Applying Formal Models of Instructional Design to Measurably Improve Learning in Introductory Computing

Austin Cory Bart working under Eli Tilevich, Clifford A. Shaffer, Dennis Kafura

Instructional Design

- "The systematic design of learning experiences that give measurable results by following a well-defined process." [1, 3]
- A theory of teaching, orthogonal to theories of learning.
- Highly comparable to Software Engineering, but predates it
- Applications in higher education, industrial training, and K-12 education

Computer Science Education

- Most prior research suggests how Software Engineering techniques can be applied to Instructional Design [2, 5].
- The remainder focuses on pedagogical tactics rather than a holistic life-cycle. [4]
- Core Assumption: Most CS Educators don't use formal methods when developing their instruction.

Research Questions

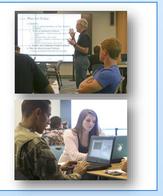
1. What does the Instructional Design life cycle look like in practice for Computer Science Education?
2. What are the advantages and disadvantages of applying Instructional Design to CS Education?
3. What practical actions should a CS educator take based off Instructional Design best practices?

Methodology

Practical case studies applying the model

1. Understanding variables in Computational Thinking course for non-major students
2. Designing a 5-day workshop on Computer Science for rising high-schoolers

Review literature, observations, reflections



(1) Identify Instructional Goals

What is the learner able to do after instruction?

Remember · Understand · Apply · Analyze · Evaluate · Create

Verbal Information
- Define abstraction
- List control flow structures

Intellectual Skills
- Analyze a program's runtime
- Trace variables in code

Cognitive Strategies
- Explore topics in Compilers

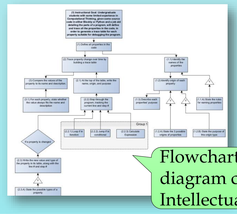
…

Avoid fuzzy goals (e.g, "understand abstraction") by using observable verbs

Establish domain/level of knowledge, a la Bloom's Taxonomy or Gagne's Learning Domains

(2) Conduct Instructional Analysis

Decompose the expert performance into observable steps.



Not a list of teaching steps!

Flowcharts to diagram complex Intellectual Skills

1. List all the variables used in the code
   1. Identify the names of the variables
      1. State the rules for naming variables (Verbal information)
   2. Identify binding of each variable…

(3) Analyze Learners and Contexts

Understand the learners and environments as well as you can.

Characteristic	CT Tracing	CS Workshop
Age	Freshman-Senior	High-school Juniors
Gender	50% male/female	75% male/female
Majors	Mostly non-STEM	STEM
Interests	Varied	Unknown – survey
Self-efficacy	Low	Medium
…	…	…

Distribution, averages, variances, and outliers

Helps coordinate designers

Establishes what we know and also what we don't know

(4) Write Performance Objectives

Judgement criteria for each goal and subgoal, moderated by the learners traits.

Conditions	Given some source code in python…
Desired Behavior	… learners will trace the value and type…
Judging Criteria	… of every variable in the code.

Specifies what these learners should be able to do, as opposed to an expert

Conditions	Given the term "Abstraction"
Desired Behavior	… learners will be able to give a definition…
Judging Criteria	…. Including all components in the rubric

Largely a formality, can be quite tedious

(5) Develop Assessment Instruments

Create pre-, post-, and practice assessments to measure learners.

	Generic	Specialized
Manual	• Essay • Short response	• Program creation
Automatic	• Multiple-choice • True/false • Fill-in-the-blank • Apply formula	• Trace execution • Structured program creation

(1) The following is valid python code:

```
words = "something"
a_list = add_data("list data")
for banana in smoothie:
    if banana > 5:
        print(words)
```

List the names of all of the properties in the code:

Same test for the pre and post.

Balance deep assessment with rapid feedback

(6) Develop Instructional Strategy

High-level "architecture" of your instruction, matched to models

Preinstructional Activities
- Motivate Lesson
- List Objectives
- Stimulate Recall

Assessments
- Pre? Post? Entry skills?

