Digital Support for Abductive Learning in Introductory Computing Courses

Atanas Radenski
Chapman University
Orange, CA 92869, USA
1-714-744-7657
radenski@chapman.edu

ABSTRACT
Students who grew up browsing the Web are skilled in what is usually referred to as *abduction*, a reasoning process that starts with a set of specific observations and then generates the best possible explanation of those observations. In order to exploit the abduction skills of contemporary students, we have developed digital CS1/2 study packs that promote and support active learning through abduction, i.e., *abductive learning*. The study packs integrate a variety of digital resources: online self-guided labs, e-texts, tutorial links, sample programs, quizzes, and slides. These online packs stimulate students to learn abductively by browsing, searching, and performing self-guided lab experiments. In two years of study pack use, the failure rate in the CS1/2 courses at Chapman University has been reduced from 14% to 5%. The study packs have been published online at studypack.com and adopted in various institutions.

Categories and Subject Descriptors
K.3.2 **[Computers & Education]**: Computer and Information Science Education - *computer science education, curriculum*; Computer Uses in Education - *distance learning*

General Terms
Human Factors, Experimentation

Keywords
Active learning, abduction, CS1/2, laboratory, Python, Java

1. RATIONALE
A printed textbook is the principal learning resource required by a computing course instructor. Yet, there are reports that textbook popularity among teachers and students may be declining. A recent survey - conducted in six European universities in Germany, Iceland, Finland, Romania, and Latvia - reveals that teachers rate programming textbooks as the least beneficial learning resource [7]. According to the same survey, students also rank textbooks low - together with lecture notes, exercises, still pictures, and interactive visualizations. In contrast to textbooks, sample programs are considered the most useful learning resource by both students and teachers who participated in the survey [7]. In our own surveys conducted three times at Chapman University in Southern California since 2004, CS1/2 students consistently rate paper textbooks among the least helpful resources, together with paper lab manuals [8]. Our students identify sample programs as most beneficial, exactly as European students do [8].

In informal forums, various educators have expressed concerns about the increasing unwillingness of students to systematically read textbooks. The following message that was published by a computer science professor and a department chair in the SIGSE mailing list in April 2006 addresses this problem. "Alas, I find that one of the biggest challenges is the increasing inability (or willingness) of students to read. Witness my Alice lab this semester where a student looked at a page in the textbook (only half a page of text, since the top half was a picture, and large print at that), sighed, and said 'I'm just going to muddle through with the software because I don't have the patience to read these [ed: 4 or 5] sentences'." [1]

Along of the decreasing popularity of textbooks, the efficiency of traditional classroom teaching may also be decreasing. Survey results reveal that lectures are considered the least useful learning activity by both students and teachers [7]. Our own experience is that it is indeed becoming increasingly difficult to keep students' attention during classroom presentations, especially when students have Web surfing opportunities during class.

Low ratings of textbooks and lectures can possibly be attributed to various reasons. We believe that such low ratings can be caused by discrepancies between (a) the teaching preferences of textbook authors and instructors on one end, and (b) the learning preferences of students on the other. While textbooks happen to focus extensively on general concepts and paradigms, entry-level computing students normally prefer to learn by concrete examples and experiments. For instance, many computing textbooks lead instructors to lecture on abstraction early in their classes, whereas beginning computing students often fail to connect to these concepts. This issue is illustrated by the statement posted by an educator in the SIGSE mailing list. "Increasingly I find [that] student's [sic] ability to understand, let alone write, even slightly abstract statements of any kind about programs is seriously deficient. All that many can relate to is numerous examples of program behavior." Some educators even question students' willingness and ability to reason about the subject matter: " …students today are looking for the quickest way to solve their problem … -- 'just tell me how to solve it now as I do not want to (nor do I know how to) think'" [1]. We believe that contemporary

computing students do reason as well as their professors do, but in a different way, as advocated in the rest of this section.

Most academics, by virtue of their profession, are masters of sound *deduction*, a reasoning process that involves the inference of conclusions from general premises. Because most academics are at their best when they derive the specific from the general, they like to read and learn this way, and they often choose to write textbooks and to teach this way.

