Lightning Talk: Curating Analyses for Programming Log Data

Thomas W. Price
twprice@ncsu.edu
North Carolina State University
Raleigh, NC

Ge Gao
ggao5@ncsu.edu
North Carolina State University
Raleigh, NC

ABSTRACT

In this lightning talk, we will solicit input from the SPLICE community on an effort to collect and curate analyses for programming log data. The talk will explore what a repository of CS educational data mining (CSEDM) tools might look like. It will focus on the challenge presented by the diversity of programming log data, which can vary in granularity, which events are recorded, and programming language features. We will present an initial effort to organize analyzes according to the attributes of data that they require.

1 INTRODUCTION

The goal of this lightning talk is to seek input from the SPLICE community around two questions: 1) What are the most useful things that researchers have done with students' programming log data, and 2) How can we curate these efforts to enable other researchers to replicate and build on that work? A hypothetical repository of CS educational data mining (CSEDM) tools might include metrics that predict student success (e.g. [1, 8, 20]), data-driven tools that help us understand, visualize or support student programming (e.g. [5, 18]), and methods to model student knowledge that inform task selection and provide feedback to the learners (e.g. [3, 21]). Such a collection, along with the growing availability of programming data for research (e.g. [2, 7, 9, 12]), would offer an opportunity to evaluate the generalizability of prior findings across datasets, to reuse tools across classrooms, or to validate predictive models across populations. It could help to address existing challenges with carrying out this work, which has historically required extensive collaboration (e.g. in [7, 19]). In this talk we will explore what a repository of CSEDM tools might look like, and focus on one particular challenge to implementing it: describing the data attributes that are required by CSEDM tools and present in datasets.

2 VISION FOR A CSEDM REPOSITORY

Efforts to replicate analysis of programming data have historically taken two approaches. The more common is to share datasets (e.g. [2, 10, 13]), allowing for secondary analysis by other researchers. While this is a critical effort that promotes collaboration and replication, it has limitations. Many datasets cannot be shared due to privacy or ethical concerns. Effective data sharing also requires effort on the part of the data provider to carefully document the data, and support the data consumer as challenges arise. Even when this is done well, researcher performing this secondary analysis still lose some critical context when using data they did not collect, which is often important for interpreting results. A second approach is to share analysis code and tools. This allows collaboration even when data cannot be shared and empowers new researchers to build on the state-of-the-art using their own datasets. While it is common for researchers to release the source code for their tools, it is much rarer to see them reused in practice, since the tools often require new users to adapt the code to their own data and to understand the tools' inputs and outputs. To overcome these challenges, a CSEDM tool repository would need the following:

1. A shared data format. A standardized format to represent programming log data, such as ProgSnap2 [15], can make it easier to share data and analysis code. In theory, a researcher who uses the ProgSnap2 format could release their analysis code (e.g. to calculate a metric that quantifies students' difficulty with syntax errors [8]), and any other researcher using the ProgSnap2 format could run that analysis code on their own datasets. ProgSnap2 consists primarily of a MainTable with rows for events (e.g. Compile) and columns for properties of those events (e.g. SubjectID, CompileMessageType). It can also support *Link Tables* that hold auxilary data, such as a mapping from AssignmentIDs to their respective descriptions.

2. A central analysis repository. LearnSphere's Tigris platform (learnsphere.org) provides a way for researchers to share and browse analysis code and apply it across datasets. The platform already contains reusable components for general analysis (e.g. logistic regression) and educational data mining (e.g. student modeling). While authoring a CSEDM tool as a Tigris component does create some overhead for the author, it also helps to standardize tool documentation, inputs and outputs, and allows components to be chained together into sharable workflows. Possible components in a CSEDM repository might include:

- *Metrics*, that summarizes the behavior of a student, a problem, or an entire dataset with as a numeric value.
- *Visualizations* that capture such behavior as a figure.
- *Models* that can be trained on the data and used to describe it succinctly or predict new behavior (e.g. student outcomes).
- *Data-driven tools* that use the data to offer some service to the student or teacher.