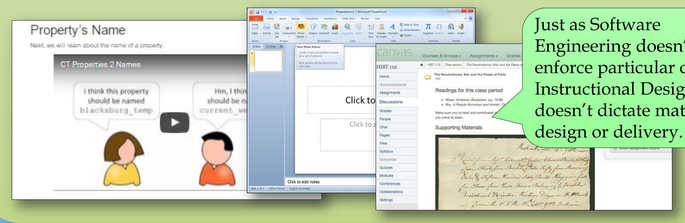
Follow-through
- Hand-out? Transfer?

Presentation	Participation
Watch an introductory video on variables	Answer multiple choice questions on properties
Watch a video on variable naming rules	Circle all variables in a program
…	…

Organize based on learning theories

(7) Develop Instructional Materials

Actual development of necessary presentations, hand-outs, etc.



Just as Software Engineering doesn't enforce particular code, Instructional Design doesn't dictate material design or delivery.

(8) Conduct Formative Evaluation

Gather data using 1-1, small groups, and field trials to evaluate the materials.

Qualitative survey and interview results
- Confusion about "identifying purpose" of a variable
- Some of the videos ended too abruptly
- Video audio was too low
- The recap at the end was very helpful

Learner	Pre	Post	Gains
1-1: #1	.378	1	**1**
1-1: #2	.518	1	**1**
1-1: #3	.357	.846	**.761**
SG: #1	.217	.933	**.914**
SG: #2	.546	1	**1**
SG: #3	.643	.910	**.749**
SG: #4	.327	.800	**.703**
SG: #5	.694	.910	**.706**
Average	**.460**	**.925**	**.825**

Quantitative data results, suitable for analysis, e.g., Item-Response Theory

(9) Revise Instruction

Based on the data gathered, iterate on the materials, assessments, objective, etc.

Planned Revisions
- Remove learning objective about "explaining a variable's purpose"
- Move assessment to an automatically graded system
- Re-record audio on videos
- …

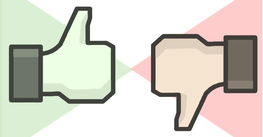
(10) Summative Evaluation

Use external experts and impact analysis before dissemination

Largely unused in academia due to the difficulty in conducting longitudinal impact analyses and the limited dissemination.

Potential Trade-offs

Measured gains: Very clear and obvious improvements or performance
Focused instruction: Strong connection between assessments and instruction
Early Errors: Designing before developing catches some kinds of problems early
Simplified replication: Improves dissemination, revision, and collaboration



Too much assessment: Recognize diminishing returns and testing fatigue
Railroading pedagogy: Stay flexible and open to revising your materials
Time-consuming: Short-term increases to workload, over 24 hours and 30 pages in one of our case studies. Moderate the process vs. the product.

Practical Takeaways

Instructors should…
1. Focus first on what learners can do after instruction
2. Plan tests before development to focus instruction
3. Match your instructional strategy to design patterns
4. Evaluate your materials for improvements, quality
5. Document your process

Instructional Design has potential as a technique for rigorously improving and evaluating instruction, and merits further study.

References

[1] Dick, W., Carey L., and Carey, J. O.. "The systematic design of instruction." (2005).
[2] Douglas, I. "Instructional design based on reusable learning objects: Applying lessons of object-oriented software engineering to learning systems design." Frontiers in Education Conference, 2001. 31st. Vol. 3. IEEE.
[3] Gagne, R. M., Wager, W. W., Golas, K. C., Keller, J. M., et al. (2005). Principles of instructional design.
[4] Hadjerrouit, S. "Learner-centered web-based instruction in software engineering." Education, IEEE Transactions on 48.1 (2005): 99-104.
[5] Tripp, S. D., and Bichelmeyer, B.. "Rapid prototyping: An alternative instructional design strategy." Educational Technology Research and Development 38.1 (1990): 31-44.
This material is based upon work supported by the National Science Foundation Graduate Research Fellowship, Grant No. DGE 0822220