AC 2010-1565: MEASURING THE EFFECT OF INTERVENING EARLY FOR ACADEMICALLY AT RISK STUDENTS IN A CS1 COURSE

William Punch, Michigan State University

Bill Punch is an Associate Professor in the Department of Computer Science at Michigan State University as well as the director of Michigan State's High Performance Computing Center. He is co-director of the Genetic Algorithms Research and Applications Group or GARAGe. His main interests are genetic algorithms and genetic programming, including theoretical issues (parallel GA/GP) and application issues (design, layout, scheduling, etc.). He also has conducted active research in data mining, focusing on the use of ontologies such as WordNet and Wikipedia for text search. He has also just published the textbook "The Practice of Computing using Python", a CS1 text using Python as the main language.

Richard Enbody, Michigan State University

Richard is an Associate Professor in the Department of Computer Science and Engineering. He joined the faculty in 1987 after earning his Ph.D. in Computer Science from the University of Minnesota. Richard received his B.A. in Mathematics from Carleton College in Northfield, Minnesota in 1976, and spent six years teaching high school mathematics in Vermont and New Hampshire. Richard's research interests are in computer security, computer architecture, web-based distance education and parallel processing, especially the application of parallel processing to computational science problems. In 1998 Richard pioneered a CS1 course (first course in Computer Science) over the World Wide Web using RealVideo synchronized with PowerPoint. Together with Bill Punch he recently published a textbook using Python in CS1: The Practice of Computing Using Python (Addison-Wesley, 2010).

Colleen McDonough, Michigan State University

Colleen A. McDonough is a graduate assistant at the College of Engineering at Michigan State University. She is the coordinator of two component projects of a National Science Foundation grant focusing on retention issues and engaging early engineering students, and also serves as an academic advisor. Prior to coming to MSU, Colleen spent ten years as a development officer in the non-profit sector. She earned her bachelor's degree in sociology from William Smith College and her master's degree in Public Administration from the University of Southern California. McDonough is currently a doctoral student in the Higher, Adult and Lifelong Education program at Michigan State. Her areas of interest include educational theory, student development and education policy.

Jon Sticklen, Michigan State University

Jon Sticklen is the Director of the Center for Engineering Education Research at Michigan State University. Dr. Sticklen is also Director of Applied Engineering Sciences, an undergraduate bachelor of science degree program in the MSU College of Engineering. He also is an Associate Professor in the Department of Computer Science and Engineering. Dr. Sticklen has lead a laboratory in knowledge-based systems focused on task specific approaches to problem solving. Over the last decade, Dr. Sticklen has pursued engineering education research focused on early engineering; his current research is supported by NSF/DUE and NSF/CISE.

Measuring the Effect of Intervening Early With at Risk Students in a CS1 Course

Abstract
We recently converted a CS1 (Introduction to Computing) class to use the Python language in place of C++. Among other reasons, we hoped that the new language would help students who typically struggled with the course. Our typical drop+fail rate was around 25%-30% for C++, and we hoped the conversion would reduce this number. Though it did reduce slightly, 15%-25%, it was not as significant as we had hoped. We therefore tried an early intervention strategy to help those students whom we could identify as struggling. We provided extra tutoring for only those students. We then calculated statistics on the effects this extra tutoring. The results were not good: we found no significant difference between the group of students who used the tutoring and those that did not. We review some of the potential reasons for this result.

Background, Why Python
A CS1 course is a first course in computer science, and usually emphasizes an introduction to programming. It is also a course on problem solving and applying a programming language to solving a problem. As a result, the choice of programming language can have a significant impact on the implementation of the course (see Pears et al.[8] for an excellent survey). A recent survey of the top thirty Ph.D. CS degree-granting programs showed a distinct preference for Java [Forbes and Garcia[3]]. For fifteen years C++ has been the language for our CS1-CS2 sequence—a long time in the computer science world.

