ProgSnap2: A Flexible Format for Programming Process Data

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

David Hovemeyer
daveho@cs.jhu.edu
Johns Hopkins University

Kelly Rivers
krivers@andrew.cmu.edu
Carnegie Mellon University

Ge Gao
ggao5@ncsu.edu
North Carolina State University

Austin Cory Bart
acbart@udel.edu
University of Delaware

Ayaan M. Kazerouni
ayaan@vt.edu
Virginia Tech

Brett A. Becker
brett.becker@ucd.ie
University College Dublin

Andrew Petersen
petersen@cs.toronto.edu
University of Toronto

Luke Gusukuma
lukesg@cs.vt.edu
Virginia Tech

Stephen H. Edwards
edwards@cs.vt.edu
Virginia Tech

David Babcock
dbabcock@ycp.edu
York College

ABSTRACT

In this paper, we introduce ProgSnap2, a standardized format for logging programming process data. ProgSnap2 is a tool for computing education researchers, with the goal of enabling collaboration by helping them to collect and share data, analysis code, and data-driven tools to support students. We give an overview of the format, including how events, event attributes, metadata, code snapshots and external resources are represented. We also present a case study to evaluate how ProgSnap2 can facilitate collaborative research. We investigated three metrics designed to quantify students' difficulty with compiler errors—the Error Quotient, Repeated Error Density and Watwin score—and compared their distributions and ability to predict students' performance. We analyzed five different ProgSnap2 datasets, spanning a variety of contexts and programming languages. We found that each error metric is mildly predictive of students' performance. We reflect on how the common data format allowed us to more easily investigate our research questions.

KEYWORDS

compiler error metrics; data sharing; programming process data

ACM Reference Format:
Thomas W. Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M. Kazerouni, Brett A. Becker, Andrew Petersen, Luke Gusukuma, Stephen H. Edwards, and David Babcock. 2020. ProgSnap2: A Flexible Format for Programming Process Data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20), June 15–19, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341525.3387373

ITiCSE '20, June 15–19, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-6874-2/20/06…$15.00
https://doi.org/10.1145/3341525.3387373

1 INTRODUCTION

Analysis of programming process data, logged as students complete programming tasks, has furthered the field of computing education research (CER) in many ways, allowing researchers to gain insight into common programming errors [6], develop data-driven tools to support student learning [24], and improve plagiarism detection for take-home exams [13]. However, there are few common standards for how such data should be collected, represented, or shared, making it more difficult for researchers to collaborate, share tools, and replicate findings. Initiatives such as the PSLC Datashop [20] provide a common data format and tools to store, analyze, and share *generic* educational log data, accelerating the pace of educational research. However, programming datasets have a number of distinct, domain-specific features, which make it difficult to use generic formats. Programming datasets may track entire projects with multiple files, and interpreting them often requires specific metadata, such as the version of the IDE or compiler.

In this paper, we present ProgSnap2: a standardized format for logging programming process data. Researchers can use ProgSnap2 as a tool to assist in conducting computing education research, as it is specifically designed to support researchers in collecting, sharing and analyzing programming process data. The format was designed to prioritize the needs of both the *data producer* and the *data consumer*. For the data producer, ProgSnap2 makes exporting data straightforward, with a default structure to encourage best practices (e.g. what to log and how), a small set of required elements, and extensibility to support a variety of datasets. For the data consumer, it makes importing and analyzing data straightforward, while making explicit how the data were logged, and any caveats or specifics that might impact analysis.

The **primary goal** of ProgSnap2 is to enable the CER community to more effectively collaborate when analyzing programming process data. We see three primary use cases: **1) Sharing Data**: There is a high cost to sharing unstandardized data. Both parties must invest time for the consumer to understand and parse the new format. A common format lowers these barriers and increases

adoption, while improving the quality of new and existing logging systems by defining a standard set of events and attributes to log. Efforts to standardize the format and storage of learning data in other domains have led to datasets and research efforts that spanned multiple researchers and institutions [20]. **2) Sharing Analysis Code**: A common format also allows researchers to write analysis code that can be shared and reused on new datasets that have the same format. This enables researchers to collaborate, even when sharing data is not possible (e.g. for privacy reasons). Publishing analysis code can also encourage critical replication of CER, which is quite rare [12], and encourage the development of shared analysis libraries. For example, many researchers use the Error Quotient [17] to quantify learners' compilation behavior [3, 18, 23]. In this paper, we developed a public implementation of the Error Quotient, capable of operating on any dataset in the ProgSnap2 format, which saves future research effort and ensures consistency. **3) Sharing Tools**: A number of data-driven tools have been developed to support computing classrooms, such as student models [36] and on-demand hints [26, 29]. A common input data format would allow these tools to be more easily shared, reused, and composed together. These tools could even be published as services that any researcher can utilize, for example allowing any programming environment to employ an adaptive student model by sending its ProgSnap2 data to the appropriate service.

