

An Unusual CS 1 With High Standards and Confirming Results

Guido Rößling
roessling@acm.org

Max Mühlhäuser
Max.Muehlhaeuser@acm.org

Technische Universität Darmstadt
Dept. of Computer Science
64289 Darmstadt, Germany

ABSTRACT

Our department has opted not to do the “usual” CS 1 course. We discuss the features of the resulting course, and how they are evaluated by our students. While several elements are unusual, evaluations have shown them to be very successful and popular with our students. We hope this report will inspire other educators to “think outside the box in CS 1”.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer & Information Science Education - *Computer Science Education*

General Terms

Management

Keywords

CS1, game programming, Java, Moodle, Scheme

1. INTRODUCTION

One of the first things that many universities teach their CS students is how to program. This is usually done in the CS 1 course, which often teaches the basics of Java.

In discussions with several colleagues teaching “similar” introductory programming courses at their home universities, it has become clear that the approach adopted by our CS department is unusual in several regards. We therefore want to present our approach for wider discussion, for a dual purpose. On the one hand, we are always interested in feedback and supporting or opposing arguments regarding our approach. On the other hand, we also hope that aspects of our approach will meet the interest of other educators.

In this paper, we present the contents and structure of our course and its accompanying teaching and learning materials. We also present statistics and evaluations of learning outcomes (as measured by assessments, exams, or student feedback), and student attitudes towards our approach.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ITICSE 2010, June 28-30, 2010, Bilkent, Turkey.

Copyright 2010 ACM 978-1-60558-381-5/09/07 ...\$5.00.

2. STRUCTURE OF OUR CS1 COURSE

Our CS 1 course is the first course in Computer Science attended by students from several degree programs. It is mandatory for students of Computer Science, Information Systems, Computational Engineering, and Information System Technology. Additionally, the lecture is attended (but not necessarily in the first term) by students of Mathematics and other degree programs with a minor in CS, e.g., students of physics, as well as by joint bachelors of CS and a secondary degree program from the humanities.

The CS 1 course consists of two 90 minute lectures per week for about 15 weeks, and a weekly 90 minute exercise. The topics covered in the lecture are rehearsed during the exercise sessions and then further trained by graded homework assignments. CS 1 exercise sheets are available on Monday at 8 AM, will be covered by the exercise sessions spread over the full week (except for Mondays), and have a homework due on Friday of the following week. All students thus have at least one week between their exercise session and the deadline for handing in the homework, and can prepare the homework for almost two weeks.

The regulations for passing the CS 1 course are largely determined by our CS department. As shown in Figure 2, students need to gather points from homework assignments, a mid-term exam and a final lab. If they have passed all individual elements and also reached the required sum of points, they are allowed to register for the final (written) exam.

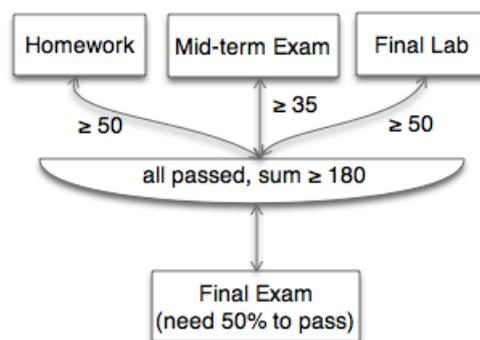


Figure 1: Organization of the CS 1 course

The CS 1 course ends with a lab (described in more detail in Section 4.3) and a written exam. The written exam tests

most of the course materials by providing tasks that cover code reading and understanding, programming, and complexity. Both mid-term exam and final exam take 2 hours.

So far, the CS 1 course may seem similar to many others worldwide—expect perhaps for the “exam qualification” required for the exam. However, since this is a regulation of our university, we could not change this requirement, even if we wanted to.

In the next sections, we will address step by step what makes “our” CS 1 unusual, why we have chosen to do it this way, and benefits or other effects this has caused.

