Toward Continuous Assessment of the Programming Process

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

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 student's software development skills, such as incremental development, testing, and time management, could greatly increase productivity and improve the ability to write *tested* and *correct* code. In my research, I develop models to quantify a student's programming process in terms of these metrics. By measuring the programming process, I can empirically evaluate its adherence to known best practices in software engineering. With the ability to characterize this, I can build tools to provide them 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 include not only the final product, but also the process undertaken to produce it.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering**;

KEYWORDS

incremental development, procrastination, software testing

ACM Reference Format:
Ayaan M. Kazerouni. 2019. Toward Continuous Assessment of the Programming Process. In *International Computing Education Research Conference (ICER '19), August 12–14, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3291279.3339429

1 BACKGROUND AND MOTIVATION

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 [4]. However, there is little attention or tooling for the assessing the software development process itself. This assessment becomes both more challenging and more crucial to address as software complexity increases. Many students display inadequate skill in time management [5] and fundamental development processes such as software testing [2, 15], and teaching these skills is not formally included in the typical undergraduate CS curriculum [8].

Beller proposed the concept of Feedback-Driven Development [1], in which software developers are given *descriptive* feedback on their workflows, as opposed to *prescriptive* recommendations to follow certain methodologies. The Normalized Programming State Model (NPSM) [3] models the programming process of CS 2 students as a series of state transitions between code editing, debugging, error states, etc. Time spent in these states were shown to be effectve predictors of assignment and course performance. Edwards et al. found evidence that: 1) Procrastination has a strong correlation with lower project scores [6], and 2) Regular, automatic, and adaptive feedback helps reduce late submissions [14]. Finally, Loksa et al. found evidence linking explicit guidance on metacognitive aspects of programming with productivity [13] and self-regulation with improved programming success [12]. These studies provide encouraging evidence (in varied contexts) suggesting that we might be able to change certain programming behaviors with the help of targeted interventions.

Unlike the works described above, I focus on more advanced students working on larger projects, and model higher-order behaviors such as procrastination, software testing, and debugging. In the next section, I describe my research questions, the data collected to facilitate answering them, and my overall dissertation progress.

2 RESEARCH METHOD

I propose the following thesis: *Continuous feedback on the programming process will help improve student programming behaviors, and improved behaviors will lead to improved project outcomes.*

For the motivating studies in [6] and [14], data was collected in the form of Web-CAT submissions. While Web-CAT's model of multiple submissions gives us a rough idea of the project's trajectory over time, submission level data is considered the least granular form of student activity data [7]. To assess incremental development and testing, and to address the research questions posed below, we would need to collect data *during development*, rather than at submission time.

To this end, we built DevEventTracker [10], an Eclipse plugin that collects event-level data for *executions, compilations, file saves, line-level edits*, to use terminology from [7]. It also captures periodic Git snapshots. Coupled with Web-CAT submission data, this provides for more robust analysis of a student's programming process. We use these data to address the research questions described below.

ICER '19, August 12–14, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6185-9/19/08.
https://doi.org/10.1145/3291279.3339429

RQ 1. To what extent do students procrastinate on programming projects? How does this relate with project outcomes?

We developed a measure of central tendency of *when code was produced* in terms of days until the project deadline, and the *volume of code produced*. We calculated this metric using DevEventTracker for four large programming assignments [9], and found that: 1) Most students tended to do work < 10 days before the deadline, even though they were given 3 to 4 weeks to work on projects; 2) Students who tended to work earlier tended to produce more semantically correct implementations, and had earlier final submission times; and 3) There was no difference in total time spent on projects between early and late workers.

These findings served as confirmation of intuitive expectations of the effects of procrastination on project outcomes. For further evaluation, we interviewed students about their programming habits and compared their responses with our quantitative measurements of their behaviors [10].

RQ 2. How much do students engage with testing throughout the project lifecycle? How does this relate with project outcomes?

Using static analysis of Git snapshots captured by DevEventTracker, we measured students' levels of engagement with test writing on the entire project, in individual work sessions, and for individual methods in their projects [11]. We found that 1) Students do not tend to spread out their testing effort over work sessions; 2) Students who write more tests each time they work on their projects tend to produce more semantically correct programs and stronger test suites (in terms of object branch coverage); and 3) Writing tests first or last was not associated with better or worse project outcomes.

RQ 3. How does regular and adaptive feedback on programming behaviors like time management and testing impact behaviors and project outcomes?

With these quantified behaviors and their empirically determined relationships with project outcomes in hand, I plan to design and deploy interventions for students who are in danger from straying from best practices. This can be done 'on demand' in the form of a learning dashboard, or with periodic or intelligently timed notifications through email. Further investigation is required to determine what kind of impact these interventions might have. I am encouraged by previous work ([14]) that was able to show a causal relationship between adaptive emails and reduced rates of late submissions.

I am currently working on this research question, and I plan to have interventions in place during the Summer and Fall offerings of our Data Structures & Algorithms course.

3 CONTRIBUTION

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. 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, and against student testimonials. Regular interventions based on our validated models of the programming process might help keep students on track to complete assignments and follow best practices. The students we study are working on large and complex assignments, and are typically only two or three semesters removed from professionals entering the workforce. This work 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] Moritz Beller. 2018. Toward an Empirical Theory of Feedback-Driven Development. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*. 503–505. https://doi.org/10.1145/3183440.3190332

[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*. 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*. 105–110. https://doi.org/10.1145/3017680.3017785

[4] Stephen H Edwards. 2003. Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing* 3, 3 (2003), 1–24. https://doi.org/10.1145/1029994.1029995

[5] Stephen H Edwards and Manuel Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *ITiCSE 08 Proceedings of the 13th annual conference on Innovation and technology in computer science education*, Vol. 3. 60558–60558. https://doi.org/10.1145/1597849.1384371

[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*. 3–14. https://doi.org/10.1145/1584322.1584325

[7] Petri Ihantola, Matthew Butler, Stephen H Edwards, Virginia Tech, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, 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*. 41–63. https://doi.org/10.1145/2858796.2858798

[8] Edward L. Jones and Edward L. 2000. Software testing in the computer science curriculum – a holistic approach. In *Proceedings of the Australasian conference on Computing education - ACSE '00*. 153–157. https://doi.org/10.1145/359369.359392

[9] 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*. 104–109. https://doi.org/10.1145/3059009.3059050

[10] Ayaan M Kazerouni, Stephen H Edwards, and Clifford A Shaffer. 2017. Quantifying Incremental Development Practices and Their Relationship to Procrastination. In *International Computing Education Research Conference (ICER) 2017*. 191–199. https://doi.org/10.1145/3105726.3106180

[11] Ayaan M. Kazerouni, Clifford A. Shaffer, Stephen H. Edwards, and Francisco Servant. 2019. Assessing Incremental Testing Practices and Their Impact on Project Outcomes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 407–413. https://doi.org/10.1145/3287324.3287366

[12] Dastyni Loksa and Andrew Jensen Ko. 2016. The Role of Self-Regulation in Programming Problem Solving Process and Success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research, ICER 2016, Melbourne, VIC, Australia, September 8-12, 2016*. 83–91. https://doi.org/10.1145/2960310.2960334

[13] Dastyni Loksa, Andrew Jensen Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*. 1449–1461. https://doi.org/10.1145/2858036.2858252

[14] Joshua Martin, Stephen H Edwards, and Clifford 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

[15] Jaime Spacco and William Pugh. 2006. Helping students appreciate test-driven development (TDD). In *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications - OOPSLA '06*. 907. https://doi.org/10.1145/1176617.1176743