3. A vocabulary for required data attributes. Even when programming datasets use a shared format, in practice many analyses can only be run on datasets that capture specific information (e.g. compilation or keystroke events), or have specific properties (e.g. programs are written in Java). Programming data is rich and diverse, and datasets vary dramatically in granularity (keystrokes to submissions), size (tens to millions of programs), and programming context (compiled or interpreted; textual or block-based). While a common format makes it easier to identify this information, it is not explicitly represented, making it difficult to know which CSEDM tools can be applied to a given dataset, or conversely, which of the existing programming datasets can be used in a comparative analysis that relies on a given analysis (e.g. [7]).

Specifically, we define an **attribute** as a property of a dataset that is required to perform an analysis. Ideally, attributes should describe low-level properties of the data, rather than what can be

calculated from these properties. For example, one analysis may take as input the mean number of attempts students made on each problem, while another may require the number of problems attempted. However, both of these values can be calculated if the dataset records "Submit" events, which would make a more appropriate attribute. As a starting point, this talk will define four types of attributes (tied to elements of a ProgSnap2 dataset [15]), and explore them through case studies:

- **Events**: Types of events, e.g. Submit, Compile.Error.
- **Columns**: Properties of events that must be recorded for each one, e.g. SubjectID, Score.
- **Snapshot Properties**: Requirements of captured code snapshots, e.g. language (Java) or granularity (edit-level).
- **Link Tables**: Additional data about the programming context, linked to specific columns in the dataset, e.g. knowledge components or test cases for problems.

3 CASE STUDIES

3.1 Compilation Error Metrics

One active area of CSEDM research is compilation error metrics, such as the Error Quotient (EQ) [8], Watwin Score [20] and Repeated Error Density (RED) [1] metric. These metrics attempt to characterize the extent to which a given student struggles with syntax errors over a period of time as a single numeric value, and have been found to correlate with students' course outcomes. All three of these metrics require similar information about students' compilation behavior:

- **Compile** and **Compile.Error** Events: A record of each time a student compiles their code, and any associated errors
- **SubjectID** and **ProblemID** Columns: For each event, which student and problem it was associated with
- **CompileMessageData** Column: For each compile event, the specific compiler message shown to student

Importantly, while these approaches have very similar objectives, their data requirements differ. Calculating the Watwin score [20], for example, additionally requires information on when students run their code (**Run.Program** events), on which line of code compilation errors occurred (**SourceLocation** column) and an exact **Timestamp** column for each event. This demonstrates how data requirements can help identify applicable CSEDM approaches.

3.2 Modeling Student Knowledge

Student knowledge modeling is a powerful data mining technique that can be used to predict students' success on future problems[1], enable mastery learning [4], understand students' difficulties [17], and reflect that knowledge back to students [3]. Basic student modeling approaches only require a record of students' attempts at each problem, and whether each attempt was successful, which we can represent with the **Submit** event, along with the columns: **SubjectID**, **ProblemID** and **Score** (the correctness of the submission). However, to be effective, these techniques also require that each problem be labeled with one or more skills, or "knowledge components" (KCs), that it practices. This is a different kind of requirement that is most easily represented as a *LinkTable* [15] that maps ProblemIDs to their respective knowledge components. Alternatively, some authors have tried inferring these KCs automatically from student code, rather than requiring prespecified KC [17, 21]. This does not require a LinkTable, but it does introduce new requirements, such as the need to including the code itself (as a **CodeStateID**). Analyses that operate on code snapshots may require additional *Snapshot Properties* such as a specific programming language (e.g. Java, as in [21]) or the ability to represent the code as an abstract syntax tree (AST; e.g. in [17]).

3.3 Visualizing Student Progress

Visualizing how students navigate and solve programming problems can help identify important patterns and inform instruction. For example, prior work has visualized how individual students add and remove programming concepts over time [6], the paths that groups of students take when solving a programming problem [11], and how this "state space" can be discretized for increased interpretability [22]. These approaches have fewer data attribute requirements in common, though they each require **SubjectID**, **ProblemID**, and **CodeStateID** columns. Notably, visualization approaches can analyze data at different levels of detail, but are usually more effective with finer-grained snapshots (e.g. Edit-level). We can describe this property by specifying a required snapshot *Granularity*, such as Submission, Run, Save, Edit or Keystroke level data.