3. UNUSUAL ORGANIZATIONAL ASPECTS

3.1 Collaborating Educators

In the winter term, the lecture is held by two educators. Each educator presents the one part of the course, with a switch between educators at roughly the half of the course. As we have to offer the course in both German and English, the workload for each educator is four times 90 minutes per week, for about 6-8 weeks in sequence. The second educator also helps out in the first phase by addressing audience questions raised in the learning platform.

While the workload during the “active” phase—90 minutes of lecture on each day from Monday to Thursday—is rather high, the collaboration means that it is limited in time. Additionally, both educators can focus on “their” topic, and do not have to be extensively familiar with both topics.

3.2 CS1 is Offered in Each Term

In summer 2008, our university decided to allow students to start studying CS in both the summer and winter term. Before, students could only start studying CS in the winter term. Some prospective students thus either enrolled in a different university offering CS in the summer term, or spent one term on another degree program before they could change to CS. In many cases, the latter approach essentially meant a wasted term for the student, often accompanied by reduced motivation.

In summer 2008, about 30 freshmen started studying CS with the CS 2 course (algorithms and data structures based on Java). Since many of these freshmen did not know Java, their success in CS 2 was severely limited. The CS department therefore decided to offer CS 1 also in the summer term for CS freshmen.

To allow repeat students to continue studying successfully, we opened this course to those who had failed to qualify for the final exam (see Section 2) or had failed in the final exam. Instead of the initially anticipated 60-80 freshmen attendees, we ended up with 80 freshmen and about 125 other students, including those with a different major. By now, CS 1 is the only larger mandatory course that is offered each term by the CS department.

3.3 Extensive Use of a LMS

The lecture and exercises are held in appropriate rooms—the auditorium maximum for the lecture attended by 600+ students, and a set of smaller seminar rooms for the exercises. Both educators and the teaching assistants for the exercise groups also offer consultation hours. All else is done electronically inside our Moodle learning platform, see [1] and <http://www.moodle.org>.

The learning platform contains the lecture materials, exercise sheets and—after the deadline for a given homework has passed—their solution; sample exams published before the mid-term exam to help students prepare for the exam, with a solution published about one week before the exam; opinion pools, quizzes, a number of forums for targeted discussion, and a set of other features.

At (almost) any given point in time, at least some students are likely to be online in the learning platform, including for example Christmas and New Year’s Eve. One of the educators is highly present and also hosts and adminstrates the platform. Our students quickly get the impression that their answers are addressed “24/7”. While this is far from correct, we have noticed that a fair number of questions were indeed answered at night or in the early morning hours, for example between 2:00 AM and 4:00 AM. While these postings were not written by the educators, they still affected the perceived usefulness of the learning platform.

Students appreciated that all information regarding the course is held in the central LMS, together with search facilities and RSS feeds. From the educator’s point of view, managing everything inside the portal has made organizing and administrating the lecture far easier. Before, we had to maintain a web page (regularly updated with new exercises etc.), track a forum on a different server, coordinate with tutors over a mailing list, and reply to email inquiries. All this now essentially happens inside Moodle. For example, at the time of writing, most of the two lectures—attended by a total of about 680 active students, with almost 1000 students enrolled in the two courses—is over, but there have been less than 10 direct email inquiries outside Moodle.

3.4 Modularized Lecture Recordings

To make learning our materials easier for our students, we record each lecture and provide a link inside the learning platform. Since our change to Camtasia—see, e.g., [6]—, we can usually provide the lecture recordings on the same day as the lecture.

Instead of “regular” 90 minute lecture recordings, we have split each lecture into different subtopics. A typical lecture will cover about four to six such topics. For example, the slide set about *inheritance, abstract classes and late binding* contains 89 slides (including the title page and several “table of contents” subpages). This slide set covers nine different subtopics, starting with a motivation for inheritance, incremental programming as an analogy, and how to define new object types in Java. The contents are presented in two 90 minute lectures. Each subtopic of the slides is available as a separate recording, leading to a total of 9 recordings with a duration between 9 minutes (“overriding existing definitions”, 4 slides) and 28 minutes (“abstract classes and interfaces”, 12 slides).

