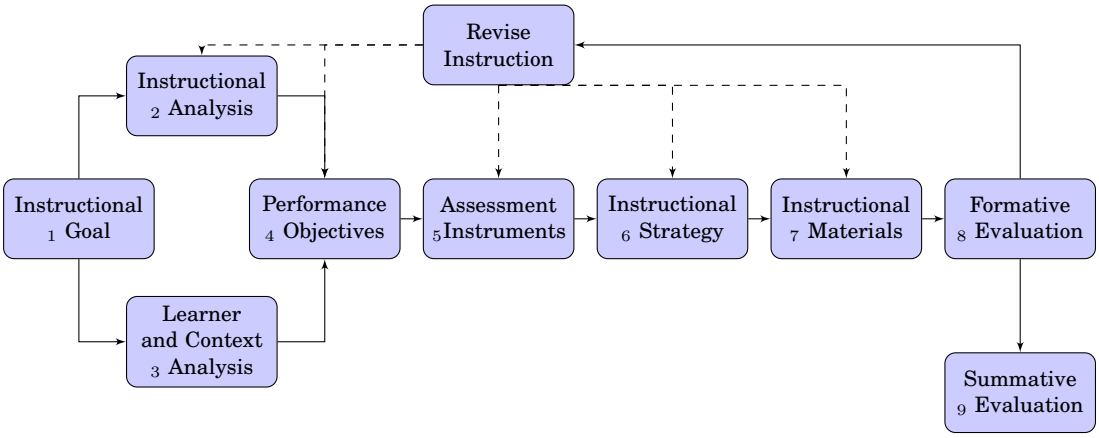

Fig. 1. The Dick & Carey Model of Instructional Design

(indeed, some instructional materials are designed without any physical instructor at all).

1.5. Varying Levels of Instruction: Classes, Units, and Courses

Throughout this paper, we refer to courses, learning experiences, and other units of instruction. Instructional Design is flexible and can operate at any level of instruction, although it is perhaps most directly applied to singular learning experiences meant to fit into a single lesson (e.g., a classroom session). In many cases, applying the principles to a coarser grained unit, such as an entire semester-long course, is simply a matter of iterating through a series of individual lessons. In this paper, we do not cover any formal differences in designing instruction for longer curricula, but we do believe that the concepts are adaptable.

2. PRACTICAL TECHNIQUES OF INSTRUCTIONAL DESIGN

Becoming a professional instructional design is an entire career. However, we believe that there are practical techniques and methods that instructors of Computer Science can start using immediately. Throughout this section, we attempt to relate these techniques to relevant concepts in software engineering, in order to make them clearer and their utility more convincing.

2.1. Measure Your Learners ($\approx$ Unit Testing)

In software Engineering, testing is used to measure the correctness of code. There are unit tests for assessing individual code components, integration testing for assessing the system as a whole, operations testing for assessing the system in production-like environments, and many more variants. Similarly, in Education, testing is used to measure the performance of the learners. Further, assessment can happen at many different levels: in relation to a lesson, to a course, or even to an entire degree of study. Assessment is crucial in both Software Engineering and Instructional Design for understanding the state of the code or learners. However, testing is also crucial for measuring *change*.

If you inherited a large code-base and started committing bug fixes, how would you know if you fixed any bugs? Unless you run unit tests before and after the bug fixes, it would not be possible to know whether you had made an impact. A simple educational fact is that different students start with different initial performance levels and grow

at different rates. Comparing students' performance only at the end of the semester means comparing very different things - without a baseline, it is impossible to get accurate data on the impact of your instruction. This is true within a single course and across semesters. For this reason, Instructional Design emphasizes the role of pre- and post- assessments as a key element of measuring student progress: the relative difference of their performances over the semester.

$$Gains = \frac{Post - Pre}{100\% - Pre}$$

In a major journal paper summarizing the results of the multi-year Media Computation project, Guzdial suggests that they should have been measuring learning *gains* instead of learning *outcomes*.

A proper pretest is administered before students have an opportunity to engage with the instructional materials. At the course-level this usually means the start of the semester, and at the lesson-level it means the start of the class. The pretest should include every single question (unmodified) from the post-test, but can also include entry skill questions. Instructors may find this surprising – if the pre-test and post-test do not match, then the validity of the learning gains can be called into question. An exception to this rule are questions that involve formula or rule application, which can vary in their details as long as the same intellectual skills is being assessed each time. Entry skill questions help instructors ensure that all students entered with the expected prerequisite abilities.

Left to their own devices, instructors may be hesitant to use pretests with their instruction. There are arguments against doing so. Deploying a pretest properly means doubling the amount of time spent on assessment, at the cost of time spent on lecture or other activities. Instructors might fear that students will become anxious if given a test on material they have not been taught. They could also fear that students' performance on the post-test could be affected by seeing the exact same questions on the pretest.

Instructional Designers argue that assessment should not be seen as a "cost", but as a valuable part of the learning process. During a pretest, students get exposure to the kinds of problems they will have to solve, making what they are expected to do much clearer. This is similar to how test-first development can be viewed as a powerful tool for better understanding the problem the future code is meant to solve. Most instructors already spend far longer on lecture and presentations than is needed, as will be discussed in a following section. Care must be taken when explaining the format and process of the pre-test to students, so that they are not overwhelmed – it is common to make pretests graded for completion or effort instead of strictly. Avoiding anxiety must be balanced against leaving students complacent, requiring instructors to have as clear an image of their learners as possible. Finally, as to the fear that students may perform better only because they've already encountered the question, it should be reminded that students should also have encountered the same questions during practice opportunities (in-class activities, homework, projects, etc.). Students should never be surprised on a post-test. In the end, if an assessment is properly designed, then there should be no problem with how students learned, as long as you can prove that they did.

Another advantage of using a pretest is to give personalized or targeted instruction. Identifying the students that need extra attention early-on is challenging, but a pretest can suggest students that need extra attention. Of course, the chance of false positives is very high, similar to how the number of failed test cases for a function does not perfectly predict how hard it will be to debug that function. Besides targeting individual learners, the pretest can also reveal a class' misunderstandings or intuitive knowledge about specific topics. It is possible that administering a pretest will reveal that stu-

dents already have an understanding of some material, and that further instruction on that particular topic is not necessary.

