

DrProject: A Software Project Management Portal to Meet Educational Needs

Karen L. Reid
Dept. of Computer Science
University of Toronto
reid@cs.utoronto.ca

Gregory V. Wilson
Dept. of Computer Science
University of Toronto
gvwilson@cs.utoronto.ca

ABSTRACT

DrProject is a web-based software project management portal that integrates revision control, issue tracking, mailing lists, a wiki, and other features. Unlike other such systems, DrProject is specifically tailored to the needs of undergraduate teaching and team programming assignments. We describe the pedagogical motivations for DrProject and our experiences with it to date.

Categories and Subject Descriptors

K.3 [Computing Milieux]: Computers and Education;
D.2.6 [Software]: Software Engineering, Programming Environments

General Terms

Management

Keywords

Education, Software tools, Software Engineering, Courseware

1. INTRODUCTION

Real-world software engineering is a collaborative activity: programs are designed, developed, and maintained by teams. Learning how to work productively with others should be a central part of undergraduate education in software engineering [2, 4].

Project teams in industry, academia, and the open source community are increasingly reliant on web-based project management portals [5]. The best known of these, SourceForge [10], currently hosts over one hundred thousand projects and over a million users. It, and imitators like GForge, CollabNet, and CodeHaus, are now as important to developers as IDEs like Eclipse.

Learning how to use such portals should therefore be a central element of software engineering education. In classroom settings, these tools help students coordinate their

efforts and use their time more efficiently. They give instructors a concrete environment in which to teach basic teamwork skills such as time management and communication, as well as a hands-on starting point for teaching larger software engineering issues such as maintenance and release management.

More importantly, these tools encourage truly collaborative work. Many student teams divide the work into pieces, then toil away individually until a “big bang” integration shortly before the due date. By helping students see that software development is a social activity, rather than an anti-social one, use of collaborative tools may help attract and retain students from at-risk groups [7].

The only way for students to master these tools is through practice. However, the setup overheads of industrial-strength tools are often intimidating. What’s more, many of their features are over-engineered for six-week assignments undertaken part-time by teams of three to five students. As a result, many students come away from their first contact with these types of tools believing that they make life more difficult, rather than less.

Conversely, existing systems lack capabilities that are crucial in classroom settings: there is no easy way to create dozens of similar projects at once, and many systems do not hide any information, for the simple reason that “open” projects don’t need to do so.

We therefore decided to build a portal that would be for teams what DrJava [1] and BlueJ [8] are for individuals: simple enough to learn in an hour, but with the key components of full-featured systems. Our goal was *not* to build a course management system—grading, lecture notes, and so on are well handled by existing courseware—but rather to create a simplified, realistic, and above all compelling alternative to full-strength software project management systems.

2. REQUIREMENTS

Software project management portals offer a common core of features [3]:

- A *version control system* is a tool that coordinates changes to shared files, such as source code, while recording the history of those changes. Most such systems keep the master copies of the files in a networked central repository; developers work in personal mirrors of this while periodically synchronizing with the repository. We have successfully introduced version control into our second-year programming courses as a first step in learning tools to support good working practices [9].

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’07, March 7–10, 2007, Covington, Kentucky, USA.
Copyright 2007 ACM 1-59593-361-1/07/0003 ...\$5.00.

Most industrial-strength portals include a *repository browser* that allows users to see the current and past state of files, change sets, etc. This is almost always read-only, since allowing people to update and commit through their web browser raises security issues. Users check out and commit files to the repository using separate standard tools.

- An *issue tracking system* is used to store bug reports, change requests, ideas for enhancements, and general to-do items in a central database. These systems typically allow users to classify entries (often called “tickets”) by priority, keep track of who is supposed to be working on them, and so on. They usually also allow users to associate tickets with deadlines, so that project members can quickly get an overview of what needs to be done by when.
- A *wiki*, which acts as a shared online whiteboard for the project.
- *Mailing lists*, so that team members can communicate.
- A *search engine* to help users find project-specific information.
- Some form of *account management*. Most systems support self-registration; some also allow users to join projects on their own, while others leave that in the hands of the project administrator.
- Extra features, such as status dashboards showing project progress over time, continuous build systems that recompile code and rerun tests every time a change is checked in, and per-project blogs.

These features are all as useful in courses as they are in real life, with the exception of account management. Educational institutions typically have well-developed systems for creating student accounts, checking the strengths of passwords, locking down accounts when courses are finished, and so on. These systems also typically handle access control issues, for example by ensuring that student accounts are in the right Unix groups for particular courses. We therefore decided that account creation and management should not be part of our system: users are required to have an account on the underlying host in order to use DrProject.