Each recording is rendered into two different formats: as a Flash video (including the slides with all live annotations, the audio track, and a smaller video of the lecturer) and as a podcast (size 640x480, slides with annotations and audio track only). Thus, the above slide set ends up as 18 learning resources for the students.

The 22 slide sets that make up our lecture are included as one separate resource each in the LMS. Each one contains a link to the PDF file, a table of all recordings with title, slide range (such as “T13.15-T13.24”), duration of the recording, and a link to the Flash and Podcast recordings.

As mentioned above, the recordings are rendered directly after the lecture and are usually available in the platform on the same day or the next day. We also offer separate RSS feeds for the Flash videos and podcasts.

Additionally, as the same course was held and recorded in the preceding term, we provide the recordings of all lectures from the previous term ready for use before the first lecture has even started. This helps students to prepare before the lecture and to rehearse the materials after the lecture. This especially concerns those with little previous exposure to programming or foreign students who may not be able to directly understand the educator. Three especially talented seniors from a senior high school also attend CS 1. As one of the two weekly lectures starts at 8 AM, and thus at a time when they have to attend high school classes, the recordings and the LMS allow them to be fully integrated in the course, although they cannot attend all lectures.

Typically and unsurprisingly, the peak access for lecture recordings is shortly before the mid-term and the final exam.

3.5 Bonus Points for the Final Exam

To improve the final grade, bonus points are awarded if a certain minimum number of points was reached in the mid-term exam, the final lab and the homework assignments. These are factored into the exam points if a certain minimum number of exam points was reached—typically, students require 30 out of 100 exam points to allow us to include the bonus, and 50 points (including the bonus points) to pass the exam. In this way, students can improve their final grade by up to one full grade, e.g., from D to C. It is also possible to improve from a “fail” grade to the (worst) “pass” grade if sufficient exam and bonus points have been achieved.

Figure 2 illustrates the requirements that have to be met to be allowed to register for the exam:

- at least 50% of the homework points,
- 50 points (out of 100 base points) in the final lab,
- 35 points in the mid-term exam,
- and a total of at least 180 points out of the 100 points possible each in homework, final lab and mid-term exam—and thus, 60% of the points possible.

Note that all conditions are combined with an “and”: students who failed the exam could not simply “do better” in the homework, which they were encouraged to submit as a small learning group. Requiring a certain number of points in the mid-term exam is done to allow students to become familiar with the stress of a written exam. Also, we need to be reasonably sure that the points achieved by a student were actually achieved by the student, not by copying from others. The controlled set-up of an exam makes this easier than for homeworks and other assignments, many of which we actively encouraged to be discussed in a group.

The first bonus points, equivalent to getting one sub-degree (e.g., C- to C) improvement, were granted if 210 (=70%) points were reached. The next improvement steps were at 240 (80%) and 270 (90%) points. *Note for reviewers: At the time of writing, the course still runs. We will add information on how many students reached the 60%/70%/80%/90% mark in the camera-ready version if the paper is accepted.*

4. UNUSUAL TOPIC AREAS

4.1 Teaching Two Programming Paradigms

Our community of lecturers jointly responsible for CS 1, consisting of three full and usually three (currently only one) associate professors, decided in winter 2004 to split the CS 1 course in two parts. The first part covers the fundamental elements of programming, such as methods, recursion, abstraction, complexity and functional abstraction. This part of the course is taught using *DrScheme* and the *How to Design Programs* (HtDP) teaching languages [3]. While these languages are not the same as “real” Scheme, we will refer to them as “Scheme” in the following for the sake of easy reading. This part of the course follows the excellent book by Felleisen et al. [3].

The second part of the course—about the last 8 weeks—deals with Java. After addressing object-oriented programming, inheritance and stepwise refinement, we teach our students what they need to know about Java, including topics such as polymorphism and generics, I/O streams and GUIs.