2.2. Align Your Course Components ($\approx$ Software Development Lifecycles)

Software development has a clear idea of a "Life Cycle", a structured sequence of development phases to guide development. Instructional Design is also a systematic process, partitioned into stages with meaningful ordering. An experienced developer writes unit tests before code, and plans their architecture before writing unit tests. Similarly, instructional designers begin developing new learning experiences by writing their assessments before their lessons, and their learning objectives before their assessments. Novice educators must resist the urge to start making PowerPoint slide decks, just as junior developers have to resist the urge to start writing code before they've planned out their program. Developers give care and attention to these processes to make sure that the end product meets expectations. It is easy to write code without knowing the goal, but it rarely leads to a successful program. Similarly, it is very easy for lesson plan development to go offtrack without careful planning.

Figure 1 demonstrates one particular Instructional Design models' process, although there are many possible arrangements. However, ultimately, some activities happen before others. Crucially, planning activities such as writing learning objectives or analyzing the learners happen early on, actual development of assessments and lessons happens in the middle, and evaluations happen at the end. Typically, these models assume that revision and iterative development will occur, rather than a definitive end. Software is never truly complete, and lessons can always be improved.

Writing learning objectives can seem like a painful activity that should only be done when forced by ABET, but its advantage should not be underestimated. Clearly laid out learning objectives are beneficial to both the developers and the learners. It is easier to keep on track mentally if the goals are known. Assessments can be matched directly against learning objectives, ensuring that full coverage of the objectives is reached. This is similar to reviewing user requirements to ensure that all scenarios are covered by the planned software. Clearly laid out assessments lead to a clear instructional strategy, since the designer knows exactly what the learners need in order to succeed. Although this can lead to teaching to the test if taken to the unhealthy extreme, it can also result in very focused teaching with minimal distractions. Finally, evaluations of the instructional experience can be done with a clear knowledge of the final desired state.

Sometimes, designers have a clear idea of a particular architectural pattern they want to use, or a particular lesson that they want to create. For expert developers, it is sometimes appropriate to skip formal methods in favor of focusing valuable time and effort where it is most needed. Alternatively, learning objectives may only be sketched out loosely or assessments relegated to half-formed ideas. There are many dangers to skipping careful planning; as the old joke goes, "weeks of programming can save you hours of planning." This ultimately needs to be left to the judgment of the instructional designer.

Although the flow of development is presented here as strictly linear, sometimes it does make sense to skip or return to another phase of development. Perhaps while developing instructional materials, you realize that you left out a crucial learning objective that you felt was important. Or perhaps while building your assessments, you realize you missed a key assumption about the entry skills of your learners. It is important that if a change is made to an already completed step, that the implications of that change flows to subsequently completed steps. In the example of the new entry skill, you would want to ensure that your assessments now cover the skill, for instance.

2.3. Follow Best Instructional Patterns (≈ Design Patterns and Architectural Paradigms)

Software Engineering does not dictate what programming language to use, whether you should follow functional or object-oriented paradigms, or whether you need to use a long-polling architecture or a publish/subscribe architecture. Using analysis techniques from Software Engineering may lead you to find that, for a given context, one of these techniques is superior to the other, but there is nothing inherent in Software Engineering that prohibits or inhibits using these more tactical-level techniques. In the same way, Instructional Design is a theory of pedagogy, not a theory of learning, and is compatible with a wide range of theories and educational strategies.

Instructional Design does not strictly demand a particular teaching style, although many models offer suggestions or templates on how to build individual lessons. For instance, the Dick & Carey Model advocates for designing lessons around Gagné's Nine Learning Events, given in figure 3. Briefly, these events include an introduction to the material, the presentation of the material, an opportunity to practice the knowledge gained, assessment of the learner, and a conclusion to promote transfer. The plan for a developed learning experience can be compared against each of the events. Critics of Gagné's Learning Events suggest that the theory is grounded too firmly in Behavioralism, although Gagné was later influenced by Cognitivist principles [Molenda 2002]. Variations on the Nine Learning Events exist that reflect a more Constructivist style, with more emphasis early on towards participation with less presentation. Other instructional design models provide other templates for learning experiences.

There is an expectation in the Dick & Carey Model that learning experiences will begin and end with assessments (pretests and post-tests). There are courses where this might be considered inappropriate. Studio and design-based courses at the advanced level may not wish to hamper students' creative process. On the other end of the spectrum, introductory courses meant for younger learners may find it too motivationally prohibitive to incorporate serious assessment. It is the job of the designer to determine when to follow patterns and when to make the hard decisions about breaking from a convention. As software engineers, we have to become used to balancing factors beyond what we were taught in our classes – thinking beyond space and time costs to consider impacts on security, code readability, and development time, for instance.

2.4. Evaluate Your Assessments (≈ Quality Assurance)

Code Coverage is an important part of Software Testing, used to measure the effectiveness of a test suite. This metric is not perfect, but gives a rough idea of where to focus attention.

Similar techniques and methods exist for assessments and resources developed with instructional design.

At a high level, all instructional materials need to be evaluated with real learners to identify defects. These evaluations can range from individual sessions with learners to large field trials, and can yield varying levels of quantitative and qualitative evidence about the instruction.

However, assessments in particular can benefit from statistical analyses such as Item-Response Theory to reveal inadequacies and validate the models. Consider what makes for a good exam. Based on the learning objectives established during the early phases of the Instructional Design process, the most important content is selected and exam questions are created. We hope and expect that students will answer these questions correctly if and only if the students perfectly comprehends the corresponding content. As a whole we then have a few evaluation criteria that we use to define a "good test".

— The class median and grade distribution looks what we expect (e.g., a normal curve).

— The time to take the test is appropriate for what time we allocate.
— Students appear to understand the question (its not ambiguous or misleading).
— People don't complain to much (which is primarily dictated by good results on the three points above).

But is this good evaluation criteria for an exam? Not really, according to formal methods.

Assessing correctly is very hard. Concept Inventories are assessment instruments that become well-validated by external experts and can be considered robust

2.5. Course Documentation ($\approx$ Design Documentation)

If a fellow Software Engineer handed you a zip file of undocumented source code, you would probably find it difficult to navigate and debug the repository. It is about as helpful to only receive a collection of slides.

Developing a learning experience results in a number of artifacts.

Some of these artifacts are meant for end-user learners: powerpoint slides, assignments, and assessments. However, other forms of documentation can also be very valuable.

Following an Instructional Design process can result in useful documentation on learning objectives, learner analyses, and instructional strategy. Although students never see this information, it can be crucial context for other adopters.

Programmers document code not just for other people, but also for themselves. Have you ever reviewed older course materials that you created, and were unable to understand why you made a particular decision? Commenting your learning resources can be just as valuable as commenting code.

