GRADING QUALITATIVELY WITH TABLET PCs IN CS 50

David J. Malan
School of Engineering and Applied Sciences
Harvard University
malan@post.harvard.edu

1. ABSTRACT

CS 50 is Harvard College's introduction to Computer Science for majors and non-majors alike. Each week, our 330 students submit programming assignments comprising hundreds of lines of code that must then be graded. Although we can assess the correctness of some code automatically, some measures of quality require human attention. In Fall 2008, we equipped the course's 27 teaching fellows (TFs) with Tablet PCs in order to grade more efficiently but no less qualitatively. By blurring the lines between files and paper, we hoped to facilitate typed and handwritten feedback alike so that grading itself would be not only evaluative but instructive as well. At term's end, most TFs (63%) reported that grading took less time with a Tablet PC, and nearly half (48%) also reported that they provided students with more feedback because of the same. We present in this paper CS 50's experience with Tablet PCs along with pedagogical benefits thereof.

2. PROBLEM STATEMENT AND CONTEXT

CS 50 is Harvard College's introduction to Computer Science for majors and non-majors alike, a one-semester amalgam of courses generally known as CS 1 and CS 2. By way of lessons in programming, the course vows to teach students how to think algorithmically and solve problems efficiently. Toward that end, the course assigns weekly problem sets (*i.e.*, programming assignments) that challenge students to write programs that solve problems inspired by real-world domains. Students write most of these programs in C, a programming language that resembles terse English, per Figure 1. Even though most students (94%) enter this course with little or no prior programming experience, they all exit with hundreds of lines of code under their belt.

This course, though, has hundreds of students, which means tens of thousands of lines must be graded. Fortunately, we can (and do) assess, to some extent, the correctness of students' code automatically, as by comparing their programs' output against our own solutions [4,5,7]. But other measures of quality require human attention. Much like two essays with the same thesis can differ in quality (even though tools like Microsoft Word might judge both essays' grammar correct), so can two programs with the same output differ in quality as well. Rather than rely on measures of correctness alone, then, we evaluate students' programs along three axes:

- *Correctness.* To what extent is the code consistent with our specifications and free of bugs?
- *Design.* To what extent is the code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
- *Style.* To what extent is the code readable (*i.e.*, commented and indented with variables aptly named)?

Although students gather twice weekly for large lectures, CS 50 implements apprenticeship learning [2, 3], whereby each student is apprenticed at term's start to a teaching fellow (TF) who

