Retina: Helping Students and Instructors Based on Observed Programming Activities

Christian Murphy, Gail Kaiser, Kristin Loveland, Sahar Hasan
Dept. of Computer Science, Columbia University, New York NY 10027
{cmurphy, kaiser, kl2289, sh2503}@cs.columbia.edu

ABSTRACT

It is difficult for instructors of CS1 and CS2 courses to get accurate answers to such critical questions as "how long are students spending on programming assignments?", or "what sorts of errors are they making?" At the same time, students often have no idea of where they stand with respect to the rest of the class in terms of time spent on an assignment or the number or types of errors that they encounter. In this paper, we present a tool called *Retina*, which collects information about students' programming activities, and then provides useful and informative reports to both students and instructors based on the aggregation of that data. Retina can also make real-time recommendations to students, in order to help them quickly address some of the errors they make. In addition to describing Retina and its features, we also present some of our initial findings during two trials of the tool in a real classroom setting.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education—*computer assisted instruction*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Human Factors, Measurement

Keywords

Tutoring Systems, Compilation Errors, CS1, CS2

1. INTRODUCTION

Instructors of CS1 and CS2 courses often must rely on anecdotal evidence, second-hand information, or hearsay to find out answers to such critical questions as "how long are students taking to complete the programming assignments?", or "what sorts of compilation and runtime errors are they making?" Teaching assistants might be able to give some feedback based on what is observed during office hours and tutoring sessions, but often the instructors have no concrete, objective empirical data regarding students' programming activities. If instructors knew precisely what problems their students were having, they could address those in lecture and tailor future assignments accordingly.

At the same time, students in these classes often have no idea of where they stand with respect to the rest of the class in terms of time spent on an assignment or the number or types of errors that they encounter. It is common for an introductory-level student to have great difficulties with initial programming assignments. A student may think "this is probably easy for everyone but me", and not have evidence of the fact that, actually, other students are struggling, too. If students knew for certain that other students are enduring similar experiences, they may not be as inclined to consider computer science as too challenging and give up on it as a field of study that they want to pursue [3].

Lastly, both instructors and students can often suffer from a lack of "organizational memory". Instructors may forget what types of problems students had with assignments in previous semesters; students may not remember how they addressed particular compilation errors, or might not have an accurate way of assessing how long an assignment will take them to complete. Past information about students' programming activities could thus be used to guide both the instructors and the students.

To address these problems, we present a tool called *Retina*, which collects objective observational data about students' programming activities (focusing on compilation and runtime errors), and then provides useful and informative reports based on the aggregation of that data. These reports provide instructors with information such as which students seem to be struggling with an assignment or how long the class is spending on it as a whole; students can see information about their own work, such as the errors they have previously made, and get a glimpse of their peers' activities as well. Retina can also make proactive recommendations and suggestions to the students, for instance how long to expect an assignment to take or what errors to look out for.

In addition to describing Retina and its features, we also present some of our initial findings during two trials of the tool, in which instructors used the collected data to improve their courses, and the data was analyzed to see if any correlations exist that may indicate which aspects of students' programming activities relate to academic success in the course.

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'09, March 3–7, 2009, Chattanooga, Tennessee, USA.
Copyright 2009 ACM 978-1-60558-183-5/09/03 ...$5.00.

2. RELATED WORK

Retina is closely related to the work of Jadud [9] and to the tool ClockIt [17], both of which monitor and log student programming activities to determine such information as when a student starts working on an assignment, how much time is spent, and what errors are made during that time. These tools are focused only on BlueJ, however, whereas Retina works with a variety of compiler environments: since BlueJ only reports one compilation error at a time, this means those tools cannot capture some data captured by Retina, such as errors per compilation attempt. More importanly, Retina adds additional functionality, particularly the recommendation and suggestion features that allow the student not only to see her past activities, but also get an idea of what to expect in the future.