1.1 Related Work

Some prior work has attempted to standardize programming log data. The Marmoset system [31], for example, included a specification for logging edits, runs and tool messages. The Blackbox shared programming data repository [8] also uses a well-defined data format for BlueJ logs, and analysis of this data has led to valuable research insights, including 18 papers as of August 2018 [7] and at least two [19, 27] since, demonstrating the utility of shared data. However, neither format has been adopted in other systems, perhaps because of their system-specific representation. When asked about the most problematic features of the Blackbox dataset, researchers ranked the Java- and BlueJ-specific aspects of the data as most problematic [7], suggesting a need for a more general representation. Our work builds on the original ProgSnap [14] format, which defines a language-agnostic standard for representing code snapshots and submissions. ProgSnap2 extends this format by representing a richer set of event data and related resources, and by using a "flat" representation more suitable for direct analysis by statistical software. Outside of CER, the PSLC Datashop [20] is a platform for sharing log data from learning environments, defining a common format for this data. This work has lead to numerous papers and insights, as well as a platform for sharing educational analysis code to run on this data [21]. A similar model could greatly increase the CER community's ability to produce and replicate research findings [25].

A standard format should facilitate cross-institutional analyses with programming data, which have previously lead to important insights about the generalizability of findings. For example, Petersen et al. investigated the power of the Error Quotient metric (EQ; detailed in Section 3.1) to predict student grades across 6 datasets from 4 programming environments [23]. They found that results differed meaningfully across contexts, and that they were sensitive to the way the the EQ was parameterized. They conclude that "data-driven metrics for evaluating student behaviours must be scrutinized across a wide variety of contexts." We argue that this is true, and further work is needed, but that without a standard format, collaborative, cross-data investigations are challenging and require strong coordination among authors. Ihantola et al. [16] described some of these barriers in their attempt to re-analyze, replicate and reproduce results from prior CER papers with both new datasets and the original datasets. They found many necessary implementation details (e.g. how to define a programming session) were not specified in the original papers, leading to difficulty replicating results. These data-centric barriers to replication likely contribute to the very low percentage of replication papers in CER (2.38%) [12]. In this work, we show how ProgSnap2 addresses these barriers by allowing authors to share analysis code that operates across datasets, requiring minimal coordination, and enabling replication.

1.2 History and Development Principles

The ProgSnap2 format arose from the needs, experiences and real datasets of a large working group of researchers, working through the CS-SPLICE [1] project, an NSF-funded effort to develop standards, protocols, and learning infrastructure for CER. Initial concepts were adapted from 'DATASTAND" [32], a report from the Leveraging Programming and Social Analytics to Improve Computing Education Workshop at ICER 2017. Through several remote discussions from 2018-2019, we developed the following guiding principles for the standard: **1) Model diverse data**: ProgSnap2's data model is rich enough to describe real data from a variety of sources and contexts. It is grounded in the experiences of working group members, who are authors of systems that collect programming process data and researchers who use this data. **2) Specify what is known and unknown**: Diverse systems also differ in terms of what *is* and *is not* recorded. Therefore, the standard allows data providers to specify when information is unknown, rather than requiring them to create "synthetic" data values to fill in required fields. **3) Prioritize ease of analysis**: A standard is useful only insofar as it it *used*. To make analysis straightforward, ProgSnap2 uses comma-separated value (CSV) files, and a denormalized format for the main event table, allowing direct analysis with statistical software (e.g. R, Python). **4) Standardize best practices**: We designed the format to guide data-producers towards best practices for logging, so that ProgSnap2 would be useful for those designing new systems. For example, we have learned through experience that logging both *client* and *server* timestamps is useful for web-based systems, so *both* are encouraged fields in ProgSnap2.

2 THE PROGSNAP2 SPECIFICATION

A ProgSnap2[1] dataset consists of logs and relevant data that capture how users interacted with a programming environment. A dataset includes a *main event table*, a *metadata table* and optional *link tables* to reference outside resources, all represented as CSV files. A dataset also contains a *code repository* containing sequential snapshots of students' code and optional auxiliary *resources*.