Organizing documentation artifacts can be a struggle, especially with multiple instructors collaborating to develop a course. Discipline is required throughout the lifetime of the instructional design process. Taking advantage of version control software such as Git can make it easier later to review the changes made to a course, to distribute materials, and even push and pull content from external adopters. Still, there is significant overhead in using such technologies, further taxing what can already be a tedious process of documentation.

3. A FORMAL MODEL FOR INSTRUCTIONAL DESIGN

In this section, we will describe each phase of the Dick & Carey model in more detail, following the order shown in figure 1. For each phase, we will contextualize the description with examples and suggest related techniques from Software Engineering. The goal of this section is to give practitioners a more detailed look at how to use Instructional Design, but is not meant as a complete explanation of all the nuances involved. For more in-depth reading, we recommend "The Systematic Design of Instruction" [Dick et al. 2005] and "Principles of Instructional Design" [Gagne et al. 2005].

The examples used in this section are drawn from our own experiences using Instructional Design to improve our teaching. We will refer to two major scenarios:

(1) Improvements made to a university-level, semester-long course on Computational Thinking for non-majors. By the time the authors started applying Instructional Design techniques to improve the course, it had already been created and offered several times.
(2) The creation of a high school level, week-long summer workshop on Computer Science. The authors had great freedom to create and establish curricula, although they were under severe time and resource constraints to develop the materials.

The artifacts and results of these experiences are not meant as a primary contribution of this paper, although the curriculum materials are freely available. Instead, they are meant as guiding examples and context to give a clearer picture as to how instructional design is conducted in realistic scenarios.

3.1. (1/9) Identify Instructional Goal

The first step in the Dick & Carey Model is to identify the instructional goal(s) that you want to accomplish. Critically, the goal does not describe anything about how the instructor will teach the material – only what learners are expected to know and are able to do at the end of the instructional experience. This goal can come from a variety of sources, either top-down or bottom-up. For instance, the core four learning objectives for our Computational Thinking course came from a university initiative, although it was up to the course instructors to decompose these objectives into finer-grained objectives. Alternatively, these goals might be determined completely by the instructor, as is what happened in the Computer Science workshop. Many professional organizations and colleges establish objectives according to a more national agenda, such as the Computer Science Curricula project [ACM/IEEE-CS Joint Task Force on Computing Curricula 2013]. Goals can also come from a specific, perceived gap in learners existing ability. In the Computational Thinking course, new learning objectives were

In the full Dick & Carey model, a *Needs Assessment* is conducted to determine if instruction is truly needed. For instance, instead of teaching employees how to use complex software, the user interface of the software could be streamlined instead. In the example of the Computational Thinking course, students were struggling with the chosen IDE; instead of training the students, a simpler IDE was chosen instead. Because instructional design can be a very time and energy consuming process, it can be valuable to find evidence (either quantitative or qualitative) of the need for instruction to better focus resources.

A complete goal statement should describe:

(1) The learners
(2) The performance context
(3) The action that learners should be able to do in the performance context
(4) The tools that are available to the learners in the performance context.

A major, common misstep in writing an instructional goal is to create a *fuzzy goal*, usually describing an abstract, unobservable outcome using verbs such as “appreciate” or “know”. To correct a fuzzy goal, the designer should begin by writing down the goal. Then, they list examples of what people do to demonstrate that they have achieved this goal. The designer can use this list to iteratively develop more concrete goals that will lead to observable outcomes.

In the Computational Thinking course, we wanted students to better understand variables and program state. The instructors and teaching assistants of the course repeatedly mentioned this as one of the weakest areas for students, despite being one of the most important conceptually. Many students struggled to write basic programs because of their inadequate understanding of how to use variables. Although there were no assessments at this point that could provide definitive evidence, the questions from office hours and one-on-one sessions made the need clear. The goal statement created for this learning objective reads as follows:

(1) **Learners:** An undergraduate, non-Computer Science major...
(2) **Performance Context:** ... Given source code...
(3) **Action** ... Will be able to identify the name, value, origin, and type of a variable at any given step within the code...

Fig. 2. The Abridged Skill Analysis Diagram

(4) **Tools** ... Taking advantage of any resources from their integrated debugging environment.

3.2. (2/9) Conduct Instructional Analysis

The next step in the model is to conduct an Instructional Analysis, which will spell out exactly what the successful learner will do in the performance context. Like the Instructional Goal, it does not describe how to teach the material, although it is easy for product-oriented thinkers to accidentally start diagramming their lesson plan instead. The Instructional Analysis first requires the designer to classify the goal statement based on the the kind of learning it requires. Then, the designer identifies and organizes the *subskills* involved in the skill being learned, and places them into an Instructional Analysis Diagram.

Dick & Carey describe 4 learning outcomes, based on the work by Gagné. **Verbal Information** requires the simplest form of learning: it represents simple, declarative knowledge to be memorized and recalled (e.g., "What is the worst-case runtime of the major sorting algorithms?"). Very little presentation is usually needed for Verbal Information, although students may find it useful to practice such material. **Intellectual Skills** are more complicated problem-solving tasks that demand some level of deeper cognition (e.g., "Write a program to compute the sum of a list of numbers"). Many types of knowledge in Computer Science require Intellectual Skills, but range greatly in complexity. **Psychomotor Skills** are much rarer in Computer Science, describing the coordination of mental and physical activity (e.g., "Type the alphabet using a keyboard"). **Attitudes** is the fourth type, describing goals where the learner makes a choice (e.g., "Choose to persevere in the face of segfaults"). Gagné also identified a fifth domain, **Cognitive Strategies**, but Dick & Carey includes this as a particularly ill-structured form of Intellectual Skills. Readers familiar with Bloom's Taxonomy will find these outcomes similar – mappings exist between these lists, suggesting their rough equivalence. It is up to the designer to choose their preferred schema. In both the Computational Thinking course and the Computer Science workshop, many different kinds of learning outcomes are necessary: verbal information related to basic facts of computing, intellectual skills related to the nature of programming, and even attitudes related to fostering domain identification in Computer Science.