However, industrial-strength systems lack some features required for classroom use. The first is customizable roles and permissions. Most open source projects divide the world into administrators, developers, and viewers. In courses, however, we need to differentiate between professors, TAs, students working on particular projects, students who can view those projects but not alter them, students for whom those projects are opaque, and so on. More importantly, we need to be able to customize these permissions to fit the pre-existing rules of different educational institutions.

Another key feature that we needed was an automation interface. An instructor may need to create several dozen identical projects at the beginning of the term, something that doesn’t happen in an industrial setting. An instructor may also want to assign the same work item to each team in the course, to commit the same handful of files to every team’s repository, or set the same milestones in each project.

3. SYSTEM EVOLUTION

Web-based project management systems are invariably built using a three-tier architecture. The user’s web browser communicates with a web server, which passes HTTP requests to a CGI or servlet “engine” that uses a relational database for persistent storage. The engine integrates with a pre-existing version control system, such as CVS or Subversion; depending on the system’s capabilities, it may also integrate with a mail transfer agent, authentication services such as LDAP or Kerberos, and so forth.

In 2003-04 we developed three prototypes of DrProject using this architecture: one from scratch using Java Server Pages, a second that modified GForge, and a third (also from scratch) using Hibernate, Tapestry, and other “New Standard Model” Java tools. These experiences were disappointing: the Java tools’ learning curve was so steep that it would have been impossible to maintain the system with student labor, while there were so many security problems in GForge that further work seemed pointless.

We started over in 2005, using a young open source tool created by Jonas Borgström and Daniel Lundin called Trac [11] as our starting point. Trac consisted of roughly 11,000 lines of Python, plus a further 3000 lines of HTML templates. Five undergraduate students worked full-time on the project in the summer of 2005; progress was rapid, and we deployed the system in three courses in the fall of that year. Further work was done in January 2006, and two more undergraduates worked on it again in the summer of 2006.

DrProject 1.1, released in September 2006, contains 21,250 lines of Python source (including blank lines, comments, and embedded documentation strings), a further 13,000 lines of test code, and 3600 lines of HTML templates. It is built on top of a Postgres database and Subversion repositories. As the screenshot in Figure 1 shows, it offers all of the key features identified earlier, as well as tagging (to facilitate search) and a timeline of recent events.

4. DRPROJECT FEATURES

Each project in DrProject 1.1 has the following components:

Wiki The wiki ties all the components of DrProject together. Using wiki syntax, links can easily be created to refer to a specific ticket, changeset, mail message, or wiki page.

Subversion repository Each project has a separate Subversion repository. This helps isolate projects from one another and also simplifies revision numbering.

Tickets We tried to minimize the number of fields in a ticket to make it as easy as possible to use while still providing the key features. For example, tickets can be in only two states, open or closed. We eliminated other possibilities such as rejected, completed, and reopened, on the grounds that students were unlikely to see the value in these distinctions in a short-lived project.

Mailing list The mailing list is simply a relay plus an archive. Users white-list the email addresses they want to send from, and set an address for receiving email.

Roadmap Users (or instructors) can set milestones with either a fixed date or a symbolic name, and assign

dp dr project

Logged in as reid (logout | preferences)

Current project: **DrProject** -- Change Project --

Ticket	Summary	Milestone	Owner	Priority	Reporter	Type	Created
62	Need single-step archive of system	someday	nobody	high	gvwilson	enhancement	2006-01-06 10:23:00
278	Unicode Audit	Xenon	nobody	high	sdawson	task	2006-05-04 16:30:26
334	Subversion authentication against our internal database.	someday	apple	high	apple	enhancement	2006-06-13 15:52:07
338	Create Authenticator component interface.	someday	apple	high	apple	enhancement	2006-06-15 12:32:58
358	In the trunk version, once you remove the 'Status' filter, you can't re-add it again.	Xenon	dscannell	high	apple	defect	2006-06-26 15:31:54
449	Installation cannot find Setuptools	Radon	g3greg	high	glapouch	defect	2006-08-12 19:00:41
466	Stack trace when trying to	Radon	g3greg	high	gvwilson	defect	2006-08-28

Figure 1: The Ticket Summary View in DrProject

tickets to a milestone. The “Roadmap” view shows how many tickets assigned to each milestone have been completed and how many remain.