[1]This paper describes ProgSnap2 version 6. See the ProgSnap2 website: http://bit.ly/ProgSnap2 for the most recent version of the standard.

2.1 Main Event Table

The main event table represents a collection of all events that took place in the programming environment. These events can represent both fine-grained interactions, such as individual keystrokes, and high-level actions, such as entire problem attempts, depending on the granularity of the logging system. Each row in the table represents one event, and each column represents an event property. All events have an EventType column, and ProgSnap2 provides over 20 predefined EventTypes, listed in Figure 1 (e.g. File.Edit, Compile.Error, Run.Program). Data providers can also define new EventTypes, prefixed with "X-" (e.g. "X-HintRequest"). ProgSnap2 defines a small set of mandatory columns: 1) **EventType**: a value indicating the type of event; 2) **EventID**: the unique ID of the event; 3) **SubjectID**: the ID of the human subject (or group) associated with the event; 4) **ToolInstances**: the names and versions of tools (e.g. IDE, compiler) associated with the event; and 5) **CodeStateID**: an ID for a snapshot of the source code when the event occurred.

ProgSnap2 also defines a variety of optional columns with standard names. These include *recommended columns* (e.g. ClientTimestamp, CourseID), which should always be included if available, but may not be recorded by every system. They also include *event-specific* columns, which are either required or recommended for certain EventTypes (e.g. CompileMessageType is only required for "Compile.Error" and "Compile.Warning" events). This leads to an intentionally sparse table. Data producers are encouraged to include as many optional columns and as much detail as possible. They can also define new columns when needed, prefixed with "X-". Examples of optional columns include: **ID Columns** (e.g. *CourseID*, *AssignmentID*): these provide contextual information for the associated event, and allow additional information to be specified by a "Link Table" (described below). **Edit Columns** (e.g. *EditType*, *CodeStateSection*): these describe how code was edited (e.g. typing, paste, undo) and where the edit took place. **Run Columns**: (e.g. *ProgramInput*, *CompileMessageData*): these record relevant information about how the code was compiled and run. **Experiment Columns** (e.g. *ExperimentalCondition*, *InterventionType*): these record data about experimental interventions used in research studies.

2.2 Metadata, Link Tables and Resources

The **Dataset Metadata** is a mandatory CSV file specifying the global properties of the dataset as key/value pairs, shown in Figure 1. **Link Tables** are optional files used to associate one or more ID values with a Resource providing more information. For example, a link table could associate a TermID/CourseID pair with the URL of a course website for that course and term. A **Resource** is an arbitrary data blob, identified by a URL in a Link Table, which can be either external (accessed via the internet) or internal (local to the dataset). The inclusion of Link Tables and Resources is optional but encouraged. We also encourage data producers to use internal resources (e.g. saving a static version of the course website in the dataset) to ensure they are not lost or changed.

2.3 Code State Representations

A Code State, or snapshot, represents the entirety of a student's code at one point in time and is often the focus of log data analyses. ProgSnap2 is intended to represent a variety of data, coming from single-function exercises, complex final projects, or block-based programs. To capture such diverse data, ProgSnap2 supports three source code representations: Git, Directory, and Table. This choice allows data producers to use the most appropriate representation, while constraining that choice to formats which are easily processed. Each format maps a CodeStateID value to a code state, which is simply a collection of one or more files with an optional directory structure. In the Git format, code states are represented as commits in a Git repository stored within the dataset. This format is appropriate for datasets where code states may consist of a relatively large number of files. In the Directory format, each CodeStateID maps to the name of a directory stored within the dataset which contains a collection of all files that are part of the code state. This format is appropriate for datasets where code states contain a small number of files. In the Table format, a dedicated CSV file maps CodeStateID values to source code. This format is only appropriate for datasets where each code state consists of a single text file, and where the amount of data per code state is small.

3 CASE STUDY

The goal of ProgSnap2 is to facilitate collaboration, replication, and the sharing of data and analysis. To evaluate how well it does so, we set out to replicate the same analysis to address the same research questions across five different programming datasets. Our collaboration context included common challenges for multi-institutional studies: 1) The datasets were diverse, collected in a variety of classroom contexts, using three different programming languages; 2) The analysis code was primarily written by two authors, who did not have access to three of the datasets, as they could not be shared for privacy reasons; 3) The collaborating authors were at different institutions, with limited communication outside of email. In this case study, we highlight the ways in which ProgSnap2 helped us to overcome these challenges and lessons we learned. We also show how this cross-dataset analysis can lead to unique research insights.