As in some other institutions, non-CS majors have found our CS1 course to be useful. We find that now the majority of students in the course are non-CS majors who are not required to take the course. STEM students (Science, Technology, Engineering, Mathematics) are naturally drawn to the course, but we have found students from all majors in our CS1 course. As the impact of computing has grown across all fields there has been an increasing need for students in all majors to develop some programming skills. In particular, a computing course that, after one semester, develops students into effective programmers is needed. We found that C++ did not adequately satisfy that need within one semester, and we were not convinced that its sibling languages, Java and C#, satisfied that need significantly better.

Languages such as Alice [Powers et. al.[9]] and Scratch [Malan and Leitner[7]] have proven to be attractive introductions to computing, especially for non-majors. Media computation [Guzdial[4]] has also been effective. Non-language approaches such as the Principles of Computation [Cortina[1]] have also proven to be effective. However, many such approaches are for "CS0" courses. Such courses are valuable, but we are working with a course that must prepare students for CS2, and it has not yet been demonstrated that those approaches satisfy that goal.

Python features a mixture of readability and practicality—nice features for an introductory language. It is also an interpreted language that encourages experimentation—a great learning aid. It has a number of immediately available data structures (strings, lists, dictionaries a.k.a. associative arrays, and sets) with associated functions and methods to easily manipulate those structures. It is object-oriented which helps in preparation for both solving complex problems and other languages. It is a free language that runs under most environments including, but not

limited to, Microsoft Windows, Mac OS-X, and Linux. It includes many modern programming language features together with a seemingly limitless set of modules that extend it. Finally, Python interacts well with other languages.

In short, Python can be described as a best-practices language, providing practical tools to do a job with a minimum of effort. Taken together these features allow a novice to focus more on problem solving and less on language issues. In addition, the built-in language features make data manipulation particularly easy allowing students to more easily work on real data. As a result, not only do students solve more challenging problems, but they also have a tool that can be used in subsequent courses, research or even personal use.

Subsequently, a textbook that incorporates our approach is available: "The Practice of Computing Using Python" [Punch and Enbody[10]], a textbook for using Python in a CS1 course.

For these reasons, Python reduced our drop+fail rate, but the question remained: how can we reduce it further? To understand the issues, we need to explain the course.

CS1 course

Our CS1 course is a fairly standard course covering problem solving, data structures, and basic programming concepts. The course has a weekly closed lab where students, with the aid of a teaching assistant, work exercises. Of greater importance is the weekly programming assignment—there are eleven such projects throughout the semester. We have observed that students were struggling more with the problem than with the implementation in Python. That is, once they understood the problem, the solution came relatively easily.

A preliminary indication from three offerings is that Python is an improvement for our CS1 students—we are working on data to better establish those indications. However, the kinds of problems the students can address have changed. Students have been able to download real data from the Web and analyze it (fitting the ideas of [Jukic and Gray[5]]). Examples include building a simple classifier for breast-cancer data, finding cycles in sunspot data, and simulating DNA transcription. The idea is to grab some real data, parse it into useful form, and analyze it—ideal problem solving skills for scientists. Text analysis such as building a concordance or tag-cloud is quite simple in Python, and provides a tool that humanities students can use in their courses.

Overall Goals

Overall, the goals we had for the conversion were:

1. Learning to program for the first time presents two problems a student must address simultaneously: improving student ability to solve problems and translating that ability into a programming language. While the first is typically the most challenging, the second is a significant issue.
2. Our CS1 class is typically 25% CS majors, 25% other Engineering majors, and 50% other. The class should serve all groups: allowing majors to progress in their subsequent studies while also providing a practical foundation for non-majors. Our opinion was that all students should have the following thought subsequent to the course. When presented with a problem, they would think "Hey, I'll just write a program for that".