Once the goal is classified, the instructor performs a Subskill Analysis to concretely describe the goal and its components. Intellectual Skills and Psychomotor Skills is a typically straightforward process of creating a flowchart that breaks down each step into multiple levels. Much as you would decompose the steps of a program, you decompose the skills into increasingly simpler steps. Unlike a programming language, however, it is strictly up to the designer to decide when to terminate the iterative decomposition. It is also possible to have branches and cycles, as is typical for flowcharts. Some of the most basic subskills can be considered "Entry Skills" – the learner can be expected to already have mastered them before the instruction begins. For instance, in the Computational Thinking course, basic keyboard use (a psychomotor skill) and computer literacy (intellectual skills) are expected. Substeps might also involve Verbal Knowledge, which is denoted with a triangular "V" node. Verbal Information is much simpler to diagram, although the designer can cluster the topics if they want; in the Run-time Analysis example from before, for example, you might group the algorithms by their Big O values. In the formal methods, Attitudinal Skills can be considered a special form of the other skills: to resolve it, identify a skill that the behavior in the goal is most akin to, and elicit students to demonstrate that behavior. In our experience, teaching and measuring attitude goals was tricky and was not worth the effort.

The instructional goal for teaching variable tracing, used in the previous section, is best classified as an Intellectual Skill. Figure 2 is an abridged version of the Instructional Analysis Diagram created for the goal. The complete diagram is available in the online appendix. Although the end product could benefit from further iterations, and other instructors might have their own opinions on the result, the clear outline of the material provided a clarity to designing the instruction.

3.3. (3/9) Conduct Learner and Context Analysis

During the next phase, you will need to identify properties about the learners, the learning context, and the performance context. There are many factors involved in learner and contexts analysis. Some of these factors will be obvious and immutable, but others might require investigation or mutable. Some factors will be useful and have great influence on your decision decisions, but some may be irrelevant. In collaborative instructional design, they are useful to make sure the entire team has a clear picture of the learners and their needs.

Dick & Carey suggests 8 crucial factors for understanding your learners: (1) their entry skills (the previously described abilities relevant to instruction), (2) prior their knowledge of the topic area, (3) their attitudes toward content and potential delivery system (e.g., whether they prefer web-based or in-person instruction), (4) their general academic motivation, (5) their educational and ability levels, (6) their general learning preferences, (7) their attitudes toward the organization giving the instruction, and (8) group characteristics (e.g., is the group of learners heterogenous or homogenous?). In some cases, the designer will intuitively know this information. In others, it is necessary to investigate through surveys and interviews. It is possible that this data cannot be collected until the course has already started, such as through a pre-test, or even collected at all. While unfortunately, designers should plan accordingly to collect the most meaningful data possible and make reasonable inferences where they can not.

In the Computational Thinking course of our case study, it was difficult to characterize the average learner and their performance context. Students enter the course with limited prior experience with computation and, in many cases, low self-efficacy. They can be expected to have entry skills related to basic computer usage (e.g., keyboarding, browser usage), but even these are varied. In this case, it is helpful to know the distribution of the learners' characteristics, including what "typical" learners look like, and also what kinds of outlier students there are. For instance, although most students have low prior experience, every semester there have been a handful of students who have prior experience or particular aptitude for computing, making most of the material very easy for them.

Conducting the performance and learning context analysis is a simpler task, but still important. The 4 criteria for the performance context are: (1) what supervisor support is present, (2) physical aspects of the site (e.g., tools, facilities, etc. that are available on the site), (3) social aspects of the site (e.g., coworkers, underlings, etc. that are available on the site), (4) relevance of skills to the workplace (described as a "reality check" by Dick & Carey – how will they use what they learn?). There are 4 similar criteria for the learner context: (1) compatability of site with instructional requirements, (2) adaptability of site to simulate workspace, (3) adaptability of delivery approaches, (4) learning site constraints. The reader might note a strong emphasis on the concept of a workplace, which stems from Instructional Designs' widespread use in industry. If this emphasis is unpalatable, you can consider the performance context as a more abstract environment.

It is also possible that it is impossible to consider any kind of "typical" performance context, if you are dealing with very diverse learners. In the Computational Thinking course, for example, we are dealing with students from many different disciplines and

with many career paths. In the Computer Science workshop, the performance context is even more vague and nebulous, since the goal was to simply give an introduction to the material. For the purposes of our instructional unit, we largely ignored the performance context and constrained the learning context to be remote and online, in order to deliver the content outside of the existing course time line. For upper-level in-major courses, or for more practical material, it becomes more valuable to consider the performance context. For instance, when teaching collaborative strategies in a senior-level software engineering course, it may be useful to model a performance context and simulatory learning context.

When conducting a learner and context analysis, we recommend on identifying the most salient features and not worrying about exhaustively modeling learners. In particular in Computer Science, we are often most concerned with prior knowledge and motivational factors such as Self-efficacy, as opposed to students' preferences for a particular learning delivery system. However, it is up to the strategic instructional designer to decide what is relevant and not relevant to their process. Our complete learner and context analysis is available in the online appendix, arranged in tabular form for convenience, and may serve as an example of what sort of questions to consider when conducting such an analysis.

3.4. (4/9) Write Performance Objectives

Performance objectives, also known as behavioral objectives, establish clear and precise goals for learners. These objectives are similar to those created during Goal Modeling in requirements engineering. Although Performance Objectives demand structure and rigor, taxing an already constrained instructional designer, they can be helpful for keeping the development process grounded and on-track. Additionally, good objectives lead very naturally to assessment instruments and lesson plans, simplifying later design steps. If the objectives are used as part of the instructional materials, it is even possible that students can directly benefit from them; some studies have found that simply eliciting clear learning objectives to learners can improve their understanding and aid their cognitive strategies [Torrance 2007].

Writing performance objectives is a simple, but somewhat repetitive, process. For every skill and subskill, the designer defines three parts of a performance objective: 1) the conditions that the learner will perform under, 2) the behavior that they should exhibit, and 3) the criteria that they will be judged against. Objectives are specific and detailed, avoiding language that applies equally to any objective. Furthermore, they use verbs that describe an observable action, rather than ethereal verbs such as "know" and "understand" – if you cannot collect observable evidence for the objective, the objective serves no purpose. When iterating through the subskills, you can start with the Instructional Goal and proceed in a depth-first exploration of the graph.

In our example of tracing variables in the Computational Thinking course, a concrete terminal performance objective for the instructional goal might be:

(1) **Conditions:** Given some source code in python and any resources from the internet...
(2) **Behavior:** ... learners will list the name and origin and trace the value and type of all the properties in the code ...
(3) **Criteria:** ... of every variable in the code

Alternatively, consider node 1.1.A, "State the rules for naming variables":

(1) **Conditions** Given some source code in python...

(2) **Behavior** ... learners will state the rules for naming properties...
(3) **Criteria** ... in their entirety.