Similarly, while Hackystat [12] addresses the problems of transparent data collection of programming activities, and Gauntlet [5] and Expresso [8] are just two of the many tools that seek to provide help about compiler errors, none of these provide any out-of-the-box functionality with respect to the use of such data for providing feedback and analysis to the instructor and students.

Marmoset [19] and Web-CAT [4] are two important tools that observe student programming behavior and automate unit and system testing, but these tools are focused on logical errors ("does the program produce the correct output?") and not on syntax/semantic errors ("does the program compile?") or runtime errors ("does the program run to completion?"). Thus, Retina is complementary to these tools, since it focuses on a different aspect of programming activities. Additionally, neither Marmoset nor Web-CAT provides proactive recommendations to students, as is done in Retina.

The recommendation features of Retina are related to many other important works that make suggestions based on past programming activities, such as [7]. Previous work in mining student source code repositories has looked at whether programming activities can predict grade performance [13] [15], but they only consider version control activities and coding style, and not compilations and errors as we do with Retina. Furthermore, none of these address the particular needs of novice-level programmers and their instructors.

3. RETINA FEATURES

Based on the data that is collected about students' programming activities, Retina provides numerous features for both the instructors and the students.

3.1 Data Collection

Retina records students' compilation attempts and compiler errors so that they can be stored in a central database and later mined and analyzed. Currently, Retina supports the command-line compiler through a modification to javac, and we have also implemented plugins for both Eclipse and BlueJ. When the student invokes the compiler, the student's name (or ID) and the current date and time are recorded in a local XML file. Additionally, any compilation errors are reported to the student as normal, but for each error, the type of error, the file name and line number, and the associated error message are all recorded as well. The data is then sent to a central server, where it is stored in a relational database. The entire process is completely transparent to the student, and the performance impact is negligible.

To address privacy issues, Retina allows the student to opt out of the collection of data entirely. Alternatively, the student can specify that information be collected anonymously so that it contributes only to the overall data for the class, and cannot be tied back to the individual student.

When using the command-line Java VM for executing the program, Retina can also record students' runtime errors, *i.e.* uncaught exceptions. This is done via integration with Backstop [16], which intercepts any uncaught exceptions and provides more user-friendly error messages. We modified Backstop so that the student's name, the current date and time, the type of exception, the corresponding error message, and the stack trace are all recorded to an XML file and then sent to the central server.

3.2 Retina Instructor View

Once the data has been collected, instructors can access the collected data through the Retina Instructor View, which is implemented as a standalone Java application that runs locally on the instructor's system; the instructor can also access the data remotely via a web browser interface. The application has two modes: "Browse" and "Class".

The "Browse" mode, shown in Figure 1, allows the instructor to get an understanding of an individual student's efforts on a particular assignment. The instructor can select a student and an assignment for either the current semester or a past one, and then see a list of all of the student's compilation and runtime errors, as well as aggregate data about the total number of compilations, the total number of compilation errors, and the most common compilation error.

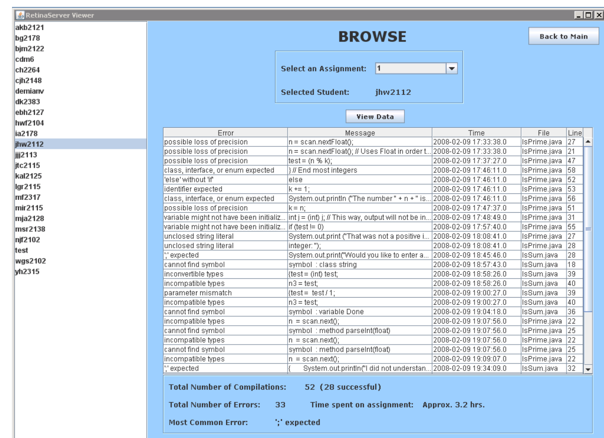

Figure 1: Retina Instructor View in Browse Mode

The instructor can also see an approximation of how long the student has spent on the assignment. Although it is impossible to know exactly how long the student is working on the assignment, especially since we only capture discrete compilation events and we do not know when the student has started or finished the programming session, we can approximate the time by looking at each compilation event and assuming that compilation events within a fixed time (say, 30 minutes) of each other are part of the same session. We then use the sum total of the individual programming sessions as the approximate time spent on the assignment. Others have also addressed this particular problem of understanding developers' time spent on programming activities [2], and tools such as the Eclipse Watcher [14] have been

developed to collect and record this information; we are investigating these for future integration with Retina.