3.1 Compile Error Metrics

For our case study, we focus on compiler error metrics, a common type of programming data analysis which attempts to quantify the amount that students struggle with syntax errors [4]. The **Error Quotient (EQ)** [17] (revised in [30]) examines consecutive pairs of compilation events and assigns a score to each pair, which increases if both compilations include an error, and again if those errors have the same error type. The **Watwin score** [34] builds on the EQ by considering the time taken to resolve errors, the location (line number) of errors, and the full compilation message. The Normalized Programming State Model (NPSM) [9] further considers execution traces, debugging and idle time.

Compile error metrics have been shown to correlate with student's grades [17, 23, 33] (as have compiler error messages themselves [5]). This makes them useful for predicting at-risk students [33] and augmenting traditional measures of academic performance [35]. However, the predictive power of the EQ can vary across different contexts and datasets [23], and it does not predict students' grades in all contexts [16]. In response, Becker developed the **Repeated Error Density (RED)** metric to be less context-dependent, specifically for shorter programming sessions where the EQ may

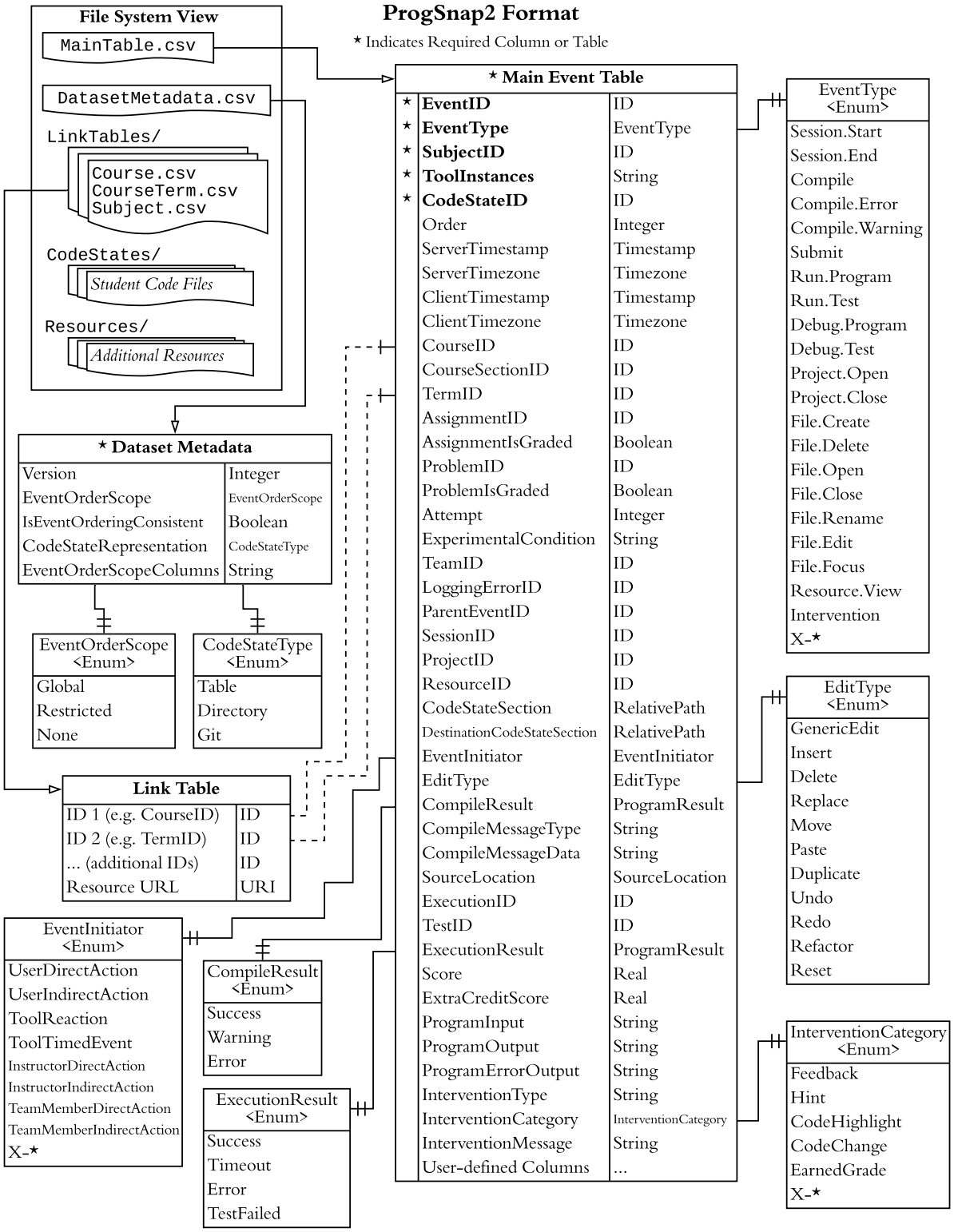

Figure 1: A diagram of the ProgSnap2 Format. Lines connect columns to their possible values, files to their respective tables, and IDs to their definitions in the Main Event Table.

