# Production Programming in the Classroom

**Eric Allen**
**Rice University**
**6100 S. Main St.**
**Houston TX 77005**
**eallen@cs.rice.edu**

**Robert Cartwright**
**Rice University**
**6100 S. Main St.**
**Houston TX 77005**
**cork@cs.rice.edu**

**Charles Reis**
**Rice University**
**6100 S. Main St.**
**Houston TX 77005**
**creis@cs.rice.edu**

## Abstract

Students in programming courses generally write "toy" programs that are superficially tested, graded, and then discarded. This approach to teaching programming leaves students unprepared for production programming because the gap between writing toy programs and developing reliable software products is enormous.

This paper describes how production programming can be effectively taught to undergraduate students in the classroom. The key to teaching such a course is using Extreme Programming methodology to develop a sustainable open source project with real customers, including the students themselves. Extreme Programming and open source project management are facilitated by a growing collection of free tools such as the JUnit testing framework, the Ant scripting tool, and the SourceForge website for managing open source projects.

## Categories & Subject Descriptors

K.3 *Computers & Education*: Computer & Information Science Education - Computer Science Education.

## General Terms

Management, Documentation, Design, Reliability, Experimentation, Human Factors, Languages.

## Keywords:

Extreme Programming, Open Source, DrJava, Production Programming, Software Engineering, SourceForge, Ant, JUnit.

## 1 Introduction

Despite the best efforts of universities to teach effective software engineering, few students are properly prepared for professional software development [5]. There are two primary reasons for this deficiency in the computer science curriculum. First, until recently, there was little consensus on which software development model should be used to build production software systems. Second, the resources and constraints that characterize production programming have been difficult if not impossible to reproduce in the classroom [7].

Production programming involves creating or modifying a software product to meet the needs of real customers. Writing large programs is not enough. Without real customers, student programmers are not held accountable for producing software that works "in the field" and has a usable interface. Of course, it is unrealistic to expect an instructor to become a software entrepreneur who produces and markets software products solely for the sake of teaching a course. However, it is possible for an instructor to adopt an existing open source project and develop course assignments that extend this code base to meet new requirements identified by the project's customers.

Not every open source project is a good candidate for use in the classroom. Many such projects have high "barriers to entry"—such as inadequate documentation, poor coding style, and the lack of comprehensive unit tests—that new developers must overcome before they can reliably extend the code base.

The key to making an open source project accessible to students is to augment the code with documentation and comprehensive unit tests. The unit tests serve two critical functions. First, they enable new developers to extend the code base without breaking existing functionality. Second, they serve as formal specifications that stipulate how each method should behave in both typical and pathological cases.

When the authors converted the Rice software engineering course from toy programming to production programming last spring, we already owned an open source project that was ideal for use in the classroom because of its development model, high quality, and familiarity to the course staff. During the preceding nine months, our research group had developed a new pedagogic programming environment called DrJava, using *Extreme Programming* (XP) methods [6]. The customers of this project included many students, at Rice and other institutions, who used DrJava for their coursework.

## 2 DrJava as a Course Project

The DrJava integrated development environment (*IDE*) was designed explicitly to make programming in Java accessible to beginners. Its signature trait is an integrated "interactions pane", providing a read-eval-print loop to manipulate code defined in the definitions pane (an editor that understands the rudiments of Java syntax) [1]. As it has matured, we have found that DrJava also scales well to the development of larger and more advanced applications. In fact, all development on DrJava is done within DrJava itself, making our development team a major onsite customer. Furthermore, because this tool was designed to be easily accessible to beginners, we have concentrated on making its behavior well-defined and reliable, a benefit for users at any level [1].

In our software development process, we have enforced a policy of providing a comprehensive suite of unit tests for every non-trivial method in the program. We have also adhered as closely as possible to the other major tenets of XP, including:

- writing most of the code using pair programming to allow for effective knowledge transfer and to audit the code as it is being written
- developing the program in small, well-defined increments where each increment is described by a brief "story" specifying how the new code will affect program behavior
- composing most of our unit tests prior to writing the code to help clarify our specifications and to enforce a discipline of comprehensive unit testing
- refactoring our code whenever it becomes clear that the structure of the program could be significantly streamlined and simplified [3].