Tags Tags can be assigned to tickets and wiki pages. This allows users to group items according to keywords. The “Tag cloud” view shows all the tags in a project sized according to frequency. A cross-project tag cloud shows all the tags in all the projects sized according to frequency and coloured by the tag’s appearance in the current project. Clicking on a tag in the cloud takes the user to a list of links for that tag. Users only see links from projects that they have permissions to view.

Administrative Interface Administration is done using either a command-line tool or a web interface. The command-line tool makes it easy to create projects, add users, assign users to projects, load wiki pages, assign tickets to all projects, and collect information about the project. The web interface simplifies small changes such as moving a user from one project to another, adding a new user or project, and locking or suspending permissions on a project.

5. BENEFITS TO STUDENTS

As stated earlier, most real software development is done in teams, so learning the tools and techniques needed for effective teamwork should be an integral part of computer science education. In the past, we used the same “sink or swim” approach as most schools: senior students were given large projects to do in groups, and a few lectures on software process, but left to figure out tooling and day-to-day operations themselves.

DrProject corrects this by making it easier for students to “do the right thing”. Like any other tool, DrProject encourages its users to work a certain way: to communicate through a searchable mailing list instead of point-to-point, to keep track of who’s responsible for what by assigning and closing tickets, and so on. In teaching students how to use DrProject, we are also teaching them how to divide up

work, manage time, coordinate with others—in short, how to tackle large projects systematically.

The result from the students’ point of view is that they are able to do more, and better, than before: fewer “dropped balls” and overwritten files translates directly into higher grades (and lower stress). Using DrProject also makes students better prepared for careers in both research and in industry, where coordination tools of this kind have become the norm.

Finally, DrProject makes it easier for student teams to exchange and inherit projects. This has been a key part of our software engineering course for many years [6]: part way through the term, each team must choose another team’s project and continue working with their code for the rest of the course. Teams get bonus marks each time their code is used by another team. Having all the key information about the project in one (searchable) place makes it easier for students to investigate other projects, and encourages teams to present a well-organized and documented system to their peers.

6. BENEFITS TO INSTRUCTORS

DrProject makes it easy for instructors to create project spaces for student groups, move students between groups, perform batch operations on all groups such as adding files to all repositories, or filing tickets against all groups, or send an email message to all the members of a group. This in turn makes it possible for instructors to set more ambitious goals for project courses, since it frees up time that would otherwise have to be spent on administrative duties.

Tools like DrProject have been used elsewhere to give instructors insight into what the students are doing at intermediate points in the project, and a way to spot teams that are struggling early on [12]. We are now studying the data collected by DrProject to learn more about how our students work together, which we hope will in turn tell us aspects of teamwork they need more training in.

One other benefit of DrProject has been the experience we have gained from supervising its construction. Having several students adding features to DrProject every term

helps us stay current with new technologies, while giving the students an opportunity to work on a project with a real customer. More importantly, it gives us “war stories” to tell in class. Quotes from textbooks about how important it is to start testing early, or how much time deployment can take, are one thing; pointing at a system students are using, and telling them what went wrong as it was being built, is quite another.

7. EXPERIENCES TO DATE

We have used DrProject in three different fourth year courses and one third year course: CSC408 (Software Engineering), CSC488 (Compilers and Interpreters), a pair of research project courses (CSC494/495), and CSC369 (Operating Systems). In the first two terms that we used DrProject, we were reluctant to rely on it too heavily. Despite this, students used it continuously throughout the year, and as we report below, their experiences were generally very positive.

Figure 2 shows the number of tickets, version control check-ins and wiki edits by each group in the CSC408 and CSC488 courses. Within each course, the groups in the graph are sorted by the number of tickets filed. We felt that the ticket system usage was a better indicator of how much students relied on DrProject than version control check-ins, since (a) students could interact with the version control system using other tools, and (b) in some cases, they *had* to check in their code in order to receive a grade.

CSC408, a required course, covers project management and the software development process. Students work in teams of four or five on a term-long project. Part way through the term, the teams swap code and continue working on another group’s code. In a survey at the end of the term, approximately half the respondents reported that they relied on DrProject to some extent, while the other half relied on it very little. Two-thirds of the students found it easy to learn to use DrProject. A few students commented that more tutorial information on DrProject would be helpful; we have accordingly produced a short screencast demonstrating its most important features.