Table 1: Statistics for the 5 datasets. The number of students removed by filtering are given in parentheses. The last three rows indicate average sessions per subject, number of compiles, and percent of compiles with errors.

System	CC	CWO	BlockPy	PCRS	ITAP
Language	C	Java	Python	Python	Python
Students	90 (-4)	410 (-3)	647 (-6)	1192 (-56)	73 (-16)
Exercises	86	50	244	99	38
Sess./Sub.	23.2	11.2	13.0	12.3	1.5
Compiles	40613	67046	460114	308830	2736
% Error	27.3%	41.3%	15.1%	20.5%	22.8%

be less accurate [3]. Despite much prior work, it is still a very open question *which of these metrics is most predictive of grades* and *how consistent that prediction is across datasets* – questions we address in this case study by comparing EQ, Watwin and RED.

3.2 Datasets

We focus our analysis on five programming datasets, collected from a variety of online practice environments used in university computing courses, summarized in Table 1. The **CloudCoder (CC)** dataset was collected in a CS1 course required for CS, Electrical/Computer Engineering, and Mathematics majors, taught using C at a small, undergraduate college in the Eastern United States. Students worked on 86 small (5-15 line) programming exercises in a web-based programming practice environment called CloudCoder [15] The **CodeWorkout (CWO)** dataset was collected during a CS1 course required for CS majors at a large, public university in the United States. It contains data from two sections of students who worked on 50 small programming exercises (10-26 lines) in Java, using a web-based practice environment called CodeWorkout [10]. Students completed multi-exercise homework assignments in CodeWorkout throughout the semester and could repeatedly attempt each exercise until getting it correct. The **BlockPy** dataset was collected in a CS1 course for non-CS majors in STEM disciplines at a large, public university in the United States. Students worked in a dual block/text programming environment called BlockPy [2] writing small programs (6-15 lines). The **PCRS** dataset was collected during a CS1 course for both CS majors and non-majors at a large, public university in Canada. Students worked in the PCRS [22] practice environment to complete short Python problems (3-15 lines) for weekly homeworks, receiving feedback from test cases. The **ITAP** dataset was collected during a CS1 course for non-majors at a private university in the United States. Students were invited to complete *optional* Python practice problems (2-10 lines) in the ITAP tutoring system [29], as part of a prior study [28] that lasted 7 weeks. Students could attempt the problems any number of times, receiving feedback from test cases and hints.

3.3 Method

We evaluated ProgSnap2's ability to assist in answering two research questions: RQ1) How effective are error metrics at predicting student grades, and is this consistent across datasets? RQ2) Are there meaningful differences between error metrics? While we use real RQs and attempt to generate real insight, our primary goal in *this work* is to create an authentic evaluation context for ProgSnap2.

3.3.1 Error Metric Implementation. Implementing analyses requires making a number of assumptions and decisions, and an important benefit of sharing analysis code among researchers is that these decisions are embodied in that code[2]. In this section, we detail the decisions we made when implementing our three error metrics, based on earlier work by Petersen et al. [23].

Sessions: Each error metric was designed to be calculated for a single student over a specific period of time, which we call a *session*. ProgSnap2 allows a dataset to provide a SessionID column, and we used this value when it was present. Otherwise, we defined a session as a series of events with no more than a 20 minute gap between any two consecutive events, as in [23].

Cleaning the Data: As in [23], we removed any session that contained only a small number of events: we required a minimum of 4 Compile events, since they are the subject of this study. We then removed any students from our analysis if their session count was more than 2 SDs below average for the dataset. Table 1 reports the number of students removed in this process.

Error Pairs: Each error metric operates over pairs of compilation events. Like Petersen et al., we extracted each consecutive pair of events from a session, and ignored any event pairs that occurred in *different* problems/assignments, or had the exact same code.