As far as we know, this combination of “half a term each for functional and OOP programming” is unique. We have decided to follow this approach for several reasons:

- We firmly believe that all CS students should have a good understanding of the basics of programming.
 - Programming languages like Java may be “too much” for beginners. `public static void main(String[] args)` alone can confuse many programming novices. Explaining this carefully at the start is difficult without referring to things that will be learned later; not explaining it is also not a didactically good choice.
 - The HtDP teaching languages are very easy and small, and—perhaps most importantly—have almost no syntactical elements. Compared to a rather “talkative” programming language like Java, this means that novices have to read and write far less lines of code to reach a result.
 - Several powerful aspects of functional programming and the HtDP teaching languages allow us to introduce several concepts comparatively easily, although they are very expressive. One such element is *recursion*. Several papers have discussed the importance—and often, the difficulty—of teaching this “effectively”, e.g., [4, 2, 7]. Since we do not introduce loops in the first part of the course, recursively stepping through a list quickly becomes natural to our students. Additionally, there is no effective way to evade this, in contrast to Java, where each recursive task could also be solved using loops.
- Other expressive features that students quickly grasp are anonymous non-recursive functions (lambda expressions) and higher-order procedures—which they will occasionally miss when we switch to Java.
- We emphasize concepts for developing good software. For example, abstraction and testing play an important role in both parts of the course. These concepts are essentially independent of a given programming language, and can thus be illustrated by contrasting the differences and similarities between “Scheme” and Java.
 - Finally, most other courses at our university are based on Java, and their educators expect the students to be

familiar with Java by the end of CS 1. Thus, we cannot simply spend the complete first term on functional programming, as some other universities do.

4.2 Java GUI Programming Inside CS 1

GUI programming also plays a role in our CS 1 course. GUI programming is important for whatever students plan to do later in their studies. For example, many of the Bachelor Theses written by our students will also require them to implement a GUI front-end for their project. However, there is no other course in our undergraduate study program that actually teaches GUI programming.

To introduce GUI programming, we use the library of the ACM Java Task Force [5] for several homework assignments. This library is used for several smaller assignments, such as drawing a stock exchange diagram, and to enable students to use interaction with the user from the first week of the lecture. We also use it as a motivation for how to model inheritance, although our model differs from the one in [5].

GUI programming using Java Swing also plays a prominent role in our key “unusual feature”, the lab assignment described in Section 4.3.

4.3 Final Lab: Game Programming

The last unusual element of our CS 1 course is the final lab. This lab was first introduced in winter 1995. We have coordinate the lab since winter 2008. The lab always starts some time after the last lecture and lasts for two weeks, and is considered as a full-time job. The lab consists of a common task for all participants that has to be implemented by teams of four students within the two weeks.

Our lab has changed both the organizational approach and the contents of the lab. We provide the task description—about 20 pages—four weeks before the lab officially starts, to give students a chance to become familiar with the task and get all questions answered. Of course, many students also use this time phase to start working on the lab, which we encourage.

The goal of our lab is to implement a fully working computer game, based on a small framework we provide. This framework can essentially place images in a grid and notify other components about keyboard or mouse events, in the latter case also indicating which element was clicked on. In winter 2008, the task was to implement a running version of the classic game *Sokoban*. In summer 2009, the chosen game was *Shisen*, a 2D variation of the classic board game *MahJong*, perhaps better known to computer game players as *Shanghai* (see Figure 2). In the winter 2009 term, our students will implement *Plumber*¹.

The task is split into several subtasks grouped in one base and three extension “levels”. Each subtask will give a certain number of points if it was successfully solved. The “levels” only serve as a general guidance, to prevent students from getting lost in the details.