Contemporary computing students grew up browsing and reading on the Web rather than reading paper books alone. On the Web, they have learned to surf rapidly from one piece of information to another, to relate a variety of observations, to search useful facts quickly, and to make relevant conclusions. Such students are particularly capable of *abduction*, a reasoning process that starts with a set of observations and then generates the best possible explanation of those observations (see also the Abductive reasoning Sidebar).

Abductive reasoning occurs naturally in most disciplines. For example, abductive reasoning in a computing class may begin with the study of a sample program then continue with some experimental changes to the program, and finally generate plausible explanations of various algorithmic and linguistic program features. When a computing student says “I will figure it out by myself”, he is likely to apply abductive reasoning – by first browsing through concrete samples and then finding the best possible explanation for what has been observed.

Serious difficulties can appear when computing textbooks and instructors put the stress on deduction when computing students are best at abduction. Such discrepancy between teaching styles and learning habits can generate frustration for both educators and students. The result can be poor retention and considerable enrollment decline in beginning computing courses.

As educators, we do not have the ability to change the reasoning and learning habits of entry-level computing students. A wise and more efficient approach is to accept students' learning preferences as a given and to adapt computing course contents and activities accordingly. We believe that entry-level computing courses can be adjusted to the learning preferences of contemporary students by providing substantial support for abductive reasoning. Deductive teaching methods can be beneficial if used cautiously.

We introduce the general term ***abductive learning*** to refer to any method of active learning that targets abductive reasoning. More specifically, abductive learning is based on activities that are intended to trigger abduction. Recall that abductive reasoning generates the best possible explanation of a set of observations. Consequently, abductive reasoning is a learning process by itself because students actually learn the generated explanations.

Active learning in general, and *abductive learning* in particular, deviate from traditional lectures and reading and involve learning by doing (physical action) and by thinking about what has been done (mental action). Active learning techniques are well supported by technology and are successfully applied in both core and in advanced computing courses.

In order to exploit the abduction skills of entry-level computing students, we have developed original CS1/2 online study packs that support abductive learning through integrated digital resources and activities. In the rest of this paper, we (1) describe the implementation of the digital study packs, (2) depict several abductive learning techniques as implemented by the study packs, and (3) present evaluation of the digital study packs and the positive effect of this teaching method on student enrollment.

Abductive Reasoning Sidebar
Since Aristotle, logic arguments have been commonly divided into two subclasses: the class of deductive arguments (necessary inferences) and the class of inductive arguments (probable inferences). In the second half of the 19th century, Charles Peirce was the first to distinguish between "two utterly distinct classes of probable inferences, which he referred to as inductive inferences and abductive inferences" [4]. Beginning with Pierce himself, researchers have assigned a variety of interpretations of the terms induction and abduction. Most often, *induction* is used to mean a logically unsound inference that generates a likely conclusion about a population - based on observation of a population sample. As originally defined by Pierce, *abduction* is a formally unsound but common inference mechanism that concludes the cause based on the presence of its effect. In the second half of the 20th century, philosophers and artificial intelligence researchers have adopted a broader interpretation of *abduction* as *inference to the best explanation* [6]. What is the *best* explanation depends on the context. Often, this is (1) the most powerful explanation - the one that explains most observations, or (2) the simplest explanation, or some combination of both. Abduction and induction share common features and also have differences. Both are undoubtedly recognized as ubiquitous patterns of reasoning. Abduction and deduction intersect; for example, smart (reasonable, valid, strong) inductive generalizations are treated as instances of abduction [6]. While induction is typically used to generate predictions, abduction is used to generate explanations [3]. Pierce advocated that abduction is the main method for generating new knowledge.

2. IMPLEMENTATION

2.1 Background

We taught deduction-based Java-based CS1 and CS2 courses from 2002 to 2004 at Chapman University. Like many others, we began the CS1 course with the top-level Java concept, the class, and then systematically focused on important fundamental principles throughout the CS1/2 sequence, such as abstraction, encapsulation, and information hiding. We certainly enjoyed our consistent logic explanations for the roots of everything during the entire CS1/2 sequence. Our students hated it, but it took us a while to notice.

