Toward Continuous Assessment of the Programming Process

Ayaan M. Kazerouni
Virginia Tech
Blacksburg, VA
ayaan@vt.edu

ABSTRACT

Assessment of software tends to focus on postmortem evaluation of metrics like correctness, mergeability, and code coverage. This is evidenced in the current practices of continuous integration and deployment that focus on software's ability to pass unit tests before it can be merged into a deployment pipeline. However, little attention or tooling is given to the assessment of the software development process itself. Good process becomes both more challenging and more critical as software complexity increases. Real-time evaluation and feedback about a software developer's skills, such as incremental development, testing, and time management, could greatly increase productivity and improve the ability to write *tested* and *correct* code. My work focuses on the collection and analysis of fine-grained programming process data to help quantitatively model the programming process in terms of these metrics. I report on my research problem, presenting past work involving the collection and analysis of IDE event data from junior level students working on large and complex projects. The goal is to quantify the programming process in terms of incremental development and procrastination. I also present a long-term vision for my research and present work planned in the short term as a step toward that vision.

ACM Reference Format:
Ayaan M. Kazerouni. 2018. Toward Continuous Assessment of the Programming Process. In *SIGCSE '18: SIGCSE '18: The 49th ACM Technical Symposium on Computing Science Education, February 21–24, 2018, Baltimore , MD, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3159450.3162337

1 MOTIVATION

In my research, I develop models to quantify a student's programming process. By measuring the programming process, I can empirically evaluate its adherence to known best practices in software engineering. Once I can characterize the observed process, I hope to build tools that provide developers with intelligent and timely feedback when they are in danger of straying from those practices. In the long term, I hope to contribute to the standardization and adoption of continuous software assessment techniques that evaluate not only the final product, but also the process undertaken to produce it.

Mid-level Computer Science courses often involve major programming projects, with lifecycles measured in weeks. Assessment of such projects tends to focus on postmortem evaluation of aspects like correctness, code style, and code coverage, as measured by tools such as Web-CAT [5], or available in Eclipse. However, there is little attention or tooling spared for the assessment of the software development process itself. This assessment becomes both more challenging and more crucial to address as software complexity increases. None of the aspects mentioned above directly address a major concern that too many students are unable to complete programming projects at this scale. This often is a result of inadequate skill by the student in good project management techniques, including time management [6] and fundamental development processes such as incremental development and proper use of testing [2, 14].

Previous work [6] showed that when students received A/B scores on projects, they started earlier and finished earlier than when those same students received C/D/F scores. Subsequent work [13] showed that course-grained adaptive e-mail interventions were associated with significantly reduced rates of late program submission and significantly increased rates of early program submissions.

These studies provided encouraging evidence that: 1) Procrastination has a strong correlation with lower project scores, and 2) Regular, automatic, and adaptive feedback helps reduce procrastination, and might help change other types of student programming behavior.

2 RELATED WORK

The Web-Center for Automated Testing (Web-CAT) [5] is a web-based automated grading system that allows students to make multiple submissions to an assignment and receive immediate feedback. Feedback can be about correctness, code style, or code coverage by student-written tests. While Web-CAT's model of multiple submissions affords us the ability to get a rough idea of the project's trajectory over time, submission level data is considered the least granular form of student-data [8]. To assess incremental development and testing, we would need to collect data *during development*, rather than at submission.

To this end, we developed DevEventTracker [11], an Eclipse plugin that collects event-level data. As described in [8], DevEventTracker collects data for: *executions, compilations, file saves, line-level edits*. It also captures periodic Git snapshots, providing a rich representation of a project's evolution over time. Coupled with submission data from Web-CAT (and the associated results from instructors' unit tests), this provides for more robust analysis of a student's programming process.