These practices enabled us to develop DrJava very quickly with scarce resources. Yet DrJava is quite robust, and it is currently used in a number of universities and high schools for teaching introductory Java courses, as well as by independent developers who prefer a light-weight but powerful development environment. Several of our customers rely on DrJava to teach hundreds of students, which makes them as demanding as the customers of many commercial software products. The latest release of DrJava is available on the web at http://drjava.sourceforge.net.

Because DrJava was written to be extensible by an ever-changing team of programmers, it is an ideal application for extension in the classroom. Adding students to the staff of a production system for the duration of a course creates a very high rate of developer turnover. Only extreme measures such as those in Extreme Programming can cope with such high turnover.

For the inaugural version of the course, our research group created a list of projects that addressed the most pressing bugs and feature requests for DrJava. In accord with XP management practice, students were allowed to sign up for projects that reflected their individual interests. Since we had three project managers, we selected three major extensions to DrJava as the primary goals of the course, with one manager guiding the development of each:

- the integration of the JUnit testing framework
- the addition of a configuration framework for customizing DrJava
- the addition of a conventional debugger, complementing the debugging facilities provided by the interactions pane [1].

### 2.1 Open Source Benefits

Because DrJava is an open source project, we have been able to leverage many existing tools and some existing code during development. Most notably, the SourceForge website provides us with a central location from which to distribute our software and a database for logging tasks and organizing feature requests, bug reports, support requests, and documentation. SourceForge also provides space for the code repository itself and a system for maintaining product newsgroups, among other services. This system has been invaluable to us in organizing and addressing user feedback. Many bugs in DrJava have been discovered and reported by external users via this system, leading to fixes in subsequent builds.

We have also relied upon existing open source development tools, such as Ant and JUnit, to support the XP methodology. We have leveraged the Ant scripting tool to enforce unit testing requirements and to automate the generation of jar files and Javadoc HTML files. The JUnit project has provided a comprehensive framework for implementing our unit tests. The functionality of these tools is on par with that of most (cost-prohibitive) commercial tools.

In addition, we have incorporated portions of other open source projects into DrJava, including the core of the interpreter used in the Interactions pane. This allows us to avoid duplicating work unnecessarily, and significantly increases our development velocity.

## 3 Teaching Open Source Extreme Programming

There are two keys to producing quality code in the context of a classroom XP project. The first is to immerse students in a common software design culture and to stipulate a sensible set of coding standards. Our core programming curriculum emphasizes object-oriented program design using design patterns [4], so that our students have a common design vocabulary and understand a wide variety of different programming abstractions. Given this context, pair programming and peer pressure within a team will prevent poorly designed code from being injected into the code base. Second, all code must be thoroughly unit tested to codify the behavior being implemented. It is impossible to overstate the importance of comprehensive, rigorous unit testing, since it provides the safeguard that allows students to modify the code without breaking it.

Teaching XP in the classroom also poses serious organizational and logistical problems [7]. These include:

- How can we provide effective software management for an entire class of student programmers who only devote a fraction of their "work-week" to our project, given the limited time that the instructor can devote to the course?
- How can students pair program consistently when they have varying schedules and conflicts?
- How can we ensure that computer science students, who are not experienced developers, will produce intelligible, maintainable code?
- How can we provide an onsite customer when the developers do not maintain a common work schedule?

In the following four sections, we examine these problems more closely, along with the solutions we employed.

### 3.1 Project Management in the Classroom

To solve the management problem, we appointed the three most experienced members of the DrJava development team as course teaching assistants and designated them as project managers. The managers met with the instructor to determine which tasks had highest priority and which manager would assume responsibility for each task. Each manager was assigned a small team of two to six students to supervise during the semester. Students were given as much flexibility as possible in choosing which task to work on.

Our decision to use "veteran" DrJava developers as project managers has proven to be quite fortuitous, because it will provide a sustainable flow of managers for subsequent editions of the course. After students graduate from the class, they can work on the project as research assistant programmers during the summer and then serve as project managers the following year, either as paid teaching assistants or for course credit.