3.4 Data-driven Hint Generation

Many CSEDM tools are designed to automatically generate programming hints or feedback for students using a dataset of prior student programming logs [16, 18] (c.f. [14]). These tools differ from the other CSEDM approaches discussed here, since they are not designed for analysis, but rather to interact with the student through an interface. Still, including them in a CSEDM repository and identifying the dataset attributes they require could enable their wider dissemination. The ITAP hint generation algorithm [18] demonstrates another example of a *LinkTable* requirement, since like many hint generation algorithms, it relies on a set of test cases for a given problem to verify the correctness of code snapshots, including *newly generated* snapshots not found in the original dataset. These could be included as *LinkTable* requirement that maps ProblemIDs to test case code.

4 SUMMARY

Table 1 in Appendix A summarizes the attributes in 4 categories that we have identified, and identifies which analyses require each of these attributes. This helps us to identify important attributes that should always be logged, such as the SubjectID, ProblemID and CodeStateID columns, and others that are often useful, such as Submit events and the Score column. It also suggests which analyses will be easiest to compare across datasets, since they have few requirements. For example, the compilation error metrics should be easy to reproduce across datasets, allowing us to verify the generalizability of their claims to predict student outcomes (work which has already started [7]). As we have argued, this is an important step for creating a curated repository of useful CSEDM analysis code.

[1] For example, the CSEDM Data Challenge: go.ncsu.edu/data-challenge

REFERENCES

[1] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. (2016), 296–301. https://doi.org/10.1145/2899415.2899463

[2] Neil C. C. Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 223–228. https://doi.org/10.1145/2538862.2538924

[3] Peter Brusilovsky, Sibel Somyurek, Julio Guerra, Roya Hosseini, Vladimir Zadorozhny, and Paula J. Durlach. 2016. Open Social Student Modeling for Personalized Learning. *IEEE Transactions on Emerging Topics in Computing* 4, 3 (2016), 450–461. https://doi.org/10.1109/TETC.2015.2501243

[4] Albert T Corbett. 2000. *Cognitive Mastery Learning in the ACT Programming Tutor*. Technical Report. 1–6 pages. http://www.aaai.org/Papers/Symposia/Spring/2000/SS-00-01/SS00-01-007.pdf

[5] Elena Glassman, Jeremy Scott, Rishabh Singh, and Robert C. Miller. 2015. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *ACM Transactions on Computer-Human Interaction* 22, 2 (2015). http://people.csail.mit.edu/rishabh/papers/working/overcode.pdf

[6] R Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Exploring Problem Solving Paths in a Java Programming Course. *Proceedings of the Psychology of Programming Interest Group Annual Conference* (2014), 65–76. http://repositorium.sdum.uminho.pt/xmlui/bitstream/handle/1822/30076/PPIGproceedings.pdf?sequence=1{#}page=77

[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 ACM Conference on Innovation and Technology in Computer Science Education*.

[8] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the Third International Workshop on Computing Education Research*. 73–84. https://doi.org/10.1145/1151588.1151600

[9] Kenneth R Koedinger, Ryan S J Baker, Kyle Cunningham, and Alida Skogsholm. 2010. A Data Repository for the EDM community: The PSLC DataShop. In *Handbook of Educational Data Mining*, Cristobal Romero, Sebastian Ventura, Mykola Pechenizkiy, and Ryan SJd Baker (Eds.). CRC Press, 43–55. https://doi.org/doi:10.1201/b10274-6 arXiv:0709.1706v2

[10] Andrei Papancea, Jaime Spacco, and David Hovemeyer. 2013. An Open Platform for Managing Short Programming Exercises. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 47–52. https://doi.org/10.1145/2493394.2493401

[11] Chris Piech, Mehran Sahami, Joh Huang, and Leo Guibas. 2015. Autonomously Generating Hints by Inferring Problem Solving Policies. In *Proceedings of the ACM Conference on Learning @ Scale*. 1–10. http://web.stanford.edu/{%}7B{~}{%}7Dcpiech/bio/papers/inferringProblemSolvingPolicies.pdfhttp://web.stanford.edu/{~}cpiech/bio/papers/inferringProblemSolvingPolicies.pdf

[12] Thomas W Price and Tiffany Barnes. 2017. Position Paper: Block-based Programming Should Offer Intelligent Support for Learners. In *Proceedings of the 2nd Blocks and Beyond Workshop at VL/HCC*.

