Teaching Operating Systems Using Code Review

Christoffer Dall
Dept of Computer Science
Columbia University
New York, NY
cdall@cs.columbia.edu

Jason Nieh
Dept of Computer Science
Columbia University
New York, NY
nieh@cs.columbia.edu

ABSTRACT

Learning about operating systems often involves modifying a large and complex code base. Grading student projects can be difficult and time consuming, yet students often do not learn from their programming errors and struggle to understand core operating system concepts. We present GradeBoard, a code review system designed to simplify grading for instructors and enable students to understand and learn from their errors. GradeBoard provides an easy-to-use Web interface that allows instructors to annotate student code submissions with grading comments and scores, and students to discuss the comments and scores with instructors. GradeBoard presents student code changes with syntax highlighting and lets users collapse or expand code sections to provide a desired level of context, making it easier to read and understand student programming project submissions. Comments and scores are easily identifiable by visual cues, improving interaction between instructors and students. We have deployed and used GradeBoard in a large operating systems course involving Linux kernel programming projects. GradeBoard provided robust, easy-to-use functionality for reviewing Linux kernel code changes, improved the instructional staff grading experience, and over 90% of students surveyed indicated that GradeBoard improved their understanding of the kernel programming projects better than other alternatives.

Categories and Subject Descriptors: D.4.0 [Operating Systems]: General; K.3.1 [Computers and Education]: Computer Uses in Education–distance learning; K.3.2 [Computers and Education]: Computer and Information Science Education–computer science education

Keywords: Operating Systems, Code Review, Instructional Tools

1. INTRODUCTION

Programming projects are an important aspect of learning about operating systems (OSes) [8, 10]. Many approaches have been developed for providing such hands-on programming experience, including OS simulation environments, pedagogical OSes, and kernel development in commercial OSes. However, students often struggle with writing OS code and a disproportionate amount of time is spent by instructors evaluating student code without contributing to the actual learning process.

OS hands-on programming assignments typically provide students with some common software code base as a starting point for their programming projects, whether it be a small pedagogical OS in which students add substantial components or a large commercial OS that students modify to provide new features. Grading these projects can be especially hard because the projects involve not just writing an assignment from scratch in isolation, but modifying code throughout a potentially large and complex existing code base. To provide useful feedback to students to help them learn from their mistakes, students can be asked to give live demonstrations of their work, but this carries the risk of missing errors in handling important corner cases such as concurrency and synchronization, which are especially important and at the same time difficult to get right. Another approach is to run automated tests on student implementations, but test cases treat an implementation largely as a black box and do not cover subtle error cases or code design and readability.

To properly evaluate student work and give students useful feedback, it is crucial to review the code itself. This can be difficult to do for a variety of reasons. For example, many approaches simply provide feedback in terms of a grade sheet that summarizes the points deducted for various mistakes. This provides no direct correlation with the actual submitted code, leaving students with little choice but to try to hunt through numerous lines of code to identify the mistakes they made. Even if instructors refer to code using file names and line numbers, finding and following these references is tedious and error prone for both instructors and students, making it unlikely for students to learn from the process.

An alternative approach is to manually inline instructor comments directly in submitted code and make the commented code available to students to review. Unfortunately, this is at best a time consuming grading process for both instructors and students. Instructors are provided little help in grading numerous lines of code that may be embedded in large complex pieces of software, making the grading process difficult. Students must then use command line tools to find instructor comments with no visual structure or road map for figuring out where those comments might be and which ones might be of greatest importance for learning. The end result is that such grading is often a one-way process in which students see their grades but do not gain a complete understanding of their programming errors.

To address these problems, we present GradeBoard, a Web-based code submission and review system for providing grading feedback to students to help them learn more efficiently. GradeBoard provides a rich and flexible Web interface to make code more accessible, which helps instructors grade and comment on code and

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.
SIGCSE'14, March 5–8, 2014, Atlanta, GA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2605-6/14/03 ...$15.00.
http://dx.doi.org/10.1145/2538862.2538894.

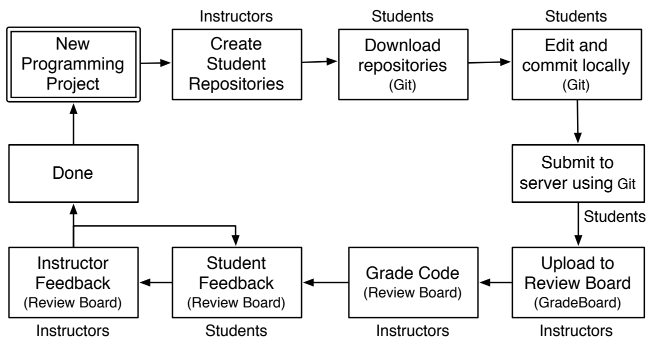

Instructors
Students
Students
New Programming Project
Create Student Repositories
Download repositories (Git)
Edit and commit locally (Git)
Done
Submit to server using Git
Students
Instructor Feedback (Review Board)
Student Feedback (Review Board)
Grade Code (Review Board)
Upload to Review Board (GradeBoard)
Instructors
Students
Instructors
Instructors

Figure 1: GradeBoard workflow

helps students read and understand code comments. Code changes are shown in a side-by-side *diff view* with code syntax highlighting, making it much easier to read and understand changes to a large existing code base than reading code diffs using command line tools. GradeBoard collapses large chunks of unmodified code, letting instructors focus on grading changed or new code, a feature especially useful when instructors need to consider small changes to large files. Students receive summaries of grading comments and see instructor comments as visual overlays in the code, which they can click on to ask questions about grading comments. Instructors can reply in one or several rounds to provide additional feedback. GradeBoard's Web-based interface is easily accessible from any computer with a Web browser without any further configuration or installation.