Python Compilation Errors: Though Python is an interpreted language, Petersen et al. [23] chose to count any Python runtime error as a compiler error. By contrast, our Python datasets only included Python `SyntaxError` exceptions as Compile.Error events.

Automatic compilation: BlockPy automatically compiles students' code as they work, presenting them with feedback (e.g. compilation errors [11]). As in [23], we created a Compile.Error event only when a student *ran* their code and received errors.

Algorithm Implementations: For EQ, we used the original algorithm proposed by Jadud [17], not the revised version [30]. The RED metric [3] is not normalized, meaning that it increases monotonically as a student continues to work. To make it comparable with the EQ and Watwin, we used a "normalized" RED, in which we divided the total RED score by the number of compilation pairs considered (as with the EQ). Watwin proposes cleaning code before analysis by removing comments, and to ignore edits in which code was deleted. We chose not to take this step to avoid programming-language-specific analysis.

We note that many of these decisions, such as the definition of a session and what constitutes a compilation or error event, are explicitly represented in the ProgSnap2 format, allowing each data provider to use definitions that make sense for their context.

3.3.2 Collaboration Context. We report here the iterative process by which our analysis was carried out, in order to illustrate the ways in which ProgSnap2 shaped our work. Eight authors contributed to the analysis, working at five different institutions. First, each dataset was converted into the ProgSnap2 format, since the datasets predated ProgSnap2. We defined a set of required EventTypes and column names for our compile error metric analysis (see [25]), and ensured that each dataset had these attributes. At the same time, one pair of authors worked to implement the EQ, RED and Watwin algorithms, along with procedures for cleaning and summarizing the datasets. However, this team did not have

[2]Code and public datasets can be found at: github.com/thomaswp/ProgSnap2Analysis

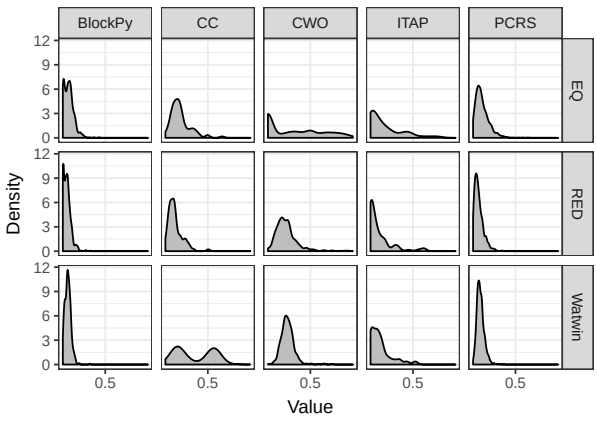

Figure 2: Distributions of each error metric for each dataset.

access to three of the datasets, which could not be shared easily due to privacy restrictions. This is a common challenge for secondary analysis like ours, and to overcome it, each team provided a *sample* ProgSnap2 dataset, consisting of non-student logs, which could be freely shared and used to test both our analysis code and the conversion to the ProgSnap2 format. This led to several iterations of both conversion and analysis code, ensuring that all required attributes were present, and debugging errors that occurred. Once we developed a stable version of the analysis code, each team ran it on their respective dataset and reported back the aggregate results.

3.4 Results and Discussion

RQ1: *How effective are error metrics at predicting student grades, and is this consistent across datasets?* Table 2 shows a Pearson correlation matrix for each error metric and for class grades across 5 datasets (though grades were only available in the CC, CWO and BlockPy datasets). All error metrics had a significant, negative correlation with student grades on the three datasets, meaning that students with higher EQ/RED/Watwin scores had lower overall grades. Across datasets and metrics, we found that error metrics explained between 3.6% and 21.8% of the variance in student grades. This range is consistent with most prior work [17, 23, 35]. Our results confirm the findings of Petersen et al. [23] that the EQ is predictive of student grades *across datasets*, but we further show that this is true of RED and Watwin as well. However, no metric explains much of the variance in student grades, suggesting the need for different predictive approaches.

RQ2: *Are there meaningful differences between error metrics?* As shown in Table 2, EQ, RED and Watwin are all highly correlated on all datasets (r = 0.854-0.988). Further, Figure 2 shows that the distribution of all error metrics are quite similar for any *given* dataset (e.g. PCRS is highly right-skewed for all metrics), but they also vary *across* datasets (CWO is much less so). As a result, Table 2 shows little difference in the correlation of each metric and student grades, suggesting that there is little difference in their predictive power. This result contrasts with prior work, which claimed that Watwin outperformed EQ [35], or theorized that RED would be less context dependent than EQ [3]. We hypothesize that these claims may have resulted in prior work because each error metric

Table 2: Correlations between each error metric, as well as each metric with class grades.