3. The typical drop+fail rate in the previous version of the class (taught in C++) hovered around 25%. On the one hand this was commensurate with our sister institutions, but on the other that value seemed rather high. We hoped the change would reduce this number

We have run the course in Python for four semesters and can report on our progress. First, Python has proven to be a syntactically simpler but still full-featured language. As a result we are able to cover more real-world problems in the course, giving students good scaffolding for addressing problems they are likely to see again (file processing, statistics, graphing, etc.)

Second, we have published results showing that majors who took CS1 in Python do as well in later courses as those who took CS1 in C++[Enbody and Punch[2]]. Subsequent courses taught in C++ have required few changes to the courses with no change in student performance. Anecdotally, non-majors have reported to us the utility of the course in both their academic work and in their subsequent employment. Many of these reports focus on simple manipulation of data for various projects.

Finally, the drop+fail rate is lower, though only slightly. It now hovers around 20%.

What could still be improved

This is all good news. However, we wondered if there were still some things that could be improved. In particular, the drop+fail rate proved to be relatively unchanged. Though it would fluctuate, the rate remained roughly 20% over the now five offerings of the Python CS1 course and approximately 1000 students.

The students had multiple opportunities for programming and for help in the present course setup. Every week a lab was conducted on the topic of the week. These labs are broken down into groups of approximately 20 students. An exercise is provided that the students collaborate to solve, along with help from a Teaching Assistant. No grading is done in the lab, only attendance is taken. This topic is then used as a homework assignment for the week, forcing the students to work on their own on the weekly topic. Approximately 18 hours of help room hours are also provided during the week where students can stop in and ask for further help with the weekly homework. And still, we have a 20% drop+fail rate.

An Intervention Approach

One idea was to try and identify those students who might be at risk for doing poorly in the class. If we could intervene with those students early, help them catch up on the topics with which they struggled, we could have an effect on the drop+fail rate. In particular, our thinking was as follows. Students who are struggling with the course often cannot master a particular topic (functions, classes, lists, etc.). Once they cannot get past that topic, they seem to remain stuck and never recover. If we can intervene with the "stuck" problem, perhaps we could push them along further, even all the way through the course.

Approach

We looked back through the previous four offerings of the CS1-Python for an indicator that showed up relatively early in the course and indicated potential poor performance. By the time of the first midterm, approximately one third of the way into the course, the students had completed

4 homework assignments, 5 labs and the midterm itself. Though no great indicator was found, the best we could find was student performance on the multiple-choice part of the first midterm (there is also a "write a program" part of the midterm). A grade of 65% or less was selected as the indicator. Previous semesters showed that approximately 80% of the students with 65% or less on the first midterm multiple-choice part scored less than a 2.0 for the course or dropped.

We sent those students who did receive a 65% or less on the first midterm multiple choice part an email. The message indicated to them what they already knew; that they did not do well on the exam and were in danger of not doing well in the course. We tried to sound a bit dire while at the same time not sounding defeatist, hoping to get their attention and spur them to action. We then offered them some help. We would conduct two weekly tutorial sessions on topics that were proving difficult for *only* those students who were struggling. We selected this subgroup for two reasons:

- We had some evidence that; indeed, these students would not do well if they did not do something else.
- We thought that having students who were all in the same boat would feel freer to ask questions and more fully explore what they did not understand.

We hired two undergraduate students to conduct these sessions, staring with an overview of functions. Again, we hired undergraduates to change the approach: we thought they would relate better to other students (as opposed to faculty). We conducted these voluntary sessions through the remainder of the semester.

Results

The offer was made to 50 of the 211 students in the class as described. However, as the sessions were voluntary, the attendance was poor. We never had more than ten students show up at once and often just one or two. At the end of the semester, we ended up with two groups: the emailed students who showed up for at least one sessions and the emailed students who did not attend a session. These two groups constituted our population. We compared the results of those two groups to see if attendance at the sessions had an effect on their final grade. Of the 50, 16 students did not take the final exam and therefore did not complete the course. Those students were therefore not used in the comparison. That left 13 students who completed the course and actually attended at least one session and 21 students who completed the course and never attended a session. The results are shown in Table 1 below.