GradeBoard is based on standard widely used software development tools, the Git version control system [13] and the Review Board code review system [3]. This provides several benefits. First, because these tools are commercially supported, we can focus limited resources on teaching rather than spending time on in-house tool development. Second, because these tools are widely used in industry, students gain experience with real-world software development. Third, because these tools are open-source and widely-used in commercial settings, they continue to be developed and improved, which naturally evolves the tools and enables students to learn in a modern context.

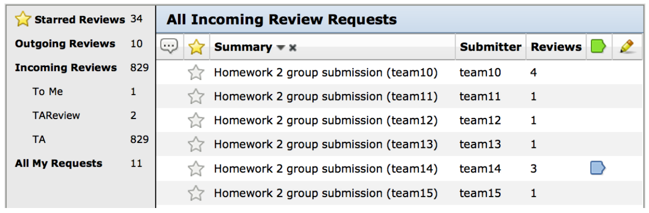
We successfully deployed and used GradeBoard in a large introductory OS course at Columbia University. Our experience shows that GradeBoard proved very useful for both instructors and students. Over 90% of students participating in a survey found that using GradeBoard improved their understanding of programming projects. Many students even said that using GradeBoard encouraged them to submit well-formatted code. The instructional staff found that GradeBoard significantly reduced time spent on grading kernel source code and reduced the number of grading mistakes by conveniently showing instructors the necessary context to code changes. Instructors were also very pleased with the ease of use and convenient overview of graded and ungraded projects. GradeBoard also facilitated more consistent grading by allowing instructors to inspect and discuss each other's grading.

2. USAGE MODEL

Figure 1 shows the GradeBoard workflow. It leverages a code distribution and submission system based on Git and a Web-based code review system based on Review Board. GradeBoard runs on an instructor-controlled server. Students can access GradeBoard from any computer and work on their programming projects on any computer on or outside the University network.

To assign a new programming project, an instructor creates a Git repository on the GradeBoard server containing the common code base used as a starting point for the student projects. For example, projects using Linux will typically use the Linux kernel as the common code base. Once the base repository is set, an instructor issues a single `create-hmwk` command and the repository is automatically duplicated, or *cloned*, for each student so that each student has a copy of the common code base. Student repositories can be cloned for individual students or teams of students, based on instructor preference.

To work on an assigned programming project, students download their repositories to their own computers using standard Git commands. Students can only access repositories assigned to them by the instructor. Students work on the code using their own computers and make incremental commits to their Git repositories using standard Git commands. Students are allowed to make commits and push those commits to the GradeBoard server up until a specified project deadline, at which point any further attempts to submit are blocked. The last pushed commit is considered the student's final submission to be graded.

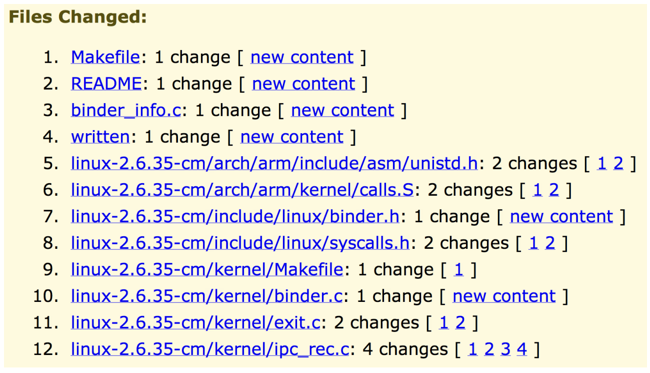


Starred Reviews	34
Outgoing Reviews	10
Incoming Reviews	829
To Me	1
TAReview	2
TA	829
All My Requests	11

All Incoming Review Requests

Summary	Submitter	Reviews
Homework 2 group submission (team10)	team10	4
Homework 2 group submission (team11)	team11	1
Homework 2 group submission (team12)	team12	1
Homework 2 group submission (team13)	team13	1
Homework 2 group submission (team14)	team14	3
Homework 2 group submission (team15)	team15	1