An experienced educator has probably encountered goal and objective writing while dealing with accreditation processes. These experiences may have left a sour taste in their mouths that they are not eager to deal with again. We encourage, as always, to consider the trade-offs in the process and the product – instructional design should never get in the way of designers, but serve as a tool. Yet, we suggest against immediately dismissing performance objectives, since they can be powerful tools for organizing the course content, especially when collaborating or disseminating results.

3.5. (5/9) Develop Assessment Instruments

There are four kinds of assessments in the Dick & Carey Model, roughly partitioned by time. We have already discussed the pretest and posttest, which are respectively at the beginning and the end of the learning experience. There are also practice assessments, which are meant to be used after student has received some amount of instruction but before the posttest. In all three of these instruments, there should ideally be perfect alignment – whatever is on the pretest shows up on the practice test, and whatever is on the practice test shows up on the posttest. The only exception is for questions which involve application of formula, which can vary the specific values used. The more closely the pretest and posttest are aligned, the more confident the designer can be that they successfully measured the change in their learners' knowledge. The final type of assessment is the Entry Skills test, which is usually combined with the pretest. These questions are meant to confirm the prior knowledge of the learners. Ideally, every student will achieve a perfect grade on these types of questions.

In Education, there are many different types of assessments possible. Multiple choice and true/false are advantageous because they are easy to grade, although they are more shallow forms of measuring knowledge about a question. Essay questions are opportunities to gather more in-depth knowledge about students' understanding of a concept. Short answer and fill-in-the-blank may be a preferable balance between these two extremes. For further information on developing assessments following classical forms, we refer the reader to the "Educational Assessment of Students" [Nitko 2001]. There are also forms of assessment that are relatively unique to Computer Science. Students can be tasked with tracing code, predicting output, or even writing and testing algorithms. Although many subjects have portfolio reviews, Computer Science in particular can benefit from code reviews and having students walk through their generated computing artifacts. When choosing assessments, carefully consider the trade-offs between the potential depth of the students' responses, the expense of grading, and the cost of developing the assessment.

When designing the lesson on tracing variables for the Computational Thinking class, our main performance objective was assessed by having students trace out variables. However, the subordinate performance objectives was assessed through a mixture of other assessment instruments. For instance, to assess the recalling of variable naming rules, they were given a multiple-multiple choice question to select all of the valid variable names.

According to the Dick & Carey model, designing assessment instruments happens prior to deciding the expected delivery mechanism of the instruction. However, it is unwise to not have some consideration of the constraints of their potential environments. In particular, if a digital assessment is planned, then there should be concerns about what resources students will have access to, what proctoring will be necessary, and what the affordances of the platform are.

Preinstructional Activities.
. *(1) Gain Attention:*. Stimulate the learner and motivate them to engage with the material.
(2) Describe Objectives:. Inform the learner will be able to do because of the instruction.
Presentation Activities.
. *(3) Stimulating Recall of Prior Knowledge:*. Remind students of their prior knowledge.
(4) Present the Material:. Using text, graphics, etc., give the instructional materials.
Learner Participation.
. *(5) Provide Guidance:*. Give instructions on how to use the instructional materials.
(6) Elicit Performance:. Have the learners actively apply the new knowledge.
(7) Provide Feedback:. Show the learner what they did well and poorly on.
Assessment.
. *(8) Assess Performance:*. Evaluate the learner on their performance
Follow-through.
. *(9) Enhance Retention and Transfer:*. Generalize the instruction by showing similar applications.

Fig. 3. Dick & Carey's Instructional Strategy Model interleaved with Gangé's Nine Events of Instruction

3.6. (6/9) Develop Instructional Strategy

The Instructional Strategy is a high-level plan for the instructor to use when deploying instructional materials. Developing an Instructional Strategy has strong parallels to developing an architectural diagram or otherwise planning out the organization of a program.

The first task when developing an instructional strategy is to sequence and cluster the relevant performance objectives, much as you would build sequence or component diagrams. The goal is to group logically connected performance objectives and to build a progression that learners can follow naturally. In most cases, this sequence will match the order of the objectives, and the clustering will be fairly discrete. However, this is an opportunity to use pedagogical content knowledge to recognize that some information, despite being distinctive content knowledge, is more easily understood in the context of other information. For example, clustering the rules for scheduling algorithms together might help highlight differences and make them easier to remember, even if the skill analysis did not group these rules. Sequencing and clustering is highly situational and will be influenced by the needs and judgments of the instructional designer.

Just as there are design patterns and programming paradigms in Software Engineering, there are well-known templates of organization for Instructional Design. These templates act as guides for the creation of new content, or as checklists for existing content. Just as the Model-View-Controller pattern suggests how to organize code to avoid certain pitfalls and avoid missing a key element, so to do models like Gagné's Nine Events help designers ensure they have achieved all the necessary components of a successful learning experience. But just like software patterns, there are opportunities to diverge in favor of practicalities.

Figure 3 interleaves Gagneé's Nine Learning Events [Gagne 1985] with the five components of a learning experience according to Dick & Carey. This model is adaptable to different instructional styles and can respond to different learning theories. For example, to adapt for more Constructivist approaches, one could swap steps 4 and 6, so that students have the opportunity to engage with the materials before receiving instruc-

tion. Whether designing constructivist or cognitivist learning experiences, it is likely that certain events will be present, no matter the order. Note also that although this model is shown as an ordered sequence, some steps might be repeated and cycled between. For instance, very negative feedback might prompt the learner to return from step 7 to step 6 until they are able to achieve some level of mastery.

Dick & Carey strongly recommends planning the Preinstructional Activities, Assessments, and Follow-through before planning the Presentation and Participation elements, to assure optimal alignment. They list a series of criteria to consider for each of these components, although some need more consideration than others. For instance, at this point in the instructional design process, the assessments have largely been planned, and only the mechanics need to be planned (Will it be open-note? Can students work in groups? Will it be administered on paper?). The Follow-through of the learning experience consists of devising Memory Aids (physical artifacts that students can keep to aid them later, such as a cheat-sheet showing programming syntax) and promoting transfer (e.g., by showing exactly how the learned skills can relate to their final project or future careers).

The Preinstructional activities are a more complicated aspect to consider. First, plan to motivate the learners; Dick & Carey suggests using a motivational framework such as the ARCS model (gain Attention, establish Relevance, promote Confidence, and provide Satisfaction) [Keller 1987]. In Computer Science education, motivational issues often relate to anxiety and self-efficacy, but there are many other concerns. The analysis conducted in phase 3 of the learners can be valuable for planning this stage. Next, list the relevant learning objectives developed in the earlier phases. Third, remind the learners of the expected Entry Skills so that they are prepared to call on those skills, or to review that material as necessary. The Preinstruction is often overlooked by novice designers, to the detriment of the learning experience. Of course, Cognitive Load Theory and a mental model of the learner should also be kept in mind; many students will often skip the instructions for an assignment, no matter how well you explain them. Still, well-written objectives and instruction can be useful for students to review as they get mired in a complicated assignment, so even if they do not read them initially, it does not mean that they will not be useful eventually.