In each instance of CSC408, nearly half the groups made non-trivial use of the DrProject, filing more than twenty tickets over the first 7 weeks. New projects were created for them within DrProject after they swapped code with another group. The usage rate following the swap followed the same pattern as before the swap.

CSC488 (Compilers and Interpreters) is optional, and had a smaller enrollment. Students reported that they found the ticket system overkill for the size of their projects, but usage patterns indicated that they still took advantage of the ticket system.

Five out of the seven groups in the Compilers course made significant use of DrProject. We believe there are three reasons why students used DrProject more in this course than in the Software Engineering course: as an optional and challenging fourth year course, it has a higher proportion of our top students; most of the students in the course had previous experience with software project management tools from their employment; and the TAs were more enthusiastic about DrProject.

DrProject was most heavily used in CSC494/495, a project course where students work individually or in small teams on a project in close collaboration with a faculty member.

Since the second author of the paper was the primary supervisor of these projects, students in this course received more training in how to use DrProject, and therefore used it more effectively than in the other courses. Students working on solo projects tended not to use DrProject very much, but as the team size increased the number of transactions on DrProject also increased, showing that students were using DrProject to coordinate their team work.

This term, we began using DrProject in CSC369 (Operating Systems) primarily to take advantage of the easy setup for the Subversion repositories. Students work in teams of two or three on an OS simulator and also have one individual assignment (leading to the creation of more than 75 projects within DrProject). We have been using the tickets to give students a formal mechanism to identify weaknesses or remaining flaws in their code. The ease of updating multiple repositories with new starter code and managing group permissions were enough to convince us to keep using DrProject in future offerings of this course.

Looking at general usage patterns, we find that some groups use the wiki a great deal. It is interesting that these groups are not always the ones who also use the ticket system. This shows that some students found value in having an unstructured space for collaboration. By looking for imbalances in usage patterns within each team, we can identify teams that may be running into difficulty.

We expect to see even greater use of DrProject this year, as instructors and TAs become more familiar with the system. The performance and feature improvements over the summer will also improve the uptake. DrProject is now in use at Queen’s University, and is being evaluated for in-house use at several local companies.

8. FUTURE WORK

Now that we have a robust system, we plan to conduct more empirical studies into how our students approach teamwork, and how we can better support their work. We also plan to gather feedback to determine if we have chosen the right set of features, and what other features would most help the students. By providing more tutorial material, and training students to use DrProject more effectively, we hope to improve the fraction of students using DrProject in each course.

We know that we need to encourage students to write more documentation. To this end we have two new features planned. The first is a team blog which could be used in a variety of ways. For example, students could write a journal about their team experience, write tutorials on bits of technology they are using, or publish design ideas. The second feature is a trigger to automatically generate Javadoc or other automatically formatted documentation and display it within DrProject. We hope that if students can easily view the documentation for their system, they will be more inspired to write the documentation early, and follow a standard format.

Documentation generation is the first step toward including continuous integration tools. We would like students to be able to build and test their code automatically upon check-in. Determining how to protect the server from malicious attacks is the thorny problem we need to solve to implement true continuous integration.

Currently, DrProject authenticates users against the system password file (`/etc/passwd`). We are working on de-