Figure 2: GradeBoard dashboard



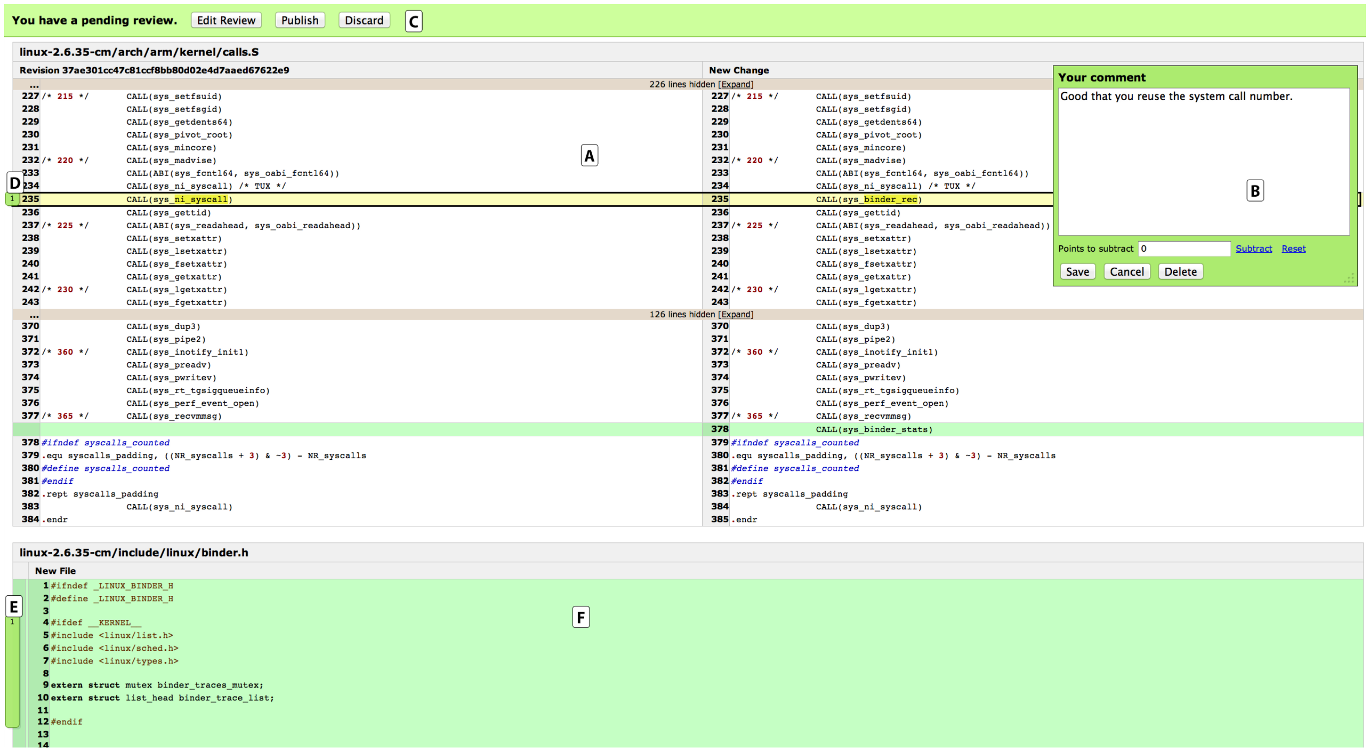
Files Changed:

1. Makefile: 1 change [new content]
2. README: 1 change [new content]
3. binder_info.c: 1 change [new content]
4. written: 1 change [new content]
5. linux-2.6.35-cm/arch/arm/include/asm/unistd.h: 2 changes [1 2]
6. linux-2.6.35-cm/arch/arm/kernel/calls.S: 2 changes [1 2]
7. linux-2.6.35-cm/include/linux/binder.h: 1 change [new content]
8. linux-2.6.35-cm/include/linux/syscalls.h: 2 changes [1 2]
9. linux-2.6.35-cm/kernel/Makefile: 1 change [1]
10. linux-2.6.35-cm/kernel/binder.c: 1 change [new content]
11. linux-2.6.35-cm/kernel/exit.c: 2 changes [1 2]
12. linux-2.6.35-cm/kernel/ipc_rec.c: 4 changes [1 2 3 4]

Figure 3: GradeBoard diff file overview

To upload the submissions to the GradeBoard review system for grading, an instructor simply executes a single `create-review` command, which causes GradeBoard to go through each student submission and upload and make it available for review through the Web-based Review Board system. As shown in Figure 2, a dashboard lists all the uploaded student submissions, along with information about each submission such as a summary description, the submitter, how many times the submission has been reviewed by instructional staff, and if there are any comments created by the specific instructional staff member currently logged in. Multiple instructional staff members is supported, which can include, for example, an instructor and multiple teaching assistants or graders. Instructors can add additional columns of information and can sort, filter, and search on all columns.