Towards the end of the two weeks, each lab group can submit the solution and have the project graded. Grading consists of a set of JUnit-based tests run by the group’s tutor, a “live” test where the tutor plays the submitted game, a (quick) inspection of the code, and questions about the code and its structure that the tutor can direct at specific students. We instruct our students that they work as a group and have to pass this test as a group. Therefore, they

¹e.g., <http://www.funny-games.biz/the-plumber.html>

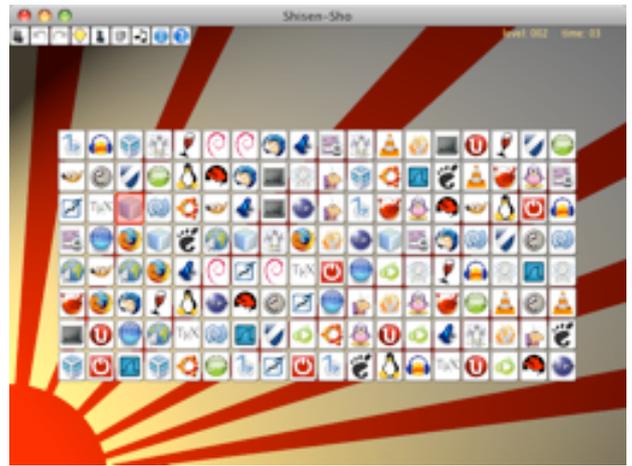


Figure 2: Example Student Lab Result, Summer 2009

should make sure that all participants understand all parts of the code (nearly) equally well, as it is solely the tutor’s choice who will answer a given question.

The final number of points of the lab are determined based on the subtasks the group has solved, adjusted by the degree of correctness of their implementation, as well as by the question and answer session. Groups can also do more than we have demanded by providing bonus features or other “goodies”. For example, the GUI by the group shown in Figure 2 was not based on our framework, but completely rewritten. Up to 45 points are possible to (outrageous) extra materials. For example, two groups in winter 2008 implemented an OpenGL-based 3D version of *Sokoban*, and were accordingly rewarded with extra points. (Note that we do not even mention OpenGL in our course!)

In the past two terms, the lab was a part of the final exam and thus directly counted for the final grade. Starting this term, it will only be part of the exam qualification. Over the last two terms, the drop-out rate for both terms was far less than 10%, and all students who showed up for the final test of the lab passed this with at least the required 50 points.

As indicated above, the switch from other lab topics to computer games has resulted in high motivation and enormous efforts by our students, leading to a large selection of very good solutions. We have also created a *Hall of Fame* for both terms, presenting the best implementations of both terms.

In winter 2008, the average number of points reached in the lab (out of 100, and up to 145 including the bonus points for extras) was 97.15, in summer 2009, it was 92.3.

5. EVALUATION RESULTS

What do students think about our lecture? The first and shortest answer is that they obviously like it, as the summer 2009 lecture was awarded the CS department students’ prize of “best lecture”, and also received the university-wide “Best E-Teaching Award”, as determined by a student jury based on nominations by our students. Additionally, the educator was nominated for the state-wide award for excellence in teaching.

The lecture was also formally evaluated by the CS depart-

Term	Points required	Failed students
Winter 2005	40	148 (50.9%)
Winter 2006	45	29 (36.7%)
Summer 2007	45	12 (42.9%)
Winter 2007	45	99 (26.3%)
Summer 2008	42	25 (54.3%)
<i>Winter 2008</i>	50	269 (38.8% - see below)
<i>Summer 2009</i>	50	30 (26.1%)

Table 1: Exam results in CS 1. Our CS1 lectures as described in this paper are put in italics.

ment and our university’s e-learning center. In the following, a Likert-scale from 1 (very good) to 5 (very bad) was used. In the evaluation by the CS department in winter 2008 (“W”), 125 students submitted a (partially incomplete) survey, while 37 students submitted a survey in summer 2009 (“S”). The total grade for the lecture was 1.79 (W) and 1.28 (S). 92% (W) / 89% (S) would recommend the lecture to a friend. The most common statements in the handwritten notes was praise for the online platform, the speed of replies, and the time and effort invested by the educator.

Our exam results cannot be directly compared to those of previous iterations. We have slightly reduced the duration of the “Scheme” part. We have also introduced Moodle and changed the focus of the final lab to game programming. These changes may also impact the final result of the exams, as shown in Table 5. The difficulty level of all exams was supposed to be similar, to the extent that this is possible. Note that the large number of failed students in winter 2008 was partially due the fact that those students could equalize a bad exam with their final lab points—where, as mentioned in Section 4.3, the average number of points reached was 97.15. Only 151 (26.9%) students actually failed the exam.