Within a two year period, our CS1 enrollment declined by 34%. We did not try to find an excuse in the fact that the enrollment decline was in sync with a widespread enrollment decline in the majority of undergraduate programs in the USA. We analyzed our CS1/2 course experience and came to realize that the majority of our computing students do not comprehend well when taught by deduction, from the general to the concrete. Instead, they learn best in the opposite way: from specific observations to explanations and generalizations.

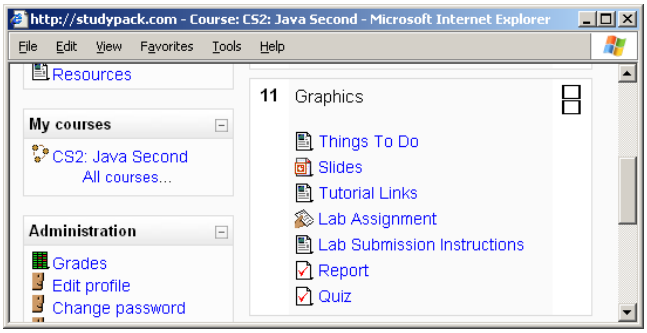

Figure 1. The *Graphics* chapter in the *Java Second* study pack home page

We thus decided to exploit the abductive learning capabilities of contemporary computing students. To achieve this goal, we wrote and published online unique and novel CS1/2 digital study packs that promote and support abductive learning. The study packs are integrated collections of original digital resources, such as e-texts, tutorial links, self-guided labs, sample programs, quizzes, and slides.

2.2 Digital Study Packs

We are among those educators who advocate the use of two different languages in the CS1/2 sequence [8]. Our specific choices are Python for CS1 and Java for CS2. We have found Python beneficial for CS1 because it offers a simple, straightforward kernel that can easily be mastered by beginner programmers. In addition, Python supports an easy-to-use interactive mode that effectively promotes abductive learning. For CS2, we prefer Java because it is a mainstream commercial language that focuses on large scale OO software development. Other language choices can be similarly beneficial. Scheme, Visual Basic and Ruby, for example, are possible choices for CS1, while C++ and C# are often used in both CS1 and CS2.

We use the nicknames *Python First* and *Java Second* for the CS1 and CS2 packs. Each study pack is a collection of online chapters, referred to as *topics*. The home page of the pack contains a list of all chapters, together with links to the main components of each chapter (Fig. 1).

Python First and *Java Second* packs are designed in the same style and use the same layout.

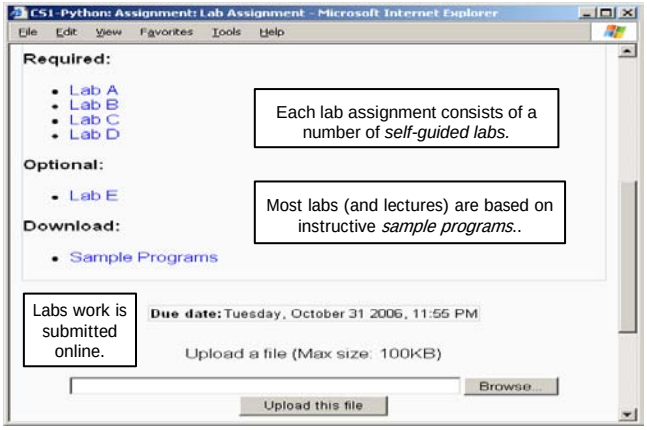

Figure 2. The lab assignment from the chapter on *Strings, Files, and the Web* in the *Python First* study pack

With either pack, the study of each topic is commenced in regularly scheduled classroom *lectures*. Lectures are supported by 730 slides in *Python First* and 600 slides in *Java second*.

After lectures, students work on *lab assignments* (Fig. 2). The *Python First* pack comprises 62 *self-guided labs* and 58 *sample programs*. The *Java Second* pack offers 41 *self-guided labs* and 56 *sample programs.*

Each *Python First* topic includes an *e-text*, which is essentially a chapter from a digital textbook. *Java Second* does not include e-texts but utilizes ready-to-use free online lessons from the popular online Java Tutorials of Sun Microsystems, Inc. [9].

The study of each topic concludes with the student submission of an online *lab report* and *quiz*.