To grade a student submission, instructors select an ungraded submission from the dashboard and see a side-by-side diff view



linux-2.6.35-cm/arch/arm/kernel/calls.S

Revision 37ae301cc47c81ccf8bb80d02e4d7aaed67622e9

```
...                                 226 lines hidden [Expand]
227 /* 215 */        CALL(sys_setfsuid)
228                 CALL(sys_setfsgid)
229                 CALL(sys_getdents64)
230                 CALL(sys_pivot_root)
231                 CALL(sys_mincore)
232 /* 220 */        CALL(sys_madvise)
233                 CALL(ABI(sys_fcntl64, sys_oabi_fcntl64))
234                 CALL(sys_ni_syscall) /* TUX */
235                 CALL(sys_ni_syscall)
236                 CALL(sys_gettid)
237 /* 225 */        CALL(ABI(sys_readahead, sys_oabi_readahead))
238                 CALL(sys_setxattr)
239                 CALL(sys_lsetxattr)
240                 CALL(sys_fsetxattr)
241                 CALL(sys_getxattr)
242 /* 230 */        CALL(sys_lgetxattr)
243                 CALL(sys_fgetxattr)
...                                 126 lines hidden [Expand]
370                 CALL(sys_dup3)
371                 CALL(sys_pipe2)
372 /* 360 */        CALL(sys_inotify_init1)
373                 CALL(sys_preadv)
374                 CALL(sys_pwritev)
375                 CALL(sys_rt_tgsigqueueinfo)
376                 CALL(sys_perf_event_open)
377 /* 365 */        CALL(sys_recvmmsg)

378 #ifndef syscalls_counted
379 .equ syscalls_padding, ((NR_syscalls + 3) & ~3) - NR_syscalls
380 #define syscalls_counted
381 #endif
382 .rept syscalls_padding
383                 CALL(sys_ni_syscall)
384 .endr
```

New Change

```
227 /* 215 */        CALL(sys_setfsuid)
228                 CALL(sys_setfsgid)
229                 CALL(sys_getdents64)
230                 CALL(sys_pivot_root)
231                 CALL(sys_mincore)
232 /* 220 */        CALL(sys_madvise)
233                 CALL(ABI(sys_fcntl64, sys_oabi_fcntl64))
234                 CALL(sys_ni_syscall) /* TUX */
235                 CALL(sys_binder_rec)
236                 CALL(sys_gettid)
237 /* 225 */        CALL(ABI(sys_readahead, sys_oabi_readahead))
238                 CALL(sys_setxattr)
239                 CALL(sys_lsetxattr)
240                 CALL(sys_fsetxattr)
241                 CALL(sys_getxattr)
242 /* 230 */        CALL(sys_lgetxattr)
243                 CALL(sys_fgetxattr)
370                 CALL(sys_dup3)
371                 CALL(sys_pipe2)
372 /* 360 */        CALL(sys_inotify_init1)
373                 CALL(sys_preadv)
374                 CALL(sys_pwritev)
375                 CALL(sys_rt_tgsigqueueinfo)
376                 CALL(sys_perf_event_open)
377 /* 365 */        CALL(sys_recvmmsg)
378                 CALL(sys_binder_stats)
379 #ifndef syscalls_counted
380 .equ syscalls_padding, ((NR_syscalls + 3) & ~3) - NR_syscalls
381 #define syscalls_counted
382 #endif
383 .rept syscalls_padding
384                 CALL(sys_ni_syscall)
385 .endr
```

Your comment

Good that you reuse the system call number.

Points to subtract 0 Subtract Reset

Save Cancel Delete

linux-2.6.35-cm/include/linux/binder.h

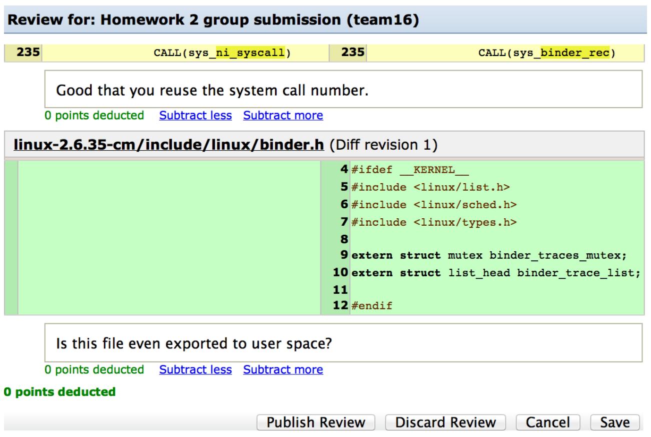
New File

```
1 #ifndef _LINUX_BINDER_H
2 #define _LINUX_BINDER_H
3
4 #ifdef __KERNEL__
5 #include <linux/list.h>
6 #include <linux/sched.h>
7 #include <linux/types.h>
8
9 extern struct mutex binder_traces_mutex;
10 extern struct list_head binder_trace_list;
11
12 #endif
13
14
```

Figure 4: GradeBoard instructor view of submitted homework assignment. Part (A) shows changes to an existing file. When clicking a line, the comment dialog for that line appears (B). When writing the first comment, the review status bar (C) appears at the top of the screen. Existing comments are shown as overlays on the line numbers for either single lines (D) or multiple lines (E). New files are shown as a single column (F). Numbers in comment overlays indicate number of comments on the line.

of the submitted code. As can be seen in Figure 3, the top of the diff view shows an overview of modified and added files. The overview shows the full path and file name and the number of changed code blocks for each file and reviewers can click on a file to scroll to changes in that file.