The evaluation of the e-learning center focused on the use of e-learning, and thus especially on the use of the LMS and the lecture recordings. The (lengthy) survey was filled out by 82 students. 77% stated that their learning was more successful due to using the e-learning offers, and 74% agreed that the amount of time invested in the e-learning tool paid off well in learning outcomes. The most commonly lauded elements were the forum (graded with 1 or 2 out of 5 by 68% and 21%, respectively), lecture recordings (51%/27%), and lecture materials (62%/26%).

More than 75% of all students felt that they could always access the learning materials, and were able to learn wherever they wanted to. 65%/20% also (strongly) agreed that the portal supported the communication with the educators. Finally, 74% strongly agreed that they were satisfied with the support by the educators, and a further 14% agreed. 9% were undecided, and only 4% were somewhat dissatisfied.

The following data comes from a feedback inside the learning platform. We focus only on the percentage of the 158 participating students who ranked a given features as “good” or “very good”. The didactical concept of the “Scheme” part (Section 4.1) was appreciated by 72.79%, and its concrete realization by 71.52%. The exercises were praised by 78.48%, and the tool support (DrScheme) by 67.08%.

The learning portal was accessed at least three times a week up to every working day by 48.83%, and at least daily

by another 36.07% of our students. The availability of learning materials including the modularized lecture recordings (Section 3.4) (91.1%), forums (86.71%) and polls (74.05%) also received very good grades. We also asked students which features we should make sure to keep if we could support less features (multiple choices were possible). The most relevant elements were learning materials (89.24%), assignments (86.71%), forums (86.71%), group selection (53.80%), quizzes (51.90%) and the database of example code from the slides (45.57%). In fact, the main criticism—voiced by only 5 students—was that the portal “offered too much”, although elements can also be hidden.

We cannot provide a formal evaluation for the final lab due to its placement after the regular evaluation phase. However, the student feedback over the last two terms was very good—and our current students are already impatiently waiting for the lab task to be available. *Note for reviewers: We plan to have a formal evaluation in this term in time before the camera-ready version.*

6. SUMMARY AND CONCLUSIONS

We have presented the structure of our CS1 course that starts with the *How to Design Programs* teaching languages, followed by Java and a final lab on game programming. Our course certainly demands a lot from our students—but also gives them many learning opportunities and much support.

As the evaluation results show, students enjoy and appreciate the course. Their learning outcomes also show that they have learned a lot. The learning platform as outlined in Section 3.3 also received a very good evaluation. We hope that other educators will consider similar “unusual” elements in their courses, and encourage interested educators to contact the first author for in-depth discussion.

7. REFERENCES

- [1] Jason Cole and Helen Foster. *Using Moodle: Teaching with the Popular Open Source Course Management System*. O’Reilly, 2007.
- [2] Jeffrey Edgington. Teaching and viewing recursion as delegation. *J. Comput. Small Coll.*, 23(1):241–246, 2007.
- [3] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs: An Introduction to Programming and Computing*. MIT Press, 2 edition, 2001.
- [4] Michael Goldwasser and David Letscher. Teaching strategies for reinforcing structural recursion with lists. In *OOPSLA ’07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 889–896, New York, NY, USA, 2007. ACM.
- [5] Eric Roberts, Kim Bruce, Robb Cutler, James Cross, Scott Grissom, Karl Klee, Susan Rodger, Fran Trees, Ian Utting, and Frank Yellin. ACM Java Task Force support library. WWW: <http://jtf.acm.org/>, 2006.
- [6] Lon A. Smith and Elizabeth Turner Smith. Using camtasia to develop and enhance online learning: tutorial presentation. *J. Comput. Small Coll.*, 22(5):121–122, 2007.
- [7] Ben Stephenson. Using graphical examples to motivate the study of recursion. *J. Comput. Small Coll.*, 25(1):42–50, 2009.