In the "Class" mode, the instructor selects an assignment and sees an overview of how the class has performed as a whole. This mode allows the instructor to see the ten most common compilation and runtime errors, and the number of occurrences, so that the instructor can address the most common ones as necessary. It also shows all students' time spent on the assignment, in descending order, and the average time spent for all students in the class. This lets the instructor gauge the difficulty of a particular assignment and to know how long students are spending on it as a class, but also to see an overview of how each student is performing.

In this mode, the instructor can also see graphs that visually display when compilation events and errors occurred. The instructor can then get a feel for when (relative to the assignment being due) students started working on the assignment, and what time of day most students are working.

3.3 Retina Student View

Students can access their own Retina information via JSP pages on a web server (so that the student need not install any special software, and the instructor can modify the web pages easily, if desired). A student logs into the Retina Student View and is able to see information about her own activities, and how she relates to the rest of the class.

A student can select an assignment (the default is the current assignment) and see what compilation and runtime errors she has made, as well as how much time has been spent. It is our belief that, by seeing previous errors that have since been fixed, the student may recall how she fixed them, and then be able to use that knowledge to fix future errors. This is one way in which Retina enables "personal organizational memory".

In order to help the student answer the question "how am I doing with respect to the rest of the class?", Retina also reveals the average number of compilations (both the total number and the number of successful ones, as well as percentages), average number of compilation errors, and average time spent for the entire class on the selected assignment, as shown in Figure 2. Retina also shows the distribution of values graphically, to give a visual clue as to where the student stands, and indicates the most common errors made in the class, in order to let the student see whether other students are making the same mistakes.

A unique feature of Retina is that it also provides the student with suggestions based on what has been observed about that particular student, her classmates, and students who took the class in previous semesters. One of Retina's suggestions is the amount of time that the student can expect to spend on an upcoming assignment. This is done by considering the student's past performance on previous assignments with respect to the class average (*i.e.* her average ranking within the class), and then finding the time that it took similarly-ranked students to complete the assignment in previous semesters. This assumes that the assignment is equally difficult across semesters, and that the student will perform as she did in the past, but gives a good first estimate in order to set the student's expectations. Through our own teaching experiences, we have observed that most novice programmers have little ability to accurately predict how long an assignment will take to complete, especially as new topics are introduced and the assignments become

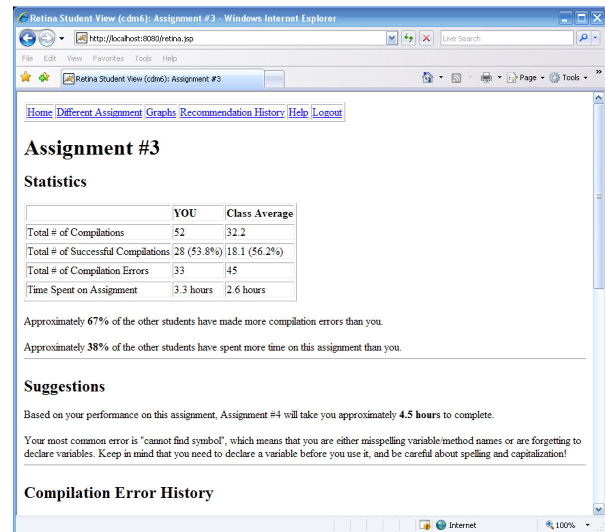

Figure 2: Retina Student View

more complex, so this information can be crucial to helping a student know when to start the assignment so as to avoid any last-minute rush caused by underestimating the assignment's difficulty.