Dataset		EQ	RED	Watwin
CC	RED	0.988***		
	Watwin	0.963***	0.988***	
	Grade	-0.409***	-0.467***	-0.374**
CWO	RED	0.975***		
	Watwin	0.871***	0.903***	
	Grade	-0.363***	-0.357***	-0.300***
BlockPy	RED	0.991***		
	Watwin	0.860***	0.854***	
	Grade	-0.254***	-0.244***	-0.190***
PCRS	RED	0.983**		
	Watwin	0.923***	0.912***	
ITAP	RED	0.946***		
	Watwin	0.900***	0.788***	

Significance codes (p <): * = 0.05; ** = 0.01; *** = 0.001

was developed and evaluated on a *single* dataset, perhaps creating localized improvements that did not generalize to other datasets. While we did see instances of one metric outperforming another (e.g. RED outperforms EQ on the CC dataset), these trends reversed on other datasets. This highlights the importance of cross-dataset analysis, such as this one, and public datasets and benchmarks, which are all facilitated by ProgSnap2.

3.5 Reflection and Conclusion

Overall, we found that ProgSnap2 accomplished its goals of facilitating collaboration and allowing us to gain research insight: despite 10 years of research on error metrics, they remain similar and only mildly predictive of grades, calling into question the merit of continued research in this area. We found ProgSnap2 enabled us to easily share and analyze programming data. Data providers were able to export to the format with a reasonable amount of effort (a few days, typically), and the analysis code was straightforward to implement. For example, the first time the EQ analysis script was applied to the CC dataset, it ran without errors, even though the author of the EQ script had no direct access to the CC data. However, ProgSnap2 did not remove the inherent challenges of data analysis, and ultimately it took 4-5 iterations of improving data cleaning code, verifying results and fixing errors to finalize our results. Additionally, while ProgSnap2's EventType and column definitions were generally agreed upon, small differences in interpretation (e.g. what goes into CompileMessageType vs CompileMessageData) had to be resolved. We note that while our discussion in this case study was abbreviated for space reasons, we could extend our analysis to investigate when and why each metric was effective. Overall, our results show that ProgSnap2 has the potential to greatly facilitate collaboration and data sharing. Its utility will require wider adoption among researchers, and we invite those interested to use the footnote links to join the conversation, produce or convert a dataset, develop a new logging system, or contribute to cross-dataset analysis code.

4 ACKNOWLEDGEMENTS

This work was supported in part by NSF grants DLR-1740775, DLR-1740798, and DLR-1740765.

REFERENCES

[1] [n.d.]. SPLICE: Standards, Protocols, and Learning Infrastructure for Computing Education. https://cssplice.github.io/. Accessed: 2019-08-19.

[2] Austin Cory Bart, Javier Tibau, Eli Tilevich, Clifford A Shaffer, and Dennis Kafura. 2017. Blockpy: An open access data-science environment for introductory programmers. *Computer* 50, 5 (2017), 18–26.

[3] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 296–301. https://doi.org/10.1145/2899415.2899463

[4] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '19)*. ACM, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[5] Brett A Becker and Catherine Mooney. 2016. Categorizing compiler error messages with principal component analysis. In *12th China-Europe International Symposium on Software Engineering Education (CEISEE 2016), Shenyang, China, 28-29 May 2016*.

[6] Neil C.C. Brown and Amjad Altadmri. 2014. Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data. In *Proceedings of the Tenth Annual Conference on International Computing Education Research (ICER '14)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/2632320.2632343

[7] Neil CC Brown, Amjad Altadmri, Sue Sentance, and Michael Kölling. 2018. Blackbox, Five Years On: An Evaluation of a Large-scale Programming Data Collection Project. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM, 196–204.

[8] 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 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 223–228. https://doi.org/10.1145/2538862.2538924

[9] 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*. ACM, 141–150.

[10] Stephen H Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 188–193.

[11] Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. 2018. Misconception-Driven Feedback. *Proceedings of the 2018 ACM Conference on International Computing Education Research - ICER '18* 1 (2018), 160–168. https://doi.org/10.1145/3230977.3231002

[12] Qiang Hao, David H Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Andrew J Ko. 2019. A Systematic Investigation of Replications in Computing Education Research. *ACM Transactions on Computing Education (TOCE)* 19, 4 (2019), 42.