Figure 4 shows the main part of the instructor diff view. Each new file is clearly marked with a separate box with the full path and file name in the header of each box. The side-by-side diff view (A) shows the new version of a file on the right and the old version of a file on the left. New files are annotated as such and show only a single column (F). Source code is syntax highlighted to ease reading code. Green lines are new lines, yellow lines are modified lines, and red lines are deleted lines. Large chunks of unmodified lines are collapsed to a single gray line, and users can click to expand these lines. The ability to expand lines and see the full context of a modified file is especially useful for reviewing kernel code, where contextual information can be crucial to understand added code. Changes on a particular line are also highlighted so users easily can pinpoint the exact changes. Line numbers are shown on both the original and changed file. Instructors comment on a single line or a block of lines by simply clicking anywhere on these lines, causing a dialog window to appear (B) where instructors can easily enter comments. GradeBoard supports a live Web 2.0-style interface with ad-hoc commenting by continuously saving comments whenever a user edits them. GradeBoard stores a collection of comments to a code submission as a *review*. When the first comment is added, a review is automatically created and a review status bar (C) is added to the top of the screen. Reviews are initially created as *drafts* not accessible by other instructors until they are *published*, and not



Review for: Homework 2 group submission (team16)

```
235            CALL(sys_ni_syscall)  235            CALL(sys_binder_rec)
```

Good that you reuse the system call number.

0 points deducted Subtract less Subtract more

linux-2.6.35-cm/include/linux/binder.h (Diff revision 1)

```
4 #ifdef __KERNEL__
5 #include <linux/list.h>
6 #include <linux/sched.h>
7 #include <linux/types.h>
8
9 extern struct mutex binder_traces_mutex;
10 extern struct list_head binder_trace_list;
11
12 #endif
```

Is this file even exported to user space?

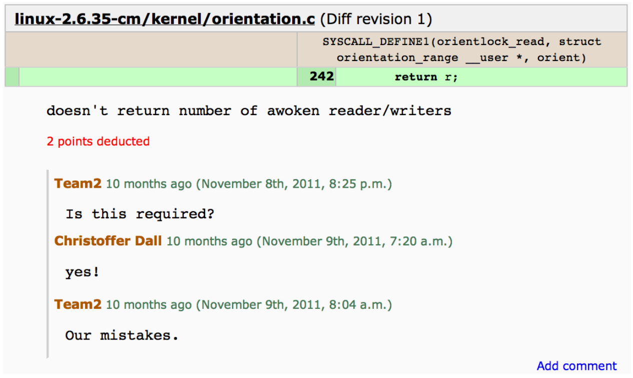
0 points deducted Subtract less Subtract more

0 points deducted

Figure 5: GradeBoard summary of review

accessible by students until they are *released*. Overlays (D) and (E) on the line numbers indicate instructor comments for those lines.

Instructors grade submissions by commenting on source code lines as described above and choose a number of points to deduct in the comment dialog for each grading comment. Instructors can also edit a draft review by clicking the Edit Review button in the review status tool bar, which presents a summary view allowing instructors to edit comments and subtracted points. Figure 5 shows the summary view, which displays the total number of deducted points and is useful to check the grading of a project before publishing the review. Publishing a review does not make the grading available to students, but lets other instructors access the


linux-2.6.35-cm/kernel/orientation.c (Diff revision 1)

```
SYSCALL_DEFINE1(orientlock_read, struct
  orientation_range __user *, orient)
242        return r;
```

doesn't return number of awoken reader/writers

2 points deducted

Team2 10 months ago (November 8th, 2011, 8:25 p.m.)

Is this required?

Christoffer Dall 10 months ago (November 9th, 2011, 7:20 a.m.)

yes!

Team2 10 months ago (November 9th, 2011, 8:04 a.m.)

Our mistakes.

Figure 6: GradeBoard inline grading comment discussion

review so they can discuss the grading style and points deducted internally before releasing grades to students. When all submissions have been graded, instructors can release all grades to the students at the same time by simply clicking a button in the Web interface.

Students review their grading by logging into GradeBoard where they can only see their own submitted projects. When students click on one of their submissions they are presented with a summary view similar to that shown in Figure 5 and a diff view similar to that shown in Figure 4, except that students cannot modify their grading score. Students see comments in the diff view as overlays on the line numbers and simply moving the mouse cursor over the overlays shows the comment. Students reply to comments directly in the summary view or by clicking comment overlays in the diff view. Students can prepare a *draft response* similar to how instructors create draft reviews. A *response* is a collection of questions on grading comments and students publish a response by clicking the Publish button on the response status bar analogous to the review status bar. There is no release step for students since there is no need for all student comments to be sent to instructors at the same time. GradeBoard shows the questions and answers as a threaded discussion under each grading comment in the summary view as shown in Figure 6. Instructors reply to student questions similarly to how students ask questions and this feedback loop can be repeated as many times as required to explain concepts to students.