The Test My Code (TMC) plugin [15] for NetBeans records events whenever the student saves, runs, or tests code using tests provided by the instructor. Hosseini, et al [7] make use of this plugin in an attempt to achieve goals similar to ours, but with a few differences in the type of data collected. For example, in terms of detecting student testing, the TMC plugin collects data on runs of pre-written

SIGCSE '18, February 21–24, 2018, Baltimore , MD, USA

ACM ISBN 978-1-4503-5103-4/18/02.
https://doi.org/10.1145/3159450.3162337

tests provided by the instructor, while DevEventTracker collects data about students writing and running their own tests, providing information about the student's **autonomous software development habits**, which is ultimately what we wish to assess.

Hackystat [10] is an open-source project from the University of Hawaii. It consists of sensors integrated into the user's development environment of choice that trigger when certain actions are taken. DevEventTracker builds upon Hackystat, using its client-side protocols and preexisting sensors in conjunction with our own to send data to the server. In terms of added functionality, DevEventTracker tracks program executions and Git snapshots, and ties the collected event data to Web-CAT assignment submissions, allowing analysis of programming behavior and project outcomes.

Carter, et al. [3, 4] use the Normalized Programming State Model to represent the programming process as a series of state transitions. They perform predictive analysis for project scores based on the time spent in each state. Other student data-tracking research includes [1, 9, 16].

3 APPROACH AND UNIQUENESS

Continuously collecting DevEvent data as students program gives us a unique insight into the development process of the typical student. The major benefit from this augmentation is twofold: 1) Our data are no longer limited by when a student decides to make a submission. 2) Since we are collecting data from the IDE itself, we have access to events that were not available through Web-CAT alone. With the data in hand, the next step is to use it to empirically determine if a student is practicing incremental development.

We collected data from three sections of a post-CS2, junior-level Data Structures and Algorithms course at Virginia Tech. After filtering, the dataset consists of the work of 162 students on 545 programming projects turned in across four assignments (approximately 6.3M events). Using this data, we developed the following measures to assess students' time management skills and software testing habits [11]: **Early/Often Index –** how often a student works on the project, weighted by the number of days until the project deadline, **Program/Test Executions –** how much code a student writes before executing the project to test it. We have calculated this index in different combinations of solution/test code coupled with regular/test executions, and **Incremental Test Writing –** a measure of how much time passes, on average, between when students write solution code and test code.

Assessing incremental development is a non-trivial problem. A primary concern is that there is no readily available 'ground truth' against which we can test our calculated measures. There very well might be a correlation between project grades and incremental development, but there could be a group of students for whom these projects are not so challenging, and so they can get good grades even with poor incremental development practices. To address this, we validate our models against a number of project outcomes like correctness, time spent on the project, and whether or not the project was submitted on time.

We performed qualitative evaluation of the metrics in [11] by interviewing students to assess the metrics' accuracy against students' self-perceived notions of their own programming process on individual projects. In addition to this, we conducted manual inspection of Git repositories. Overall, we found the metrics to be *mostly accurate*, with some room for improvement. Following this, we conducted quantitative evaluation of the four metrics to assess their relationships with the outcomes described above [12]. By examining the original four metrics, we can develop and evaluate refinements with the aim of better characterizing intuitive notions of good software development practices. Our quantification of procrastination was significantly related to project correctness and finishing solutions on time. The median test edit time as captured by DevEventTracker was significantly related to total time spent on the project. A preliminary predictive model using these two metrics showed 69% accuracy at classifying a project as 'solved' or not (where a 'solved' project scored greater than 95% correct on instructors' unit tests).

The metrics described and evaluated above were calculated in a way that could only provide useful information *after* a project has been completed. This can be useful feedback that helps the student follow better practices on the next project, but it would be better to give feedback *as the student programs*, so they may act on it before too much harm has been done. This ideal of *continuous assessment and self-correction* represents the long-term goal of my research.