2.3 Abductive Learning with Digital Study Packs: An Overview

As already stated, *abductive learning* is based on activities that trigger abductive reasoning in the learning process. Abductive reasoning begins with a set of observations and results in their plausible explanation. Thus, by eventually finding an explanation the student learns that explanation. The abducted explanation may be initially imprecise or incomplete. It is further refined and corrected through various activities, such as consultations with the instructor, reading e-texts and tutorials, and taking quizzes.

Python First and *Java Second* pack enforce abductive learning by means of the following activities:

- Self-guided labs and sample programs
  - Interactive labs
  - Non-Interactive labs
- Instant reward online lab reports and quizzes

These activities are designed to trigger and stimulate abductive reasoning in the learning process, as discussed in the next sections.

2.4 Abductive Learning with Self-Guided Labs and Sample Programs

2.4.1 Self-Guided Labs

A *self-guided lab* contains sufficient details to allow students to work independently [8]. Self-guided labs incorporate (1) necessary *background information* and (2) detailed sequences of *instructions* that walk students step-by-step through program exploration and development.

Students with little or no preliminary programming knowledge can meticulously follow the detailed lab instructions. This process is likely to bring such students to successful lab completion. Students with some preliminary knowledge can try the lab independently, while consulting the instructions when needed, or even perform the lab without looking at instructions at all. These choices also depend on student motivation and confidence level.

2.4.2 Interactive Labs

By design, Python supports a highly interactive programming style that gives our CS1 students a great opportunity to learn by interactive experiments and exploration. In Python's interactive mode, students can type various statements and immediately

observe and analyze the results from the execution (Fig. 3, left).

By design, Java is less interactive than Python. Yet, many Java IDEs support interactive exploration. Our preferred CS2 choice is DrJava, because its *interactive mode* is very similar, visually and functionally, to the interactive mode of Python (Fig3, right). This similarity contributes to a smooth transition from the CS1's Python to the CS2's Java.

The self-guided *interactive labs* are designed to walk students, independently from the instructor, through interactive experiments. Interactive labs trigger abduction in a straightforward way. Students observe each statement as they type it and then analyze the result of its execution (Fig. 3). When necessary, students browse and search various digital resources, such as e-texts, tutorials, slides, and the Web. In the process, students generate plausible explanations of the form, meaning, and purpose of each interactive statement.

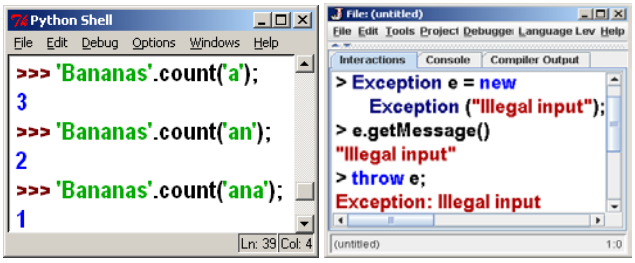

Figure 3. Interactive exploration of Python's string methods (left) and Java's exceptions (right)

For example, a *Python First* interactive lab triggers abduction to teach students that the *count* method does not count overlapping substring occurrences (Fig3, left). As another example, a *Java Second* interactive lab triggers abduction to teach students that an exception object carries its own message, and that the object does not propagate when created, but when thrown (Fig3, right).

2.4.3 Non-Interactive Labs and Sample Programs

A non-interactive *sample program* is intended to demonstrate a new concept or technique. For example, the *Transform Sequence* sample program demonstrates the Java 2D Graphics translation transform by drawing a sequence of rectangles (Fig. 4, top left).