[13] Thomas W. Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the ACM Technical Symposium on Computer Science Education*.

[14] Thomas W Price, Yihuan Dong, Rui Zhi, Benjamin Paaßen, Nicholas Lytle, Veronica Cateté, and Tiffany Barnes. 2019. A Comparison of the Quality of Data-driven Programming Hint Generation Algorithms. *International Journal of Artificial Intelligence in Education* (2019).

[15] Thomas W Price, David Hovemeyer, Kelly Rivers, Austin Cory Bart, Andrew Petersen, Brett A Becker, and Jason Lefever. 2019. ProgSnap2: A Flexible Format for Programming Process Data. In *Proceedings of the Educational Data Mining in Computer Science Workshop in the Companion Proceedings of the International Conference on Learning Analytics and Knowledge*. 1–7.

[16] Thomas W. Price, Rui Zhi, and Tiffany Barnes. 2017. Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming. In *Proceedings of the International Conference on Educational Data Mining*.

[17] Kelly Rivers, Erik Harpstead, and Ken Koedinger. 2016. Learning Curve Analysis for Programming: Which Concepts do Students Struggle With?. In *Proceedings of the International Computing Education Research Conference*. 143–151.

[18] Kelly Rivers and Kenneth R. Koedinger. 2017. Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 37–64. http://link.springer.com/10.1007/s40593-015-0070-z

[19] Jaime Spacco, Paul Denny, Brad Richards, David Babcock, David Hovemeyer, James Moscola, and Robert Duvall. 2015. Analyzing Student Work Patterns Using Programming Exercise Data. (2015), 18–23. https://doi.org/10.1145/2676723.2677297

[20] Christopher Watson, Frederick W B Li, and Jamie L Godwin. 2014. No tests required: comparing traditional and dynamic predictors of programming success. *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14* (2014), 469–474. https://doi.org/10.1145/2538862.2538930

[21] Michael Yudelson, Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Investigating Automated Student Modeling in a Java MOOC. In *Proceedings of the International Conference on Educational Data Mining*. 261–264.

[22] Rui Zhi, Thomas W Price, Nicholas Lytle, and Tiffany Barnes. 2018. Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *Proceedings of the Educational Data Mining in Computer Science Education Workshop at the International Conference on Educational Data Mining*.

A REQUIREMENTS SUMMARY

Table 1: For each CSEDM analysis discussed in Section 3 (columns), this table summarizes the required data attributes (rows). Analyses and attributes are both grouped by category.

		EQ	WatWin	RED	Corbett (2000)	Rivers (2016)	Yudelson (2015)	Hosseini (2014)	Zhi (2018)	Piech (2015)	ITAP	SourceCheck
		Compilation Error			*Student Models*			*Visualization*			*Hints*	
Events	**Compile**	X	X	X								
	Compile.Error	X	X	X								
	Run.Test						X					
	Submit				X	X	X	X			X	X
	Run.Program		X					X				
	File.Edit								X	X	X	
Columns	**Order**	X	X	X	X	X	X	X	X	X	X	X
	SubjectID	X	X	X	X	X	X	X	X	X	X	X
	ProblemID	X	X	X	X	X	X	X	X	X	X	X
	CompileErrorType	X	X	X								
	SourceLocation		X									
	Timestamp		X									
	CodeStateID					X	X	X	X	X	X	X
	Score				X	X	X				X	X
Snapshot Properties	**Language**						Java	Java			Python	
	AST					X			X		X	X
	Granularity					Submit	Save	Run	Edit	Edit	Edit	Submit
Link Tables	**KCs**				X							
	TestCases					X					X	