Following XP methodology, the managers broke each project into a series of small "stories," and each story was assigned to a pair of programmers. Breaking projects into small stories of specification enabled us to correct misconceptions, ambiguities, and inconsistencies in the high level description of each project. The assigned pair would then break the story up into a series of tasks, each on the order of one to two pair-weeks of work. By allowing the pair to determine the set of tasks themselves, we gave students the opportunity to determine how long the various components of the project would take. Not only did this give students more confidence in the tractability of their projects, it also gave them experience in one of the most difficult skills involved in real-world software engineering: estimating time to completion. As each task was completed, it was reviewed by the project manager assigned to it.

The incremental development of programs using stories and tasks requires a great deal of management to track the actions of developers. The aforementioned SourceForge tools were very well suited to this task.

### 3.2 Pair Programming

Most software projects have a high turnover rate, endangering the ability of the project team to provide ongoing maintenance and development. To address this problem, knowledge about the project's design must be continually transferred from experienced team members to newer members. Pair programming provides the ideal mechanism for such knowledge transfer.

Unfortunately, pair programming presents a major problem in a classroom setting. Students keep radically different schedules and often work in different venues. To alleviate this problem in our course, we used two complementary tactics. First, although the class was scheduled to meet three times per week, the third time (Friday afternoon) was reserved for a common lab session. This schedule guaranteed that all of the students in the class had at least one hour (and typically more since Friday afternoon classes are rare) per week available for pair programming with any other student in the class. Second, students were allowed to form programming pairs of their own choosing, enabling them to find their own best matches. Indeed, some of our self-selected pairs were roommates, allowing for a great deal of pair programming time.

### 3.3 Unit Testing and Continuous Refactoring

In accordance with Extreme Programming guidelines, no code can be committed to our repository unless all unit tests pass. This practice has proven to be even more effective at catching errors than we initially anticipated, partly because of the complexity of the project itself. DrJava is an inherently concurrent program involving multiple threads and two JVMs. As a result, synchronization bugs are a constant concern. Fortunately, unit tests tend to reveal these errors much more frequently than conventional program execution, because they eliminate human reaction time in program transactions, which can mask race conditions.

To ensure ubiquitous use of unit tests, we rely on the Ant scripting tool to enforce policies concerning project builds and commits. All team members are required to use Ant scripts to perform builds and to commit new code. Our Ant scripts will not write a new project version to the repository unless it passes all tests.

To help students become familiar with test-first programming, we gave them a simple standalone practice assignment at the beginning of the course. Most of the grade for the assignment was based on the quality of the unit tests written by the students. We also gave the students a practice assignment in which the students' only task was to write unit tests for a program that we had already written. These warm-up assignments proved very effective; for the duration of the semester, the new code that students wrote for the DrJava project was generally accompanied by suitable tests. As the DrJava code base grew, the overall percentage of code devoted to tests (roughly one-third) remained approximately constant.

Unit testing also allows code to be continually refactored and improved (both to simplify it and to fix bugs) without breaking functionality. Because all DrJava code was covered with unit tests, we were able to give all students in the class permission to refactor any part of the existing code base. Providing students with this degree of autonomy would be disastrous if the code were not controlled so strongly through unit tests. However, with the tests in place, providing students with this power and freedom served to increase enthusiasm for the project. This was quite effective; indeed, several students took it upon themselves to perform a significant refactoring of existing functionality (in the form of vastly improved code indenting) on top of their assigned project! Other students tweaked code in ways to improve behavior on their platform of choice (*e.g.*, Mac OS X, Windows XP, etc.) while maintaining portability.

### 3.4 Providing Onsite Customers

XP requires development teams to use onsite customers to provide immediate feedback on new ideas for functionality. In the context of a software engineering course, the only potential onsite customers are the students, faculty, and staff on campus. Hence, the course project must involve a tool that members of the campus community, ideally the students themselves, can regularly use. DrJava is ideal in this regard since the development team can use it to develop DrJava. For this reason, we mandated that all development work for DrJava should be done in DrJava.

This practice provided students with not just one onsite customer, but many: *all other members of the class.* To capture feedback from the class in their role as customers, we set aside about 10 minutes of each class lecture to discuss how to refine the program specifications, sparking many lively and fruitful debates. This form of customer feedback has helped us to significantly improve DrJava's user interface. Incidentally, such improvements have also enhanced DrJava's effectiveness in meeting its original pedagogic goal: a tool for teaching beginning students.