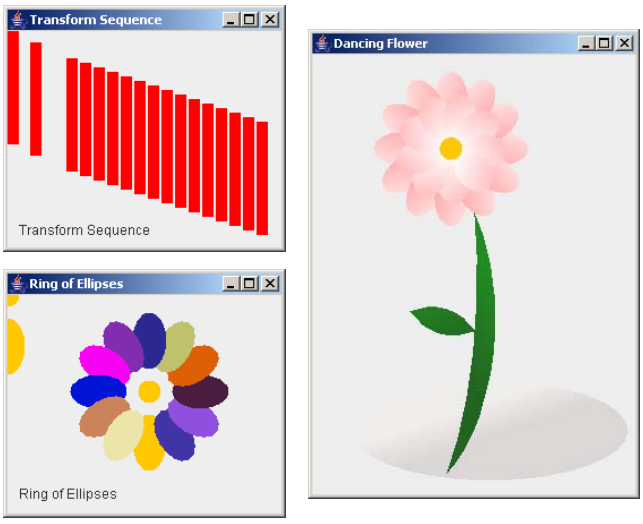

Figure 4. 2D graphics from the *Java Second* study pack

The *non-interactive* self-guided labs are based on sample programs. A typical self-guided lab instructs the student to explore and experiment with the sample program. After that, the lab calls for a transformation of the sample program into a *target program*. Then the lab provides detailed step-by-step guidance for a successful target program implementation. Students are free to follow the detailed instructions or to try the lab independently.

For example, the *Ring of Ellipses* self-guided lab is targeted at a program that paints a ring of randomly colored ellipses (Fig. 4, bottom left) by means of translation and rotation transforms. This self-guided lab is based on the *Transform Sequence* sample program. The *Ring of Ellipses* lab explicitly draws a parallel between the sample and the target programs: the sample program uses repeated translations to paint a sequence of rectangles while the target program should use repeated rotations to paint a flower-like ring of ellipses (Fig. 5). The lab then guides the student, step by step, towards a successful and complete solution.

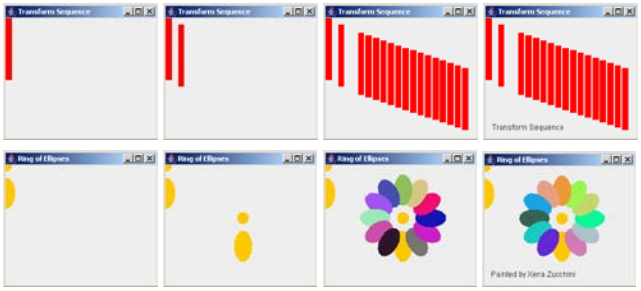

Figure 5. A self-guided graphics lab triggers abductive reasoning by drawing a parallel between a sample program (top) and the lab's target program (bottom)

Non-interactive self-guided labs trigger abductive reasoning by drawing parallels between sample programs and target programs. The study of programs in transition, from samples to targets, is a powerful abductive learning method. In order to understand sample programs and their transformations into target programs, students are motivated to browse and search various digital resources, such as online slides, e-texts, and tutorials. By performing self-guided labs, students generate plausible explanations of the form, meaning, and purpose of various program structures. Last but not least, students obtain sound intuition of *step-wise program development*.

For example, the *Transform Sequence* self-guided lab, as discussed earlier, triggers abduction by drawing a parallel between translation and rotation (Fig. 5). When students follow this lab, they spontaneously and naturally generalize concrete properties of *translation* and *rotation* as properties of the more abstract *transform* concept. Thus, students generalize that transforms in general, not just translation, have cumulative effect on rendering. In addition, students naturally affirm their knowledge of step-wise program development by simply following the recommended lab steps.

Abductive learning through self-guided labs can be very stimulating. Our experience is that students often choose to go beyond what is required by the lab. For example, one of our students in the Spring of 2006 went beyond the required course activities to study constructive area geometry, in order to produce the stem and the leaves of his dancing flower (Fig. 4, right). This student started *Python First* as a dance major and ended *Java Second* as a computer science major.

2.5 Instant Reward Online Activities

In addition to self-guided labs and sample programs, the digital study packs stimulate learning through various online activities, such as lab reports and quizzes. The study packs assign provisional credit instantly, upon completion of lab reports and quizzes. The assigned credit, in terms of scores or grades, is the main measurable recompense, or reward, that students receive for their completed work. Students appreciate receiving their scores and grades as soon as possible.

Note that in the study packs, labs and quizzes are intended to help students learn and prepare well for exams, rather than serve as principal evaluation tools. We prefer offline exams conducted in the classroom as the main evaluation mechanism.

2.5.1 Instant Reward Lab Reports

After having submitted their programs online, students file an *online lab report.* On the basis of the report, the digital study pack automatically grants provisional credit for the lab assignment. Instructors may audit student submissions and possibly adjust provisional credits. Except for occasional audits, the instructor can choose not to grade programs.