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("hello, world!");
}
```

Figure 1: A canonical program written in C, a programming language that resembles terse English. By term's end, students write programs like this one but with hundreds of lines.

is responsible for grading 12 students' work. The axes along which we grade students' work do, therefore, invite subjectivity, as reasonable minds can differ when it comes to the quality of a program's design. But we are far more concerned with the process of learning than we are with distinguishing our students numerically. We emphasize to the TFs that grading is but another opportunity to teach. Provided they supplement their quantitative feedback (*i.e.*, scores) with qualitative feedback (*i.e.*, comments), grading can be as instructive as it is evaluative. Even so, we do try to standardize grading in the interests of fairness (as by distributing rubrics to the TFs), and we also try to normalize grades across TFs at term's end by taking into account which TFs appear to have been "tougher" graders than others.

To facilitate feedback both quantitative and qualitative in years past, students not only submitted their code online, they also handed in printouts. Online submissions enabled automated scoring, whereas paper enabled handwritten comments. Much like an essay, it's quite easy to annotate code with a pen, even drawing pictures as needed. But printouts are not very green, and most TFs can type faster than they can write. And so we eventually replaced printouts with PDFs. But, in TFs' opinions, keyboards and mice have proved to be suboptimal replacements for the ease of a pen. The transition from paper to PDF was simply a tradeoff.

Moreover, even with an army of TFs, grading per our three axes still takes time. Although we urge the TFs to spend no more than 4 hours grading per week (*i.e.*, 20 minutes per student), most, to their credit, spend more.* On average, the TFs report spending no fewer than 7 hours per week grading, and those hours are on top of an additional 8 hours: these same TFs also teach sections (*i.e.*, recitations), hold office hours, and field questions by email. And yet CS 50's TFs are traditionally undergraduates themselves, each taking 4 or more courses. The more time they spend on CS 50, the less time they have for their own studies.

With so many hours each week spent on grading alone, optimization thereof is of perennial interest. The challenge, though, is to streamline without sacrificing axes. We set out in Fall 2008 to grade more efficiently but no less qualitatively.

3. SOLUTION EMPLOYED

We equipped each of Fall 2008's 27 TFs with a Tablet PC.† We hypothesized that Tablet PCs offer the best of both worlds: the ability to mark up students' code qualitatively with ease without resorting to paper. Moreover, we hoped that Tablet PCs would enable TFs to flip rapidly (as via Alt-Tab) between students' actual code and PDFs thereof, thereby expediting their workflow.

*We daresay any fewer than 20 minutes per student would be a disservice, inasmuch as most students invest more than 10 hours in each problem set.

†Each TF received a Dell Latitude XT.

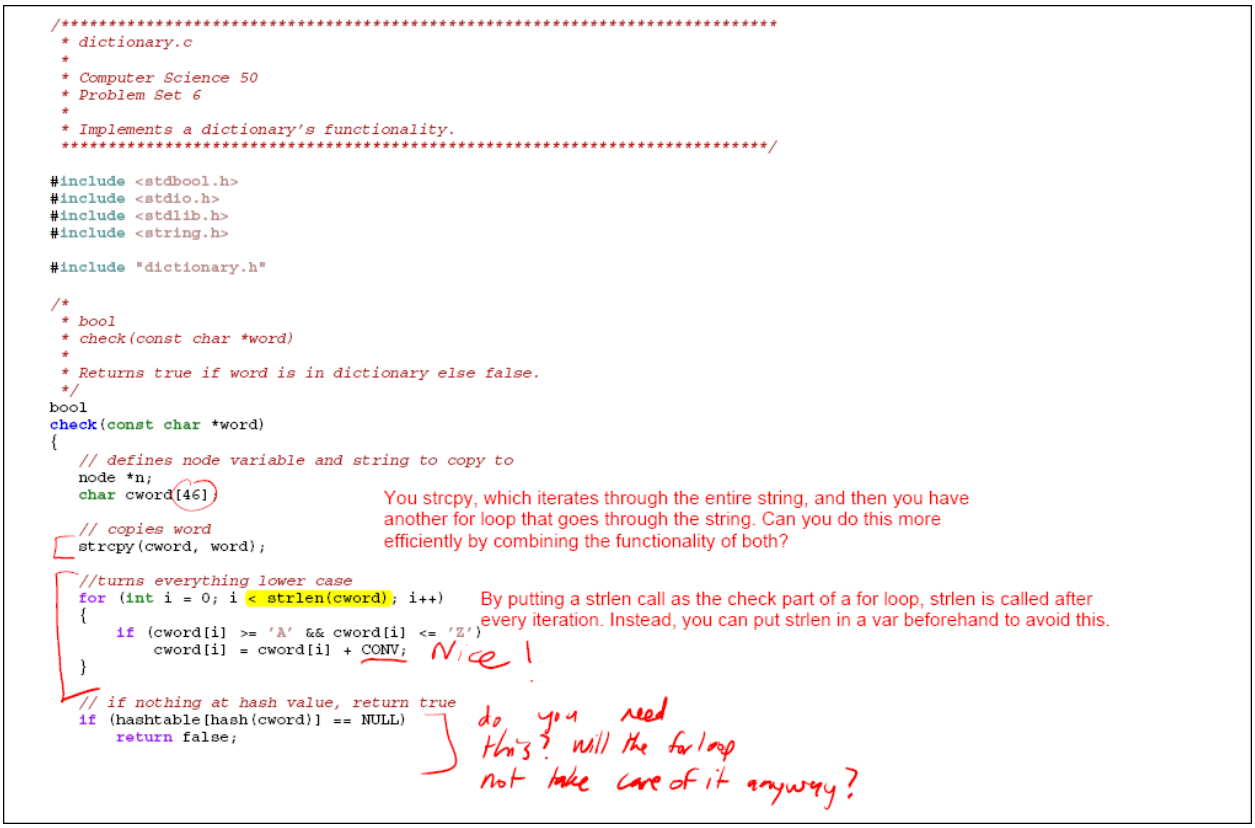

Figure 2: Using freely available software, we generated PDFs of students' code for TFs to annotate. Per this excerpt from an actual student's PDF, most TFs annotated PDFs with some combination of handwritten and typed text, circles and lines, and (yellow) highlights.

Although we experimented before term's start with alternative file formats, we found that PDFs offer the most flexibility. TFs can use any number of PDF editors, and students can use any number of PDF readers. Moreover, we can generate PDFs of students' code automatically using freely available software [1, 6, 8], complete with syntax highlighting (*i.e.*, colors that help TFs identify code rapidly) and bookmarks (*i.e.*, hyperlinks that enable TFs to jump from one program's code to another's within a PDF).