During the course, we also took input from external customers logged on the SourceForge website very seriously. Because of differences in language (*e.g.*, French, Spanish, English) and computing platforms (*e.g.*, Mac OS X, Linux, Windows 98/NT/2000/XP, Solaris), some bugs and usability issues in DrJava only show up in contexts outside of the Rice campus. Several instructors at other institutions regularly downloaded builds as they were released, used them in their own classes, and provided feature requests and bug reports for future releases. This feedback served as additional customer interaction for students in the course.

Through incremental development, our software team was able to release useful functionality to external customers as soon as it was available. These releases provided students with rapid feedback on the quality of their work. In essence, their homework was immediately downloaded and evaluated by customers around the world. No conventional grading system can provide such powerful feedback.

## 4 Assessment of the Course

In the course evaluations, all students in the course reported that they were excited about what they learned about production programming and what they managed to accomplish. They uniformly rated the course as one of the best that they had taken at Rice.

The course staff was generally pleased with the results of the class, but programmer productivity was slightly below our expectations. By the end of the class, one of the three major projects (JUnit integration) was completed, and the other two projects were nearly completed, leaving a few unfinished stories. But even JUnit integration contained a significant bug that was not revealed by the students' unit tests, which had to be corrected by the project staff after the conclusion of the course. Each of the two unfinished projects required an additional month of work by a pair of programmers before they were fully incorporated into DrJava.

Given that none of the project teams finished a fully functional project by the end of the semester, the question arises: why not? In hindsight, the projects that we chose were slightly too ambitious. Development times are notoriously difficult to estimate, particularly when the developers are students taking at least three other courses, some of which involve programming. Thus, we should have been more conservative in our planning. In addition, we believe that our flexible attitude toward deadlines—an important tenet of XP–contributed to the problem. To encourage students to write high quality code, we avoided fixed deadlines. In one sense, this policy was successful: the quality of code produced by the students was high. On the other hand, student productivity was lower than we expected because students gave priority to other classes with fixed deadlines.

Although we still want to keep students from rushing while coding, we also want to ensure that the class is given an appropriate amount of attention. To address this problem, we plan to modify our approach next spring by requiring students to log the hours that they spend working on the course, and stipulating that we expect each student to spend 8 hours per week outside of class developing software for the class.

We are also investigating new open source tools to enforce test coverage of new code. These tools promise to increase the reliability of changes made by new students.

## 5 Future Plans

We anticipate that DrJava has a long life ahead of it, both as a pedagogic programming environment and as a vehicle for supporting practical research on programming languages, such as advanced dialects of Java supporting first class genericity [2]. For this reason, we anticipate that our production programming course will continue to focus on improving DrJava for the foreseeable future. We have also started a new project to provide additional opportunities for production programming experience. This project is a pedagogic programming environment for the C# language called DrC#. As part of our course next spring, we plan to have one of the development teams focus on DrC# development.

## 6 Conclusions

Given the availability of open source program development tools and the resounding success of XP methodology, there is no reason why colleges and universities cannot teach production programming in the classroom. The robustness and functionality of DrJava demonstrates that students can build production quality software given appropriate instruction on program design and programming methodology, an interesting open source project on which to build, and effective project management by the course staff.

We believe that our production programming course can be replicated at other colleges and universities where resources are available to start a small open source programming project, training the "seed" teaching assistants for the first edition of the course.

## References

[1] E. Allen, R. Cartwright, B. Stoler. *DrJava: A Lightweight Pedagogic Environment for Java SIGCSE 2002*, March 2002.

[2] E. Allen, R. Cartwright, B. Stoler. *Efficient Implementation of Run-time Generic Types for Java.* IFIP WG2.1 Working Conference on Generic Programming, July 2002.

[3] M. Fowler, K. Beck, J. Brant. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass. 1995.

[5] A. Hunt, D. Thomas. *The Pragmatic Programmer.* Addison-Wesley, 2000.

[6] R. Jefferies, A. Anderson, C. Hendrickson. *Extreme Programming Installed.* Addison-Wesley, 2001.

[7] C. Wege, F. Gerhardt. *Learn XP: Host a Bootcamp Extreme Programming Examined.* Addison-Wesley, 2001.