Students and instructors are notified through e-mail when grades are released, when students ask questions about grading comments, and when instructors answer student questions. The e-mails contain a link to the submission being discussed and a summary of the grading comments and related discussions for that review. The e-mail summary additionally provides a link on each comment directing users to a response form for that comment on the Web interface.

Using GradeBoard to discuss grading comments improves the learning experience for students for a number of reasons. First, replying to a grading comment is faster and more convenient than sending an e-mail or attending office hours to ask for clarification, making it more likely that students ask questions about areas in which they struggle. An increase in the number of student questions on grading comments is a good thing, since answering these questions directly ensures that students understand the core concepts they struggle with. Second, GradeBoard's user interface provides a discussion type layout of the questions and answers to grading comments making it easy to track the conversation. The discussion thread is shown inline in the code so instructors and students can easily inspect the code being discussed and refer to it in questions and answers. Third, the entire instructional staff has access to the grading comments and discussions, which allows one instructor to answer a question when another instructor is not available and allows one instructor to further clarify the answer from another instructor.

3. SYSTEM ARCHITECTURE

GradeBoard consists of three major components: The first component is a source code revision system, Git, used to distribute a common code base to students, to collaborate on projects within student teams, and to submit final solutions to programming projects. The second component is a Web-based code review system based on the commercially maintained open-source software Review Board. The third component is a management tool used to integrate Git and Review Board for a teaching environment. Git is used completely unmodified and GradeBoard is only slightly modified from Review Board making it easy to leverage new features from new releases of both systems.

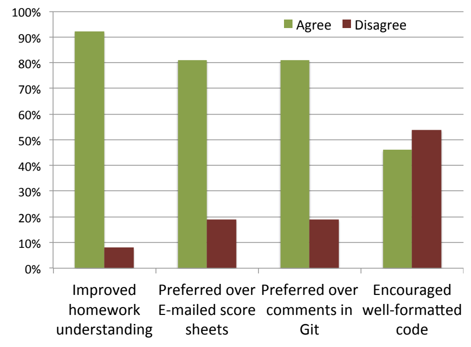
GradeBoard is easy to install and configure on any Linux server. We install the Git management system Gitosis [15] to manage repository access control and the Review Board application is easily installed by following the instructions on the official Review Board Web site [4] and applying our small patch.

The management tool provides instructors with a set of commands that are executed from the instructor's local machine and interact remotely with the GradeBoard server. GradeBoard management tools abstract low-level Git and Review Board commands from instructors and let instructors issue a single command on behalf of all students and their teams. For example, instructors issue a single `create-hmwk` command and all student template repositories for that programming project are automatically created. The management tool accomplishes this by maintaining a central list of enrolled students and their teams. When student enrollment changes during the course of the semester, instructors need only to change the central list of enrolled students, and the system adjusts authentication settings and group memberships accordingly to prevent, for example, that a student who dropped the course receives future e-mails from the system. Similarly, if a student joins the class late, an instructor simply adds that student and her team assignment to the list of enrolled students and GradeBoard automatically sends out a welcome e-mail with instructions on how to get started with the system to the new student.

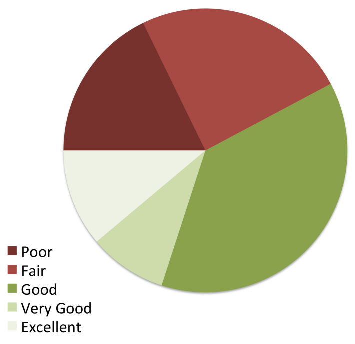
Instructors create template repositories for each student or student team from a single shared code base by first creating a base repository on the server, for example containing a copy of the Linux kernel. Instructors then use the management tool to clone the base repository for each registered student or student team. GradeBoard leverages existing Git features to use hard links for every shared file, inducing minimal disk space overhead. A standard version 2.6.35 Linux kernel repository consumes 492MB of disk space, which would amount to 50GB of disk space for 100 students if the repositories were to be simply copied or created from scratch for each student. Using hard links only consumes 128KB per extra unmodified clone, resulting in less than 500MB of total used disk space for all unmodified student repositories. In addition to imposing only modest hardware requirements, the low disk space usage also simplifies backup procedures.

The management tool command `create-review` utility collects all student changes to the base project source code repository and uploads these changes to the code review system. Review Board accesses the base repository to generate a combined diff of the changes and uses the original source code to show context when, for example, users expand a collapsed section in the diff view.