Another suggestion made by Retina involves the types of compiler errors that it feels the student is likely to make. This is achieved by noting any errors that the student has frequently made on previous assignments, especially those that fall outside the list of most common errors across all students in the class. Retina lists some of these errors, but also makes suggestions as to how to avoid them in the future. For instance, if the student's most common error is "cannot find symbol", Retina will suggest *"you are either misspelling variable/method names or are forgetting to declare variables. Keep in mind that you need to declare a variable before you use it, and be careful about spelling and capitalization!"*

3.4 Retina Recommendation Tool

Another important feature of Retina is the ability to produce immediate, real-time recommendations that can proactively be sent to students based on their observed programming activities. These recommendations are sent to the students as they are programming and as their event logs are being recorded on the server. Retina uses Instant Messaging (IM) applications as the user interface for its recommendations, which has been shown to be an effective technique in such situations [1] [18].

Using the JClaim [11] API, we have developed an IM server bot that works with the Yahoo! Messenger, Windows Live Messenger, and Google Talk chat networks. A student initiates a chat session with Retina, and identifies herself via a username (the same username that is associated with that student's collected data). When Retina determines that a recommendation is in order, a message is sent to the student so that she can get a better understanding of what to do next. Of course, a feature like this must be wary of the possibility of a "Clippy Effect" [6], in that unwanted or unwarranted messages may prove to be more annoying than helpful, but the student has the option of terminating the chat session and thus disabling this feature. In the future, we may also consider a user-configurable verbosity level to

give the student even more control.

The current implementation of the Retina Recommendation Tool is rule-based: when event logs are received, Retina checks whether any of the rules have been met, and sends a message accordingly. Retina currently checks for the following situations:

1. High rate of errors per compilation. If the rate of errors per compilation is higher than normal for that student (*e.g.* twice the normal rate), Retina will recommend that the student attempt to work in smaller intervals or address compilation errors that are at the top of the list, which may be causing other errors to appear.

2. Spending too long on the assignment. If the amount of time the student has spent on the assignment is more than twice the suggested time for that student (as described above), Retina will recommend that the student is spending too much time on it and should seek the assistance of a member of the teaching staff for help.

3. Same error made multiple times. If the same error occurs on the same line on more than four consecutive compilation attempts, Retina will explain that error in simple terms and recommend a possible way to fix it, using the same suggestions as described above.

Note that all of the values described above can be configured depending on the instructor's preferences.

Retina also allows for on-demand recommendations. A student may send an IM message of "recommend" and will receive a listing of all recommendations previously made for the current assignment. A student can also ask Retina to explain a compilation or runtime error by using the keyword "explain" and then the name or type of the error. The error explanations are similar to those found at [10].

Students can also use the keyword "who" to see how many students are currently working on the assignment, based on recent activity (for privacy reasons, Retina does not currently display the other users' names, just the total number). This allows the students to get a feeling that she is not alone in working on the assignment.

Retina does not support fully free-text input commands, but the complete listing of all valid commands is available by typing "help". Because of the use of a text-based interface, though, Retina performs some processing on the typed input, in case of any typing mistakes. The domain of valid input commands is limited so it is possible to accurately guess the user's intention even if the commands are not spelled correctly; we currently use custom-built code that employs such techniques as comparing edit distance and considering permutations. Thus, if a user types "expln", Retina can still detect that the user meant to type "explain".

4. EVALUATION

In the Spring 2008 and Summer 2008 semesters, students in our university's CS1 course volunteered to allow Retina to collect data about their compilation errors. In total, 21 students volunteered in the spring semester, and 27 did so in the summer. Only the Retina Instructor View was complete during the time of our trial, however.