The last two components of the Dick & Carey instructional strategy model are the presentation and participation components. These components are repeatedly instantiated for all of the clustered and sequenced group of objectives. Notice that participation is inherent in this model, in line with modern theories of active learning. It is already well-established in the literature that Computer Science Education benefits tremendously from such active learning techniques [McConnell 1996], and there is a considerable number of ways to incorporate such activities. Even a lecture has many opportunities to foster participatory discussions or for students to respond to simple questions. In upper-level courses, where students can be expected to be more auto-didactic, there are more learning objectives, and less time available, there might be less time to commit to participatory activities. Still, the designer should carefully consider the trade-offs and how crucial participation and feedback is to the learning process; they should probably consider giving up on learning objectives before they consider short-changing the learning experiences.

For the Computational Thinking course's learning experience on tracing variables, we planned our Presentation activities to have short videos about the material followed up by short, computer-graded quizzes that would allow students to interact with the material. For the Computer Science workshop, different lessons were planned with very different kinds of activities. For the second day of instruction, concerning the performance objectives about writing and debugging algorithms, students were presented with a short (10-minute) lecture on the key components of computational algorithms

and then instructed to write a paper-and-pencil algorithm to sort a deck of cards. On the third day, concerning the performance objectives about writing concrete programs, students were presented with the syntax of a simple block-based programming language, and then tasked with writing programs to accomplish simple goals (e.g., escape a maze). Both of the participatory activities for these days were taken from other sources, a practice heavily endorsed by both instructional design and software design.

3.7. (7/9) Develop and Select Instructional Materials

Instructional Materials is a broad phrase that covers everything from the powerpoint slides to handouts to seating charts. It is difficult to describe exact plans for developing instructional materials, but there are many resources for specific strategies for activities such as designing effective lectures, creating educational videos, etc. As the Designer moves through phases 5 and 6 (developing the instructional strategy and then the instructional materials), they move from design to concrete development. This decomposition of the high-level strategy into implementation mirrors the same process in software development. In theory, this is a smooth one-way transition. In practice, whether writing codes or lesson plans, it is an iterative and chaotic process as unaccounted-for complexity creeps in.

Instructional Design Models such as Dick & Carey usually delay the design of instructional materials till the end of the entire process, focusing initially on analysis of the learners, the performance context, the instructional goal, and a host of other critical factors. At that point, a content delivery system is chosen based on its educational affordances and suitability for the learning experience being developed. This delay reflects the typical role of an Instructional Designer as a *consumer* of instructional technology – they survey the available techniques and technology (e.g., traditional lecture, essay writing, websites), and choose the one that fits their need. It is possible that Computer Science Educators can also be *providers* of instructional technology, although they should consider this role very carefully. It is always possible that the tool that they want already exists, or that an alternative teaching strategy could make it unnecessary.

There are many existing repositories of instructional materials. A few will be listed here, potentially, like CS Teaching Tips, the ECS project, OpenDSA, CITIDEL.

3.8. (8/9) Conduct Formative Evaluation

Once all of the instructional materials are completed, it becomes time to formatively evaluate them. There are three kinds of formative evaluations: 1-1, where the designer sits with three students representative of the class (i.e. one low, one medium, and one high performer); Small Group, where the materials are distributed to a collection of students and they complete them more realistically, without interaction with the designer; and Field Trial, where the actual instructor uses the materials in an authentic learning context (e.g., the regular classroom). The types of evaluation to use depends on the designers confidence in their materials and how much revision they expect they will need to make. New materials are usually best subjected to all three phases, although this can be time-intensive and resource-intensive.

To ensure proper scientific protocol, the procedure should be planned out before the evaluation is conducted. The use of tables and charts with specific questions can help guide the designer and make sure they gather useful data. These questions might be about student attitudes, such as “did you feel confident answering when answering questions on the tests”. They might also be more cognitively-oriented, such as “did you understand what you were supposed to learn”.

The 1-1 evaluation is the most personal evaluation, with the instructional designer sitting adjacent to the learner while they complete the materials. The designer should

not assist the learner, but should instead ask questions to get a sense of their thoughts and reactions. The Small Group and Field Trials are less personal, but benefit from an increased sample size. It is typical to include qualitative surveys in the Small Group and Field Trial evaluations in order to gather more detailed information about the motivational and practical aspects of the materials.

Figure 4 shows the results of the two formative evaluations conducted for the Computational Thinking class, a 1-1 (N=3) and a Small Group (N=5). The instruction was revised based on the results of the 1-1, before it was distributed to the Small Group. It is important to understand that these results are not meant to show conclusive effectiveness of the instruction, but largely meant to spot weaknesses in the material. However, the results are promising, showing large gains in the students learning from pre- to post-. Qualitative results were also gathered during this process, revealing much more finely grained problems. For instance, the learners in the 1-1 suggested that the audio in the videos was poor, and that they were particularly confused by the text in one of the assessment questions. Once this material was refined, students in the Small Group trial reported a different, lesser set of problems.

Phase	Student	Pretest	Posttest	Gains
1-1	#1	37.8%	100%	100%
1-1	#2	51.8%	100%	100%
1-1	#3	35.7%	84.6%	76%
SG	#1	21.7%	93.3%	91.4%
SG	#2	54.6%	100%	100%
SG	#3	64.3%	91%	74.8%
SG	#4	32.7%	80%	70.3%
SG	#5	69.4%	91%	70.6%
	Average	45.9%	92.5%	86.1%
	StdDev	7.5%	16.6%	9.8%

Fig. 4. Student Performance during Formative Evaluation (N=8)

The Computer Science workshop was planned in a timeframe that did not allow for either 1-1 evaluations or Small Group; instead, the materials were piloted "live" as a field trial. Although this gave much richer data, it also meant that it was impossible to revise the materials. This is one possible explanation why the learning gain results were somewhat less favorable between these two case studies. Of course, the nature of these two learning experiences are so different, they are essentially apples and oranges: the Computational Thinking unit on tracing variables was highly targeted, whereas the Computer Science workshop had much broader and high-level objectives (e.g., complicated subjects such as algorithms and abstractions) condensed into a short relatively timespan.