[13] Arto Hellas, Juho Leinonen, and Petri Ihantola. 2017. Plagiarism in Take-home Exams: Help-seeking, Collaboration, and Systematic Cheating. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, New York, NY, USA, 238–243. https://doi.org/10.1145/3059009.3059065

[14] David Hovemeyer, Arto Hellas, Andrew Petersen, and Jaime Spacco. 2017. Progsnap: Sharing Programming Snapshots for Research (Abstract Only). In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 709–709. https://doi.org/10.1145/3017680.3022418

[15] David Hovemeyer and Jaime Spacco. 2013. CloudCoder: a web-based programming exercise system. *Journal of Computing Sciences in Colleges* 28, 3 (2013), 30–30.

[16] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 41–63.

[17] 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

[18] Matthew C Jadud and Brian Dorn. 2015. Aggregate Compilation Behavior: Findings and Implications from 27,698 Users. In *Proceedings of the 11th International Computing Education Research Conference*. 131–139. https://doi.org/10.1145/2787622.2787718

[19] Ioannis Karvelas, Annie Li, and Brett A. Becker. 2020. The Effects of Compilation Mechanisms and Error Message Presentation on Novice Programmer Behavior. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 759–765. https://doi.org/10.1145/3328778.3366882

[20] Kenneth R Koedinger, Ryan SJd Baker, Kyle Cunningham, Alida Skogsholm, Brett Leber, and John Stamper. 2010. A data repository for the EDM community: The PSLC DataShop. *Handbook of educational data mining* 43 (2010), 43–56.

[21] Kenneth R Koedinger, John Stamper, and Paulo F Carvalho. [n.d.]. Sharing and Reusing Data and Analytic Methods with LearnSphere. *Hands-on* 2 ([n. d.]), 30p.

[22] Daniel Marchena Parreira, Andrew Petersen, and Michelle Craig. 2015. Pcrs-c: Helping students learn c. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 347–347.

[23] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. 2015. An Exploration of Error Quotient in Multiple Contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research (Koli Calling '15)*. ACM, New York, NY, USA, 77–86. https://doi.org/10.1145/2828959.2828966

[24] Thomas W. Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 483–488. https://doi.org/10.1145/3017680.3017762

[25] Thomas W Price and Ge Gao. 2019. Lightning Talk: Curating Analyses for Programming Log Data. In *Proceedings of SPLICE 2019 workshop Computing Science Education Infrastructure: From Tools to Data at 15th ACM International Computing Education Research Conference*.

[26] Thomas W Price, Rui Zhi, and Tiffany Barnes. 2017. Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming.. In *EDM*.

[27] Kyle Reestman and Brian Dorn. 2019. Native Language's Effect on Java Compiler Errors. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. Association for Computing Machinery, New York, NY, USA, 249–257. https://doi.org/10.1145/3291279.3339423

[28] 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.

[29] 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.

[30] Maria Mercedes T Rodrigo, Emily Tabanao, Ma Beatriz E Lahoz, and Matthew C Jadud. 2009. Analyzing online protocols to characterize novice java programmers. *Philippine Journal of Science* 138, 2 (2009), 177–190.

[31] Jaime Spacco, Jaymie Strecker, David Hovemeyer, and William Pugh. 2005. Software repository mining with Marmoset: An automated programming project snapshot and testing system. In *ACM SIGSOFT Software Engineering Notes*, Vol. 30. ACM, 1–5.

[32] John Stamper, Stephen Edwards, Andrew Petersen, Thomas Price, and Ian Utting. 2017. Developing a Data Standard for Computing Education Learning Process Data (DATASTAND). https://cssplice.github.io/DATASTAND.pdf. Accessed: 2019-08-19.

[33] Emily S. Tabanao, Ma. Mercedes T. Rodrigo, and Matthew C. Jadud. 2011. Predicting At-risk Novice Java Programmers Through the Analysis of Online Protocols. In *Proceedings of the Seventh International Workshop on Computing Education Research (ICER '11)*. ACM, New York, NY, USA, 85–92. https://doi.org/10.1145/2016911.2016930

[34] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*. IEEE, 319–323.

[35] Christopher Watson, Frederick W.B. Li, and Jamie L. Godwin. 2014. No Tests Required: Comparing Traditional and Dynamic Predictors of Programming Success. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 469–474. https://doi.org/10.1145/2538862.2538930

[36] Michael Yudelson, Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Investigating automated student modeling in a Java MOOC. *Educational Data Mining 2014* (2014), 261–264.