	Count	Course Grade Average	Standard Deviation
Attended a session	13	51.3%	15.4%
Did not attend a session	21	54.8%	13.2%

Table 1. Students who completed the course and did or did not attend a help session.

Discussion

The results were not encouraging. The average final grade for those that attended the sessions was actually *lower* than those that did not, though the results did not prove to be significant

(p=0.50). A linear regression model of the relation between the attendance and the final grade also showed no relation (p=0.84). The low attendance numbers and lack of significance make it difficult to provide reasons for the results (or lack thereof). However, we can speculate.

1. The students who attended more often were in worse straits than those who did not. The "did not attend" students had more confidence in being able to dig themselves out of their hole. We could confirm this with access to more data (GPA, ACT results). We plan on doing this in the near future.
2. The "direness" of the email and the offer of help was not enough to encourage students to come for more help. It is a tricky thing to motivate dedicating even more time to a course that has such high workload.
3. The kind of help provided, going over existing class material, was not seen as valuable. In fact, our experience is that struggling students have little patience for reviewing material. They want direct help on solving the problems they are encountering in the present homework problem. If the students do not receive help that they perceive to be directly affecting the current problem, they stop using that help. This is a level of maturity issue that, despite our best efforts, will continue to occur in an early undergraduate course.
4. Intervening at the end of the first third of the class was too late. Students who were struggling were already defeated and therefore unmotivated to try and solve their predicament.
5. For whatever reason, usually reasons of time, the students we are trying to help will continue to struggle given their present circumstances. This could indicate a lack of maturity; lack of time (many students must work to go to school in these economic times), or more controversially, a lack of ability [Kramer[6]].

Conclusions

We identified students in a CS1 course at the one-third point of the semester who were struggling and in danger of doing poorly in the course. We informed them of the problem and offered them extra help only for students in the same situation. The motivation of helping with their problems motivated less than half the students to attend these sessions, and the overall results of attendance did not seem to affect the final outcome. We are working on tuning the approach. In particular, we are looking for a better motivator to attract students who are struggling and help them in a way that they feel more directly addresses their problems.

Acknowledgements

This material is based upon work supported by the National Science Foundation under award 0757020 (DUE). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Bibliography

[1] T. J. Cortina. An introduction to computer science for non-majors using principles of computation. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 218–222, New York, NY, USA, 2007. ACM.

[2] R. Enbody and W. Punch. Performance of Python CS1 Students in Mid-level non-Python CS Courses, SIGCSE '09, *Proceedings of the 40th SIGCSE technical symposium on Computer science education,* pages 116-120, New York, NY, USA, 2007. ACM.

[3] J. Forbes and D. D. Garcia. "...but what do the top-rated schools do?": a survey of introductory computer science curricula. *SIGCSE Bul* l., 39(1):245–246, 2007.

[4] M. Guzdial. *Introduction to Computing and Programming in Python: A Multimedia Approach*. Pearson Prentice Hall, New Jersey, 2005.

[5] N. Jukic and P. Gray. Using real data to invigorate student learning. *SIGCSE Bul* l., 40(2):6–10, 2008.

[6] J. Kramer. Is Abstraction the Key to Computing, *CACM*, 50(4):36-42, 2007.

[7] D. J. Malan and H. H. Leitner. Scratch for budding computer scientists. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 223–227, New York, NY, USA, 2007. ACM.

[8] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. *In ITiCSE-WGR '07: Working group reports on ITiCSE on innovation and technology in computer science education,* pages 204–223, New York, NY, USA, 2007. ACM.

[9] K. Powers, S. Ecott, and L. M. Hirshfield. Through the looking glass: teaching cs0 with alice. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 213–217, New York, NY, USA, 2007. ACM.

[10] W. Punch and R. Enbody. *The Practice of Computing Using Python*, Addison Wesley, 2010.