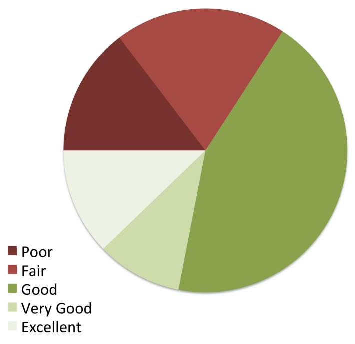
Review Board's authentication system supports creating user groups, which allows us to configure access permissions such that



(a) Overall student experiences



(b) Usefulness of discussion features



(c) Improved experience reading diffs

Figure 7: Survey Results

students can only access their own submissions and instructors can access all submissions. Review Board's default configuration lets submitters modify their own submissions without any restrictions, which does not work in a teaching environment where students cannot be allowed to modify a submission after a project deadline. To address this limitation we changed Review Board's default behavior to not allow users to upload or edit submissions. We also added a custom authentication setting that allows specific users to upload and edit all submissions, and we configure Review Board to only apply this setting to the instructor authentication group.

A modest number of simple changes are required to the standard Review Board lines of code in Review Board v1.6.1, less than 100 lines of Python code and about 200 lines of HTML/JavaScript have been added to support the authentication mechanism and calculating grading scores. These changes are non-intrusive to Review Board's standard functionality as a code review tool and do not modify existing features. They only add new optional features and can therefore easily be integrated in the standard Review Board code base without compromising other functionality. For example, we applied the same patch to two different versions of Review Board without any complications.

4. EXPERIENCES

We implemented and evaluated the use of GradeBoard in the Fall 2011 introductory OS class at Columbia University. During the course, six two-week programming projects were assigned [1]. Each project involved modifying the Linux kernel relating to a core OS subject and included a written non-programming part as well as the programming project. To run GradeBoard, we used a single central server with two Intel Xeon 3.06GHz CPUs, 2GB RAM, and 16GB RAID-1 storage, though more modest hardware could have been used. 2GB RAM was useful to keep the Linux kernel repository and Web server state in memory.

The GradeBoard commenting feature was heavily used. Students commented on over 80% of the graded programming projects, often asking for further clarification of a programming error. For example, a number of students did not fully understand problems in their programming project relating to synchronization and race conditions, a topic which students typically struggle to fully comprehend. GradeBoard's commenting feature allowed instructors to easily tailor the level of explanation detail for each student.

At the end of the course, we asked students to evaluate their experiences using GradeBoard. Out of more than 100 students enrolled at the end of the semester, 57% of the students, completed the evaluation survey. Figures 7a, 7b, and 7c show the results of the survey. More than 90% of the students who completed the survey said that the use of GradeBoard improved their understanding of the programming projects by being able to see comments from instructors directly in context with the submitted source code lines. More than 80% of the students preferred GradeBoard to e-mailed score sheets or viewing instructor comments committed to source code repositories. Almost half the students who completed the survey, said that using GradeBoard encouraged them to submit easier-to-read, more well-formatted code. Finally, the majority of the students found GradeBoard useful for discussing grading with the instructional staff, and improved the experience of reading code diffs compared to reading flat diff files. However, a few students also felt that the GradeBoard Web interface was difficult to learn.

In addition to being well received by students, GradeBoard was tremendously beneficial for the instructional staff. TAs were very pleased with the interface and felt that it significantly improved the experience of grading and strongly preferred using GradeBoard compared to manually have to write up grade sheets or clone repositories and add comments to student code. This was confirmed by TAs from previous years, who told us they would have strongly preferred using GradeBoard compared to other alternatives.

GradeBoard also turned out to be helpful when interacting in person with students, for example during office hours, where students would often ask questions about grading comments or would ask for help to understand design flaws or other errors in their submission. In this situation, instructors and students benefited from accessing GradeBoard on a shared computer to inspect and discuss the code and comments together. This avoided the need to search for previous e-mail discussions, look up grade sheets, or spend time waiting to download the student source code, improving the efficiency and experience of office hours for both instructional staff and students alike.

5. RELATED WORK

Some structured grading systems have been developed. Version control systems have been proposed [7, 11] for grading feedback by allowing instructors to directly edit student assignments and commit grading comments in student source code repositories. This command line based approach is inconvenient and does not provide an easy-to-use graphical or Web interface for browsing through graded programming assignments. This is problematic as the students who need the most feedback are those who are also most likely to have less developed skills at reading patches directly and using version control tools to be able to find and review grading feedback. In contrast, GradeBoard does not require students to use any command line utilities to read and understand grading comments. GradeBoard's Web interface is easy-to-use and provides syntax highlighting and the ability to view the full context of code changes combined with visual cues to instructor comments and student feedback.

Praktromat [16] provides basic automatic testing of submissions and allows students to review each other's programming projects. However, Praktomat only provides a text dump of the entire programming project submission. Unlike GradeBoard, it does not provide any way to view only the changes made to a common code base, and lacks a convenient easy-to-use dynamic and graphical Web-based interface for code review.

Caesar [12] focuses on social reviewing where code submissions are broken up into chunks and each chunk is reviewed by different reviewers, who can be other students or members of the instructional staff. Caesar is not suitable for OS kernel programming projects where the amount of code may be small, but extremely dense, and solutions must be read as a whole to verify correctness. Caesar lacks GradeBoard's ability to easily switch back and forth among different views of a code submission, seeing parts of files, whole files or only code changes, all valuable features especially for reviewing OS programming projects.