Each week, upon some assignment's submission, we automatically generated 330 PDFs, one for each of our students. Those PDFs, along with students' actual code, were then distributed among the TFs for grading. The TFs were asked to assign each submission three scores, one for each of our axes. To maintain our focus on qualitative feedback, though, each axis was scored using only a 5-point scale, each of whose values connoted some level of quality: 1 implied poor, 2 implied fair, 3 implied good, 4 implied better, and 5 implied best. So that correct code would not be unduly punished for lacking only in, say, style, we also assigned different weights to the axes. By asking the TFs to quantify submissions' quality using far fewer buckets (three 5-point scales as opposed to 100-point scales in years past), we hoped to free up more time for comments. We encouraged TFs to spend most of their time actually annotating PDFs. Once done grading, the TFs then uploaded their students' scores and annotated PDFs to the course's website, where their students could view them. Per Figure 2, most TFs annotated PDFs with some combination of handwritten and typed text, circles and lines, and (yellow) highlights.

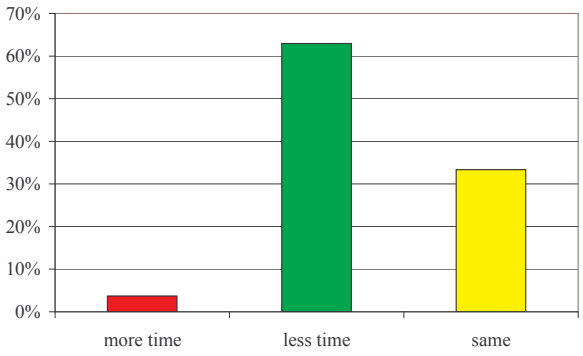

Figure 3: Most TFs (63%) reported that grading took less time with a Tablet PC.

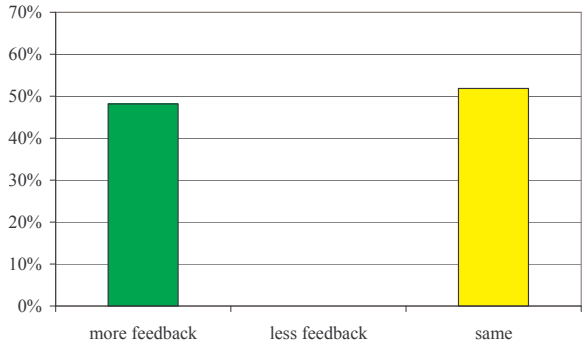

Figure 4: Nearly half of the TFs (48%) also reported that they provided their students with more feedback because of the Tablet PC.

4. EVALUATION

In order to assess the impact of Tablet PCs on the TFs' workflow, we surveyed the team at term's end. Because we distributed the hardware only after the TFs had already graded several assignments, we asked them to compare their experiences with and without Tablet PCs. Per Figure 3, most (63%) of the TFs reported that grading took less time with a Tablet PC, while some (33%) reported that grading took the same amount of time; one TF reported that grading took more time. Among those who spent the same amount of time grading, though, many explained that they ended up providing their students with more feedback. In fact, per Figure 4, nearly half (48%) of the TFs reported that they provided their students with more feedback because of the Tablet PC. One TF explained the effect: "In general, the tactile interface of the tablet makes the translation of thought to computer natural and easy. Grading-wise, this meant that writing comments while reading code tended not to interrupt my stream of thought. This was really helpful. Without the tablet, the management of comment placement on PDFs would require additional thought; the ease of using the pen made the experience far more natural and even enjoyable."

Our own students, then, appear to have benefited from the Tablet PCs, at least with regard to the amount of feedback received. We surveyed our students as well: 74% reported that they had received "enough feedback" from their TFs, and 16% reported that they had received "more than enough feedback." Only 10% reported that they had received "too little feedback." Of course, receiving more feedback does not necessarily mean absorbing more feedback. As one TF lamented, "I don't think most students read comments, which is what tablets helped the most with."