4.1 Instructor Experience

The course instructors were able to use the data collected by Retina to improve their interaction with the students. Said one, *"Retina was useful in the case where a student was asking for one-on-one help, so that I could know in advance what difficulties that student had, could anticipate the questions the student would ask, and could tailor the help appropriately."* Said the other, *"Retina allowed me to gauge when granting an extension on an assignment was justified by seeing how long a student had been working on it, and when she started."*

As expected, Retina also helped increase the quality of the lectures and of the course in general. One instructor said, *"I was surprised to see the students encountering runtime errors [uncaught exceptions] on earlier programming assignments. I really didn't anticipate that, but by knowing that, I could then talk about it earlier in the course."* The instructor went on to say, *"By seeing what errors the students were making as a whole, I could also warn the TAs what to look out for, and discuss with them good ways to help the students address those problems. Retina really helped us improve the course and the way we worked with the students."*

4.2 Data Analysis

After the trials were complete, we also investigated whether any of the data we collected correlate to academic performance, in particular the grades received on individual assignments. Our hope was that any interesting findings might prove or disprove assumptions about students' programming habits, and also could guide some of the suggestions that would be made by the Retina Recommendation Tool. Although we had a very small sample size, we did notice some interesting trends. For instance, for a given assignment, we did not find any correlation between time spent on the assignment and the grades, but when we considered the performance over the entire the semester, we noticed that students who spent less time on the assignments tended to do better than students who spent more time. Intuitively this makes sense, as students who "get it" finish the assignments more quickly than those who require more time to work through mistakes. We also noticed that, when considering semester-long totals, students who made fewer compilation errors also tended to receive higher scores, which indicates that students who make many errors may also be struggling. Therefore, it is reasonable to assume that students who spend an inordinate amount of time on the assignments or get a very high number of errors need extra attention from the teaching staff, and Retina can be used to quickly identify these students so that timely assistance can be given.

We also found that although most of the students were working in the late afternoon and early evening, the majority of errors were made between the hours of 8pm and 5am, indicating that students were making more mistakes late at night. We also found the highest rates of errors per compilation occurred between 1am and 4am, with these values being substantially higher than all other values for errors per compilation per hour. We are able to see the trend that while the majority of students work during the daytime hours, those who work later at night and into the early morning will tend to make more frequent mistakes. These results inspired us to create the recommendation that advises students to prepare better for the assignment and not wait until right before class to finish it, and also to focus their working times during the daytime and early evening.

5. FUTURE WORK AND CONCLUSION

Further studies are required to demonstrate the effectiveness of the tool and its measurable effects on students' abil-

ities to improve as programmers. As pointed out in [19], it may be unethical to conduct an empirical study in which some students have access to the tool and others are prohibited from using it, but we intend to investigate whether the students make use of Retina's suggestions, and their effect on the students' programming activities.

Additionally, there are clearly privacy concerns when it comes to transparently collecting information about students and sharing it with others. However, we found anecdotally that the students who participated in the trials were not concerned about privacy, and in general would find this acceptable as long as they could see what data is collected, especially if they could then gain benefits from doing so.

In this paper, we have presented Retina, a tool for gathering data about students' programming activities, and then using that data to allow instructors to understand more about what their students are doing, and to allow students to review their past actions, see how they relate to other students, and get suggestions and recommendations about how to go forward. We hope that this work will enable other educators - and their students - to understand and learn from students' programming activities.

6. ACKNOWLEDGMENTS

We are proud to say that much of Retina was designed and developed by undergraduate students, whose insight and effort made this work possible: Diana Chang, Michelle Forman, Tian He, Shreya Kedia, Henry Lau and Benjamin Monnin (the third and fourth authors of this paper are also undergraduates); a high school student, Jao-ke Chin-Lee, contributed to this project as well. We would also like to thank Aaron Fernandes (a Masters student in our department), Adam Cannon (the instructor for the CS1 course), and all the students who participated in the user studies and observations. Murphy and Kaiser are members of the Programming Systems Lab, funded in part by NSF CNS-0717544, CNS-0627473, CNS-0426623 and EIA-0202063, and NIH 1 U54 CA121852-01A1.