CodeWave [14] provides an Integrated Development Environment (IDE) with shared real-time editing and record/replay functionality focusing on evaluating the development process in addition to final submissions. However, its IDE may not be suitable for OS programming and is not designed for viewing changes to a large common code base. CodeWave also lacks GradeBoard's ability to easily switch between different views of a code submission.

MarkUs [9] is an open-source Web-based tool for grading student submissions. However, it is designed to only show complete submitted files and not changes to an existing large code base. MarkUs's inability to view diffs limits its utility for reviewing any programming projects involving large common code bases, such as OS kernel programming projects. OSBLE [6] is another Web-based code review tool that does not support reviewing code changes to a common code base and lacks GradeBoard's ability to easily switch back and forth among different views of a code submission. These pedagogical code review systems have to be separately maintained by a limited community of volunteer open-source developers or students, resulting in maintenance overhead and limited functionality. In contrast, GradeBoard builds on open-source tools that are already widely used in commercial software development with strong commercial community support, ensuring robust functionality that can scale to support large code bases and programming projects, such as those used in many Linux-based OS courses.

While GradeBoard builds on Review Board, other commercial code review systems are also available. Our experience with those systems indicates that they may be less well-suited for providing grading feedback. For example, Gerrit [5], used for the Android Open Source Project (AOSP), does not provide a convenient summary view of a code review, requiring students to click through all submitted files to look for annotations that could be used for instructor comments. It is built around a vastly more complicated authentication model and has a more complicated and less intuitive user interface. As another example, Atlassian Crucible [2] is proprietary, cannot be easily modified to work in a teaching environment, and may incur licensing fees.

6. CONCLUSIONS

We have developed GradeBoard, a Web-based code review system that saves time for both teachers and students by simplifying the process of reading, grading, and commenting on student code submissions. GradeBoard makes it easy to identify student code changes to an existing code base with syntax highlighting and lets users collapse or expand code sections to provide the desired level of context. These features make GradeBoard especially useful for teaching operating systems, since students may modify code throughout a large and complex code base. Students learn more from hands-on programming projects when using GradeBoard for two reasons. First, grading comments are much easier to understand as they are shown as annotations directly on student code. Second, students and instructors can easily engage in dialog directly in context with the grading comments, ensuring that students fully understand and learn from their mistakes. Because GradeBoard builds on widely used commercial tools, it is easy to maintain and keep up with commercial practice, and introduces students to standard code review procedures and tools, enhancing their educational experience. Our experiences teaching operating systems using GradeBoard show that GradeBoard is preferred by students over other alternatives, improved students' understanding of their homework code, and significantly improved the instructors' experience of grading compared to other alternatives.

7. ACKNOWLEDGEMENTS

This work was supported in part by NSF grants CNS-1162447, CCF-1162021, and CNS-1018355.

8. REFERENCES

[1] J. Andrus and J. Nieh. Teaching Operating Systems Using Android. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2012)*, pages 613–618, Mar. 2012.

[2] Atlassian. Crucible. `http://www.atlassian.com/software/crucible/`.

[3] Beanbag Inc. Review Board. `http://www.reviewboard.org`.

[4] Beanbag Inc. Review Board Installation Guide. `http://www.reviewboard.org/docs/manual/dev/admin/installation/linux`.

[5] Google. Gerrit Code Review. `http://code.google.com/p/gerrit`.

[6] C. Hundhausen, A. Agrawal, and K. Ryan. The Design of an Online Environment to Support Pedagogical Code Reviews. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE 2010)*, pages 182–186, Mar. 2010.

[7] O. Laadan, J. Nieh, and N. Viennot. Teaching Operating Systems Using Virtual Appliances and Distributed Version Control. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE 2010)*, pages 480–484, Mar. 2010.

[8] O. Laadan, J. Nieh, and N. Viennot. A Structured Approach to Linux Kernel Projects for Teaching Operating Systems. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE 2011)*, pages 287–292, Mar. 2011.

[9] MarkUs Project. MarkUs. `http://www.markusproject.org`.

[10] J. Nieh and C. Vaill. Experiences Teaching Operating Systems Using Virtual Platforms and Linux. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE 2005)*, pages 520–524, Feb. 2005.

[11] K. L. Reid and G. V. Wilson. Learning by Doing: Introducing Version Control as a Way to Manage Student Assignments. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE 2005)*, pages 272–276, Feb. 2005.

[12] M. Tang. Caesar: A Social Code Review Tool for Programming Education. Master's thesis, Massachusetts Institute of Technology, Sept. 2011.

[13] L. Torvalds. Git. `http://git-scm.com`.

[14] J. Vandeventer and B. Barbour. CodeWave: A Real-Time, Collaborative IDE for Enhanced Learning in Computer Science. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2012)*, pages 75–80, Feb. 2012.

[15] T. Virtanen. Gitosis: Software for Hosting Git Repositories. `https://github.com/res0nat0r/gitosis`.

[16] A. Zeller. Making students read and review code. In *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000)*, pages 89–92, Sept. 2000.