We have made progress in quantifying the quality of a student's programming process, but there is more to be done. I am currently pursuing software repository mining and static analysis techniques to assess the testing practices undertaken by students while arriving at a software solution. My hypothesis is that contextually richer data like Git snapshots will help to build a more informative model of the programming process, as compared to models built using IDE-events alone. Static analysis techniques enriched by Git histories give us the ability to assess the amount of work done between writing solution code and the testing of *that same solution code*. A model such as this one, not dependent on project-wide means and medians, shows potential for administration during a project lifecycle as opposed to after it. In addition to this, I plan to dynamically analyze a project's test cases to assess their logical effectiveness over time.

4 CONTRIBUTION

The most obvious contribution is to students taking programming courses. Regular interventions during project completion will keep students on track to complete assignments and follow best practices during project development. The students under study are junior-level students working on large and complex programming assignments. They are typically only two or three semesters removed from professional developers entering the workforce. The metrics developed in this research could easily be applied in an industrial setting. A primary vision of this research is to deploy an end-to-end pipeline that receives an incoming event stream and responds with timely and effective feedback to the developer.

REFERENCES

[1] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 223–228. https://doi.org/10.1145/2538862.2538924

[2] Kevin Buffardi and Stephen H. Edwards. 2014. A Formative Study of Influences on Student Testing Behaviors. In *Proceedings of the 45th ACM Technical Symposium*

on Computer Science Education (SIGCSE '14). ACM, New York, NY, USA, 597–602. https://doi.org/10.1145/2538862.2538982

[3] Adam Scott Carter and Christopher David Hundhausen. 2017. Using Programming Process Data to Detect Differences in Students' Patterns of Programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 105–110. https://doi.org/10.1145/3017680.3017785

[4] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2015. The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 141–150. https://doi.org/10.1145/2787622.2787710

[5] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)*. ACM, New York, NY, USA, 328–328. https://doi.org/10.1145/1384271.1384371

[6] Stephen H. Edwards, Jason Snyder, Manuel A. Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing Effective and Ineffective Behaviors of Student Programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*. ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/1584322.1584325

[7] Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Exploring Problem Solving Paths in a Java Programming Course. In *Psychology of Programming Interest Group Conference, PPIG 2014*. 65–76. http://d-scholarship.pitt.edu/21832/

[8] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15)*. ACM, New York, NY, USA, 41–63. https://doi.org/10.1145/2858796.2858798

[9] Matthew C. Jadud. 2006. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research (ICER '06)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/1151588.1151600

[10] Philip M Johnson, Hongbing Kou, Joy M Agustin, Qin Zhang, Aaron Kagawa, and Takuya Yamashita. 2004. Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*. IEEE, 136–144.

[11] Ayaan M. Kazerouni, Stephen H. Edwards, T. Simin Hall, and Clifford A. Shaffer. 2017. DevEventTracker: Tracking Development Events to Assess Incremental Development and Procrastination. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, New York, NY, USA, 104–109. https://doi.org/10.1145/3059009.3059050

[12] Ayaan M. Kazerouni, Stephen H. Edwards, and Clifford A. Shaffer. 2017. Quantifying Incremental Development Practices and Their Relationship to Procrastination. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 191–199. https://doi.org/10.1145/3105726.3106180

[13] Joshua Martin, Stephen H. Edwards, and Clfford A. Shaffer. 2015. The Effects of Procrastination Interventions on Programming Project Success. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 3–11. https://doi.org/10.1145/2787622.2787730

[14] Jaime Spacco and William Pugh. 2006. Helping Students Appreciate Test-driven Development (TDD). In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06)*. ACM, New York, NY, USA, 907–913. https://doi.org/10.1145/1176617.1176743

[15] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding Students' Learning Using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. ACM, New York, NY, USA, 117–122. https://doi.org/10.1145/2462476.2462501

[16] Christopher Watson, Frederick W. B. Li, and Jamie L. Godwin. 2013. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies (ICALT '13)*. IEEE Computer Society, Washington, DC, USA, 319–323. https://doi.org/10.1109/ICALT.2013.99