This is the phase of the instructional design process that can benefit from applying techniques such as Item-response theory. This part will be written later.

3.9. (X) Revise Instruction

It is very rare that a formative evaluation reveals no problems with instruction. Embedded into the Dick & Carey model is the assumption that designers will want to revise materials before moving onto more summative evaluations. Of course, throughout the process, the designer may choose to return to an earlier phase and make revisions. There is nothing wrong with correcting mistakes or making improvements, as long as the designer considers the downstream implications of such changes. Deleting a performance objective, for instance, might free up a considerable amount of time to

improve another performance objective, but it might also server as a prerequisite to another performance objective. Still, iterative development is crucial no matter what kind of development is happening. Neither software nor instruction is ever "done" – it is simply "good enough".

3.10. (9/9) Conduct Summative Evaluation

When materials have reached sufficient maturity, an instructor may wish to have them disseminated for broader use. A Summative Evaluation can be conducted by an independent third party on the validity of the objectives, instruments, and materials. Historically, summative evaluations have shifted from comparisons of innovations (is a given instructional strategy superior to another strategy?) to demonstrations that the learners can perform in a performance context. The dearth of concrete, monolithic performance contexts in Computer Science education can make it difficult to conduct summative evaluations. If a lesson on the use of iteration in algorithms is developed, how can this be related to the students' future careers? Each student may be destined for a very different path in a very different subject, and may not encounter the taught skills in any shape close to the one taught.

This case study did not involve a Summative Evaluation, as is common for many instructional design projects. In fact, typically summative evaluation is most useful for training in industrial scenarios, rather than education in more scholarly settings. Still, the two types of summative evaluations might have some merit if used strategically. The first type of summative evaluation is an Expert Judgment evaluation, where an acknowledged expert evaluates the materials for fitness without interaction with the target learners. Ideally, this expert has both content knowledge and pedagogical content knowledge. The second type of summative evaluation is the Impact Evaluation, where the effect on the job-site is used to determine if the organizational or institutional needs have been met. In a university setting, this might entail looking at long-term graduation results, job-placement data, or exit interviews and examinations. Although a properly-conducted summative evaluation can provide compelling evidence for the success of instructional design, it can be very difficult and ultimately worth less than spending time on other phases or even other projects.

4. BENEFITS OF INSTRUCTIONAL DESIGN

Instructional Design can be a time-consuming process, but the benefits can outweigh the potential disadvantages. In this section, we outline some advantages and hazards of the instructional design process.

4.1. Fault localization

A good instructor might be able to intuitively identify problem areas in a course. Proper assessment measures learning gains rather than absolute knowledge, making it easier to evaluate the instructional materials.

4.2. Assessment Evaluation

Many instructors consider assessment as an afterthought, deployed at the end of the course to . There are mathematical techniques such as Item Response Theory that can be used to increase the coverage and validity of the assessment instruments.

4.3. Course Documentation

While developing according to Instructional Design processes, a natural side-effect is the documentation of instruction. This goes beyond the instructional materials to include the instructional strategy, the assessments, the learning objectives, and all other components.

4.4. Replication

Few courses are built from scratch. In most cases, course replication means using another instructor's slides. There is no explanation of the decisions that went into development of these course materials, no strategy for how to use them in a greater course context, and no basis on how to modify them for a different group of learners. Consider inheriting a codebase with no documentation or supplementary information.

4.5. Ellicited objectives

Research has shown that merely listing course objectives for students can measurably improve their performance on assessment. Seeing the exact learning objectives they are responsible for can aid students' meta-cognitive strategies. This also aids the transfer process.

4.6. Templated Instruction

Instructional Design provides a model instructional strategy with five key components. Constituent in this model are opportunities for students to participate actively with materials and receive feedback.

4.7. Compatability with other techniques

As CS instructors grow professionally, they encounter new learning and teaching techniques to guide their decisions. Instructional Design is a framework that is compatible with a range of popular theories, including constructivism, peer instruction, active learning, and many more. At its heart, instructional design is an approach to developing lessons, but it only offers suggestions on where to go.

For instance, active learning is inherently suggested in the model instructional strategy; the Constructivist approach is a modification of that strategy where participation precedes content presentation. Situated Learning Theory dictates authentic assessments where learners focus on performance of tasks rather than artificial completion of, for example, written questions; Instructional Design does not require any particular assessment tools, and would suggest simply ensuring a concrete rubric (of an appropriate open-endedness) exists for the assessment activity.

4.8. Learner Analysis

In Usability Engineering, the users' needs and desires are given primacy, carefully considered before any code is written. Instructional Design similarly recommends analysis of the learners and environment before any instructional strategies or materials are developed.

In both Usability Engineering and Instructional Design, it can become apparent early in the analysis phase that new software or instruction is not necessary. Using a Needs Assessment, the designer can determine if there is a performance gap that can be remedied with new materials. In many cases, changes to the tools, management, and other factors can yield desirable results without as much work. Of course, we expect many readers to be in learning situations where they have a clear need for instruction.

5. POTENTIAL DISADVANTAGES OF INSTRUCTIONAL DESIGN

5.1. Over-rigorous

The Dick & Carey model is a teaching model, used to train new instructional designers. It is one of the most formal methods of instructional design, with rigorous amounts of structure and a strong sense of linearity. Although the entire model is iterative, the purest execution is intended to be done in disjunct phases. Each one of these individual

phases demands extensive consideration and the elicitation of many details, some of which may feel repetitive or unnecessary for an instructor. Identifying all the learning objectives in a course can lead to a staggering amount of previously implicit information. A highly detailed skill analysis leads to many performance objectives and triple the number of assessment instruments. Although this is benenficial in a spirit of optimizing the course materials, it can be detrimental to an instructional designer with time and energy constraints.

As budding instructional designers move through their zone of proximal development, they are expected to apply formal methods more strategically. In practice, many designers rely on more stream-lined models, a mish-mash of their favorite models, or even to simply rely on their own intuitions. This is similar to how Software Engineers develop intuitions about their development methods.

Some instructors may find that they work best by adopting principles or components of the instructional design model, rather than a whole-sale immersion. We encourage trial runs and .

5.2. Over-assessment

Although Instructional Design does not demand a minimum amount of assessment, it is implicit that learners will be evaluated summatively. This is not necessarily compatible with all learning scenarios. Younger learners in K-12 contexts may not be sufficiently motivated to participate in activities that they know will be graded. It is up to the instructor to make decisions Some instructors may not feel it is appropriate to give pre-tests to students, if they feel that the leaners have no possibility of prior exposure and could be intimidated by such an assessment.