And so we asked the TFs themselves whether the Tablet PCs helped improve learning outcomes for their apprentices: 57% of the TFs answered yes, 10% answered no, and 33% were unsure. Explained one TF: "I think they did, because I spent more of my grading time really reading their code and commenting on it. Some of them did not react to my comments, because I expect they weren't reading or paying attention to them. Some definitely responded and improved their output." Elaborated another: "It was much easier to mark up problem sets—I could draw arrows, diagrams, *etc.* and just write much more easily. One could do this with just typing, but it would have taken much longer, and I might not have had the time to give as much feedback. The tablet made things faster for me, which meant that my students could get more thorough feedback, also being able to visually illustrate things via drawings made explanations easier—students probably got more from that than they would have gotten through just plaintext comments."

Although we worried at term's start that the TFs might underutilize their Tablet PCs, insofar as each owned a laptop already, it turns out that having two laptops was itself advantageous for grading: "The greatest benefit of having a tablet was the ability to do work on a small, lightweight computer on the go and use two separate screens for grading. Accordingly, I was able to give more feedback to each student because grading became more efficient and mobile." That the Tablet PCs were so much lighter (4.1 lbs) than so many TFs' own laptops may have had unintended effects on their attentiveness to students' emails as well: "The tablet is so light that I wound up carrying it around with me most places, letting me respond to students with faster turnaround time."

More compelling, perhaps, was the Tablet PCs' impact on some TFs' sections. Noted one TF: "Teaching section with the tablet made it much easier to do examples in class." Another elaborated: "I think the tablet helped me convey my ideas better to my students, both in section and on homework. I noticed that sections where I used the tablet to highlight code and draw diagrams went a lot smoother in general; it was easier for me to teach, and I think students were more easily able to grasp the material. The tablet allowed me to more easily demo code and draw on the code at the same time." To be sure, not all TFs reported such benefits: "It made everything more convenient for me, but I don't think I would have taught/graded differently without it." But even I have begun using a Tablet PC in lectures so as to draw (literally) attention to lines of code on my screen. In this author's opinion, that feature alone is itself pedagogically compelling.

These benefits certainly come at a cost. Tablet PCs are not inexpensive, at least at the moment. But there is undoubtedly a price point at which this technology's adoption should be taken for granted. It's too useful not to expect teachers to have it.

5. ADDITIONAL RESOURCES

The address of CS 50's website is `http://www.cs50.net/`.

6. ACKNOWLEDGEMENTS

Many thanks to Shafeen Charania and Chris Johnson of Microsoft, Jill Dixon of Dell, and Beth Leitner of Bluebeam, without whose support this experiment would not have been possible. And many thanks to Fall 2008's students and teaching fellows, without whom this experiment would not have been possible.

REFERENCES

[1] Artifex Software Inc. Ghostscript. `http://www.ghostscript.com/`.

[2] O. Astrachan and D. Reed. AAA and CS 1: The Applied Apprenticeship Approach to CS 1. In *SIGCSE '95: Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, pages 1–5, New York, NY, USA, 1995. ACM.

[3] O. Astrachan, R. Smith, and J. Wilkes. Application-Based Modules Using Apprentice Learning for CS 2. *SIGCSE Bull.*, 29(1):233–237, 1997.

[4] S. H. Edwards. Teaching Software Testing: Automatic Grading Meets Test-first Coding. In *OOPSLA*, pages 26–30. ACM Press, 2003.

[5] D. Jackson and M. Usher. Grading Student Programs Using ASSYST. *SIGCSE Bull.*, 29(1):335–339, 1997.

[6] B. Lowagie and P. Soares. iText. `http://www.lowagie.com/iText/`.

[7] K. A. Reek. A Software Infrastructure to Support Introductory Computer Science Courses. *SIGCSE Bull.*, 28(1):125–129, 1996.

[8] M. Rossi. GNU Enscript. `http://www.codento.com/people/mtr/genscript/`.