Online lab reports stimulate abductive learning because they provide *instant rewards*, in the form of provisional credit to students who have completed their lab work. Instant rewards give students a feeling of success and an incentive to try more labs, to experiment and see what happens. An instant reward from any completed lab gives students an incentive to enthusiastically engage in the next one.

2.5.2 Instant Reward Quizzes that Drive Reading

With digital study packs, the study of each topic is completed with an *online quiz* (Fig. 1). All quizzes allow *multiple submissions*. The opportunity to submit the same quiz multiple times for higher score is a powerful learning stimulator. Students are motivated to make as many submissions as it takes to receive the highest possible score. To improve their scores with additional quiz submissions, students search, browse, and read e-texts, tutorials, and slides. Furthermore, students experiment interactively, observe experimental results and try to find the best possible explanation for what they see. In short, multiple-attempt quizzes drive reading, experimentation, and abductive reasoning.

Online quizzes stimulate learning by providing instant feedback and instant rewards.

2.5.3 Instant Accessibility

In addition to instant rewards, the online study packs stimulate learning though *instant accessibility*. Students do not need to wait for a scheduled class in order to engage in an abductive learning process. With online study packs, students engage themselves in learning at any convenient time and from any convenient location.

3. CONCLUSIONS

This paper introduces abductive learning - a form of active learning that is targeted at abductive reasoning. Many others have already explored a variety of active learning methods that can benefit computing courses (see [2, 5] for examples). Our original contribution is the comprehensive digital study packs that trigger and promote abductive learning by means of novel self-guided labs and a variety of integrated digital resources and activities.

In the 2004/5 academic year, we adopted draft versions of *Python First* and *Java Second* digital study packs in the CS1/2 courses at Chapman University. In two years, the failure rate in the CS1/2 courses fell from 14% to 5% (Table 1).

Table 1. Cumulative CS1/2 enrollment data and failure rates

Academic year	*2003/4*	*2004/5*	*2005/6*
Total enrollment	44	45	74
Passing grades	38	39	70
Non-passing grades	6	6	4
Failure Rate	14%	13%	5%

In surveys conducted at Chapman University in 2005/06, students give high ratings to the CS1/2 digital study packs and to their components (Table 2, 1-to-5 scale).

Table 2. Student rating of digital resources and activities

Study pack resources / activities	*CS1-Python*	*CS2-Java*
Sample programs	4.7	4.1
The entire online study pack	4.5	4.5
Online lab assignments	4.3	4.5
Online quizzes	3.8	3.5
Printed textbooks (if available)	1.8	2.9

In early Summer of 2006, the first full editions of the packs were published at http://studypack.com. In just a few months, several instructors at four institutions have already adopted pack instances for their courses, teaching about 280 students in Fall 2006.

4. REFERENCES

[1] ACM SIGCSE. *SIGCSE-MEMBERS archives*, April 2006, week 2. http://listserv.acm.org/scripts/wa.exe?A0=sigcse-members

[2] Bailey, T., J. Forbes. Just-in-time teaching for CS0. *SIGCSE'05* (St. Louis, Missouri), 366 - 370.

[3] Bell, J. Inductive, abductive, and pragmatic reasoning. *JCAI'97 workshop on abduction and induction in AI* (Nagoya, Japan), 7-12. http://www.cs.bris.ac.uk/~flach/IJCAI97/papers.html

[4] Burch, R. Charles Sanders Peirce. *Stanford Encyclopedia of Philosophy*. http://plato.stanford.edu/entries/peirce/#dia

[5] Gonzalez, G. A systematic approach to active and cooperative learning in CS1 and its effects on CS2. *SIGCSE'06* (Houston, Texas), 133-137.

[6] Josephson, J. Smart inductive generalizations are abductions. In: *Abduction and Induction, Essays on their Relation and Integration*, Edited by Flach P. and Kakas A. Springer (New York), 2000.

[7] Lahtinen E., Ala-Mutka K., Järvinen H-M. A study of the difficulties of novice programmers. *ITiCSE'05* (Caparica, Portugal), 14-18.

[8] Radenski, A. Python First: A lab-based digital introduction to computer science. *ITiCSE'06* (Bologna, Italy), 197-201.

[9] Sun Microsystems, Inc. *The Java Tutorial*. http://java.sun.com/docs/books/tutorial/