The nature of instructional design leads to tight, focused lessons where the materials and assessments are directly aligned. Some may feel that the assessment materials are over-focused, leading to a ”teach-to-the-test” mentality. This may encourage negative performance-oriented behavior in learners Even worse, exams carry emotional weight for students, and they may react badly if they feel test exhaustion.

5.3. Over-structured

Much like programming, there is an element of art to teaching. Introducing rigor can have implications for implicit and variable aspects of a lesson.

that secondary implicit learning objectives may fall by the way-side.

Further, tightly focused lessons can limit students' perspective. If the education is of a sufficient quality, they may believe that they are receiving the whole story. Negative and limited learning experiences

Finally, there are viable concerns about spoon-feeding

6. DISCUSSION OF THE AUTHORS' EXPERIENCE WITH INSTRUCTIONAL DESIGN

A time log was kept throughout the development of the Computational Thinking courses' unit on Tracing properties. Over the course of 3 months, a total of 24 hours and 20 minutes was spent by the lead instructional designer on developing materials. The lead designer was attempting to follow the Instructional Design process as formally as possible, with close and careful attention paid to element. In retrospect, this is not a sustainable use of development time, given that all of the original learning goals were not event met.

When applying the Instructional Design techniques to the summer workshop on Computer Science for high schoolers, a looser process was used. Although learning objectives were produced and developed into assessments, much less attention was paid to filling out extensive charts detailing learner characteristics or exact performance objectives. These modifications can be related to Agile design methods, with a focus on

developing a prototypical product rather than exhaustively following the process. Although the results from the evaluation of the Computer Science workshop were not as positive, they are promising for a first draft. However, from the designers' point of view, the best part of the results is not that they were positive (which they largely were), but that they had measurable results at all. Subsequent iterations of the workshop can now be compared and referenced to each other, benefiting from both a pedagogical perspective and from a research perspective.

7. THE IMPACT OF SOFTWARE ENGINEERING ON INSTRUCTIONAL DESIGN

Although Computer Science Education has been largely unaffected by Instructional Design, Instructional Design is aware of and been affected by principles of Software Engineering. [Ardis and Dugas 2004] relates the principles of "Test-first Design" in Extreme Programming to the early assessments advocated in the Dick & Carey model, coining the term "Test-first Teaching". However, they fail to draw deep comparisons between software engineering processes and instructional design processes, instead serving mostly as an account of Software Engineering instructors as they use the Dick & Carey model to design a new course. [Tripp and Bichelmeyer 1990] introduces the idea of Rapid Prototyping as an evolution of the principles of Instructional Design, inspired directly by the same principle in Software Engineering. Ian Douglas has published two papers on applying Software Engineering techniques to Instructional Design, one relating Object-oriented language paradigms to reusable learning materials [Douglas 2001] and another, more summarizing paper that focuses on advances in Software Engineering that relate to ID, such as rapid prototyping. Notice that all four of the above papers were published in educational journals rather than computing or computing education journals.

The IEEE Professional Development site contains a series of pages on using Instructional Design, meant for novice instructors.

Going the other way, there is a body of research on how Software Engineering techniques can be applied to ID [Douglas 2001; 2006]. Within CS, what little research that exists focuses on pedagogical tools and tactics [Damian et al. 2006; Hadjerrouit 2005] rather than holistic, cohesive processes.

8. IMPLICATIONS FOR EDUCATIONAL SOFTWARE DEVELOPERS

9. CONCLUSIONS

Todo

10. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship, Grant No. DGE 0822220

APPENDIX

Some more appendix information

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

Acknowledgements go here.

REFERENCES

ACM/IEEE-CS Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013*. Technical Report. ACM Press and IEEE Computer Society Press. DOI:http://dx.doi.org/10.1145/2534860

MA Ardis and CA Dugas. 2004. Test-First Teaching: Extreme Programming Meets Instructional Design in Software Engineering Courses. 34th ASEE/IEEE Frontiers in Education Conference, Savannah, GA. *IEEE Computer Society* (2004).

Daniela Damian, Allyson Hadwin, and Ban Al-Ani. 2006. Instructional Design and Assessment Strategies for Teaching Global Software Development: A Framework. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. ACM, New York, NY, USA, 685–690. DOI:http://dx.doi.org/10.1145/1134285.1134391

Walter Dick, Lou Carey, and James O Carey. 2005. The systematic design of instruction. (2005).

I. Douglas. 2001. Instructional design based on reusable learning objects: applying lessons of object-oriented software engineering to learning systems design. In *Frontiers in Education Conference, 2001. 31st Annual*, Vol. 3. F4E–1–5 vol.3. DOI:http://dx.doi.org/10.1109/FIE.2001.963968

Ian Douglas. 2006. Issues in Software Engineering of Relevance to Instructional Design. *TechTrends* 50, 5 (2006), 28–35. DOI:http://dx.doi.org/10.1007/s11528-006-0035-z

Robert M Gagne. 1985. *The conditions of learning*. Holt, Rinehart & Winston, New York.

Robert M Gagne, Walter W Wager, Katharine C Golas, John M Keller, and James D Russell. 2005. Principles of instructional design. (2005).

S. Hadjerrouit. 2005. Learner-centered web-based instruction in software engineering. *Education, IEEE Transactions on* 48, 1 (Feb 2005), 99–104. DOI:http://dx.doi.org/10.1109/TE.2004.832871

John M Keller. 1987. Development and use of the ARCS model of instructional design. *Journal of instructional development* 10, 3 (1987), 2–10.

RF Mager. 1984. Preparing Instructional Objectives (2nd edn), Pitman Learning Inc. (1984).

Jeffrey J McConnell. 1996. Active learning and its use in computer science. *ACM SIGCSE Bulletin* 28, SI (1996), 52–54.

Michael Molenda. 2002. A New Framework for Teaching in the Cognitive Domain. ERIC Digest. (2002).

Anthony J Nitko. 2001. *Educational assessment of students*. ERIC.

Harry Torrance. 2007. Assessment as learning? How the use of explicit learning objectives, assessment criteria and feedback in post-secondary education and training can come to dominate learning. 1. *Assessment in Education* 14, 3 (2007), 281–294.

StevenD. Tripp and Barbara Bichelmeyer. 1990. Rapid prototyping: An alternative instructional design strategy. *Educational Technology Research and Development* 38, 1 (1990), 31–44. DOI:http://dx.doi.org/10.1007/BF02298246

Received February 2007; revised March 2009; accepted June 2009