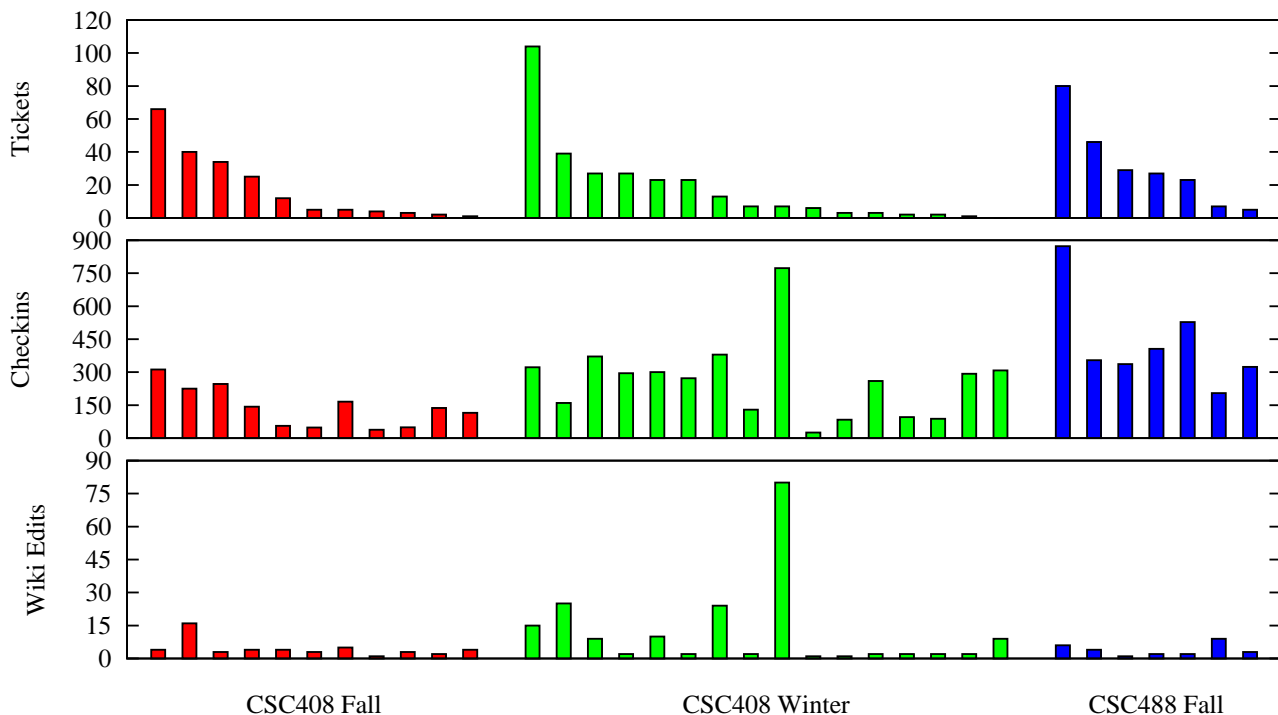


Figure 2: DrProject usage by project group for three courses in 2005/2006. In each course the bars are ordered by number of tickets filed.

veloping other authentication mechanisms to make it easier to deploy DrProject in different environments.

9. AVAILABILITY

DrProject is an open source project under a liberal BSD-style license. We actively encourage other instructors to adopt it in their courses. Please contact one of the authors, or see <http://www.drproject.org>.

10. ACKNOWLEDGMENTS

We wish to thank the students who worked on DrProject, and the faculty who have adopted it. We also wish to thank the Natural Sciences and Engineering Research Council of Canada (NSERC), the Jonah Group, Perforce Inc., Idée Inc., IBM's Eclipse Innovation Grants programme, the University of Toronto's Instructional Technology Courseware Development Fund (ITCDF), and Prof. Steve Easterbrook for their support.

11. REFERENCES

- [1] Eric Allen, Robert Cartwright, and Charles Reis: "Production Programming in the Classroom", *Proc. SIGCSE 2003*, pp. 89-93. See also <http://drjava.sourceforge.net/papers/index.shtml> (viewed October 5, 2005).
- [2] David Coppit: "Implementing Large Projects in Software Engineering Courses", *Journal of Computer Science Education*, 16(1), March 2006.
- [3] Matthew B. Doar: *Practical Development Environments*. O'Reilly, 2005.
- [4] Sally Fincher, Marian Petre, and Martyn Clark (eds.): *Computer Science Project Work*. Springer, 2005.
- [5] Karl Fogel: *Producing Open Source Software*. O'Reilly, 2005.
- [6] J.J. Horning and D.B. Wortman: "Software Hut: A Computer Program Engineering Project in the Form of a Game". *IEEE Transactions on Software Engineering*, 3(4), July 1977 pp. 325-30.
- [7] Jane Margolis and Allan Fisher: *Unlocking the Clubhouse: Women in Computing*. MIT Press, 2002.
- [8] Michael Mölling, Bruce Quig, Andrew Patterson, and John Rosenberg: "The BlueJ system and its pedagogy", *Journal of Computer Science Education*, 13(4), December 2003. See also <http://www.bluej.org/papers/> (viewed September 3, 2006).
- [9] Karen L. Reid, and Gregory V. Wilson: "Learning by Doing: Introducing Version Control as a Way to Manage Student Assignments", *Proc. SIGCSE 2005*, pp. 272-276.
- [10] SourceForge: <http://www.sourceforge.net> (viewed September 3, 2006).
- [11] Trac: <http://trac.edgewall.org> (viewed September 3, 2006).
- [12] K. Wong, W. Blanchet, Y. Liu, C. Schofield, E. Stroulia, and Z. Xing: "JReflX: Towards supporting small student software teams", *Proc. Eclipse Technology Exchange Workshop*, pp. 56-60, OOPSLA 2003.