7. REFERENCES

[1] S. Chan, B. Hill, and S. Yardi. Instant messaging bots: accountability and peripheral participation for textual user interfaces. In *Proc of the 2005 international ACM SIGGROUP conference on supporting group work*, pages 113–115, 2005.

[2] I. Coman and A. Sillitti. An empirical exploratory study on inferring developers' activities from low-level data. In *Proc of the nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2007.

[3] D. Cubranic and M. A. D. Storey. Collaboration support for novice team programming. In *Proc of the 2005 international ACM SIGGROUP conference on supporting group work*, pages 136–139, 2005.

[4] S. H. Edwards and M. A. Perez-Quinones. Web-CAT: automatically grading programming assignments. In *Proc of the 13th annual conference on Innovation and technology in computer science education (ITiCSE)*, pages 328–328, 2008.

[5] T. Flowers, C. Carver, and J. Jackson. Empowering students and building confidence in novice programmers through Gauntlet. In *34th ASEE/IEEE Frontiers in Education Conference*, pages T3H–10 – T3H–13, Oct 2004.

[6] R. G. P. Galluccio. Humanizing CALL: The use of pedagogical agents as language tutors. New England Regional Association for Language Learning Technology, Oct. 2006.

[7] R. Holmes, R. J. Walker, and G. C. Murphy. Strathcona example recommendation tool. In *Proc of the 10th European software engineering conference*, pages 237–240, 2005.

[8] M. Hristova, A. Misra, M. Rutter, and R. Mercuri. Identifying and correcting Java programming errors for introductory computer science students. In *Proc of the 34th SIGCSE technical symposium on computer science education*, pages 153–156, Feb 2003.

[9] M. Jadud. A first look at novice compilation behaviour using BlueJ. *Computer Science Education*, 15(1):25–40, March 2005.

[10] Java Glossary. Compile time error messages. http://mindprod.com/jgloss/compileerrormessages.html.

[11] JClaim. Java compliant logging and auditing instant messenger. http://www.itbsllc.com/jclaim/.

[12] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, and W. E. J. Doane. Beyond the personal software process: metrics collection and analysis for the differently disciplined. In *Proc of the 25th International Conference on Software Engineering (ICSE)*, pages 641–646, 2003.

[13] Y. Liu, E. Stroulia, K. Wong, and D. German. Using CVS historical information to understand how students develop software. In *Proc of the 2004 international workshop on mining software repositories*, 2004.

[14] J. McKeogh and D. C. Exton. Eclipse plug-in to monitor the programmer behaviour. In *OOPSLA Eclipse Technology eXchange Workshop*, 2004.

[15] K. Mierle, K. Laven, S. Roweis, and G. Wilson. Mining student CVS repositories for performance indicators. In *Proc of the 2005 international workshop on mining software repositories*, pages 1–5, 2005.

[16] C. Murphy, E. Kim, G. Kaiser, and A. Cannon. Backstop: A tool for debugging runtime errors. In *Proc of the 39th SIGCSE technical symposium on computer science education*, pages 173–177, 2008.

[17] C. Norris, F. Barry, J. B. Fenwick Jr, K. Reid, and J. Rountree. ClockIt: Collecting quantitative data on how beginning software developers really work. In *Proc of the 13th conference on innovation and technology in computer science education (ITiCSE)*, 2008.

[18] A. Ribak, M. Jacovi, and V. Soroka. "Ask before you search": peer support and community building with reachout. In *Proc of the 2002 ACM conference on computer supported cooperative work (CSCW)*, pages 126–135, 2002.

[19] J. Spacco et al. Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. In *Proc of the 11th annual conference on Innovation and technology in computer science education (ITiCSE)*, pages 13–17, 2006.