Assessing Student Engagement via IDE Instrumentation

Hsin-Cheng Chen

A report
submitted in partial fulfillment of the
requirements of the degree of

Master of Science in Computer Science and Software Engineering

University of Washington

2016

Project Committee:

Kelvin Sung, Chair

Rob Nash

Hazel Asuncion

Abstract

Introductory programming classes are challenging and have led to low retention rate in the Computer Science majors. Game-Themed Computer Science (GTCS) project aims at integrating video game creation in these courses as programming assignments. Students should be more engaged in the learning process if their assignments are entertaining. This project is designed to set up a proof-of-concept system to track programming sessions of students in introductory programming courses, specifically CSS161. Using post-analysis on the tracking data would provide insights into the pattern of students working with GTCS assignments, and potentials of in-depth understanding of the effectiveness of teaching with the GTCS curriculum. This project tracks patterns in student behavior while developing solutions to GTCS assignments, including: the number of times an IDE has been opened, total amount of time spent working with an IDE, the most common compiler errors, etc. In this way, the project explores approaches to gather relevant information, with results serving as a pathway for future investigation.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank University of Washington Bothell for supporting this project and providing all the necessary resources. I would like to thank Professors Kelvin Sung, Rob Nash and Hazel Asuncion for their invaluable advice, guidance, and support throughout the project. I would also like to thank Nancy Kool for assistance with the language and writing of this report. Last, but not the least, I would like to thank my friends and family for encouraging and supporting me through this Master's program.

Chapter 1
INTRODUCTION

The Game-Themed Computer Science (GTCS) curriculum[1] is designed to increase student engagement in learning introductory programming concepts through game creation. Although demonstrated to be effective in achieving the desired learning outcomes [14, 15], there is a lack of detailed information on how students work with the GTCS materials. Understanding of the actual sessions of students working with GTCS materials can provide further insights and point to refinement potentials. Instrumenting students working environment, in this case the Integrated Development Environment (IDE), can allow the tracking and recording of how students work to solve programming assignments. This instrumentation should be lightweight, and must add minimal if not zero performance impact to the system [11]. Software instrumentation can be implemented as a plugin to record data based on specific features or actions in the system. The plugin can produce connections between different segments of code as the students creates or modifies them [2].

The introductory programming course at the University of Washington Bothell, CSS161, has adopted the GTCS curriculum. A portion of this course teaches programming concepts through the creation of games. Students in this course use BlueJ[2] as the IDE for their programming assignments. The tracking of how students work with BlueJ could offer important details, e.g., how much time students spend on programming sessions, or which parts of the assignment they spent most effort on. In this way, the information could provide insights into the implications of working with and potential refinement strategies for GTCS curriculum.

This project aims at understanding how a software instrumentation system can be designed and implemented as a plugin to an existing IDE, tracking and collecting students'

[1]https://depts.washington.edu/cmmr/GTCS/.

[2]BlueJ http://www.bluej.org

working sessions with GTCS materials, and demonstrating the feasibility that additional insights can be gained based on the collected data.

This project would serve as the infrastructure and foundations for future research of the GTCS project. The results from this project include a local server for archiving collected information, services to transfer tracked data from the remote server, and a suite of proof-of-concept utilities and associated visualization system that demonstrate the potentials for extracting insightful information from the tracked data.

Chapter 2
BACKGROUND

2.1 Previous Research on Tracking Programmer's Behavior

Low retention rates in the computer science field could be a result of the abstract nature of the subject, which decreases the interests of students continuing into more advance classes [6]. By identifying where students struggle the most, teachers can help them get into better programming habits [13]. Difficult assignments can be identified by monitoring the students' activities and behavior during their programming sessions.

ClockIt [9] is one of several tools that could be used to monitor student activities. It is an unobtrusive plugin implemented for the BlueJ IDE. ClockIt is a tool set that collects data and provide analysis through data visualization consisting of a data logger, visualizer, and a web interface. The data logger records different activities when students use BlueJ and backs it up in a remote database. The visualizer then displays the recorded data in graphs through a web interface.

Another investigation was done using Retina, a tool that tracked information such as compiler errors, and the time spent to solve a bug or to complete an assignment [8]. This information was used to provide students with data that shows where they stand as compared to their classmates. By providing a comparison with other students, instructors can provide suggestions for improvements in areas students lack.

Previous researchers have spent in-depth time and resources to determine the kind of data to track and the possible correlation with students' performance. In this project, the research questions and the choice of the types of data to be tracked will be provided by our end user, Professor Rob Nash, who teaches CSS161. The remainder of this report will discuss the questions specified by Professor Nash and our findings.

2.2 Initial Instrumentation Attempt

We began our IDE instrumentation investigation by attempting to create a custom tool from the ground up catered to our needs. Our initial plan was to track mouse clicks, keyboard events, and the duration of the IDE session, based on Rabbit,[1] an instrumentation plugin for Eclipse.[2] Eclipse is a popular and advance IDE used by many programmers. The Eclipse IDE defines a powerful and sophisticated plugin architecture with a steep initial learning curve. Though with initial success, furnishing a complete tracking system within the project time constraint has proven to be challenging.

An investigation into alternate solution resulted in the BlackBox[3] project that is based on the BlueJ IDE. Although detailed actions like mouse clicks are not monitored, the Blackbox system tracks detailed higher level actions such as session time, project information, and function calls. More importantly, the system is complete, functional, with straightforward setup, and does not impact the performance of the IDE.

2.3 BlueJ and Blackbox

BlueJ is a freely available IDE designed specifically for teaching introductory programming classes. It has a simple interface with minimal options to avoid confusing or distracting the users. We chose BlueJ because of the Blackbox tracking project, and very conveniently, BlueJ is the IDE of choice in CSS161.

The Blackbox project was initiated by one of the investigators of the BlueJ IDE project to automatically and continuously collect data from BlueJ users across the globe. With explicit end-user permission, information on development session with BlueJ can be tracked and sent to a server located in the University of Kent, London. By allowing the end-user the option to define key phrases as the experiment and participant identifiers, Blackbox is able to distinguish and retrieve data of our specific experiment group among the millions of users. The tracking of a user in a *typical* programming section can generate hundreds of data

[1]https://code.google.com/archive/p/rabbit-eclipse/

[2]http://www.eclipse.org

[3]http://www.bluej.org/blackbox.html

points. The collected data are available to all researchers for analyzing and understanding behaviors of novice programmer.

We applied to and received Institutional Review Board (IRB) approval for tracking students' IDE work sessions provided that we guarantee the anonymity of individual students in class. Students in our CSS161 classes has the option of joining the study with a distinct experiment identifier that represents the UWB GTCS experiment,[4] and participant identifier that is associated with the instructor of their class.[5] In this way, it becomes possible to extract all participating students in the GTCS experiment according to which class they are enrolled in, and yet there is no way to distinguish or identify any individual student in class.

It is important to note that the Blackbox tracking system is designed for personal computers. Blackbox system associates an unique identifier with each individual computer and not the user that runs the BlueJ system. For this reason, it is impossible to correlate collected data from the same user working on multiple machines. It is also true that in a laboratory environment where computers are shared, it may not be possible to differentiate data from different users working on the same machine. In our case, with the unique participant identifier differentiating students according to their class instructors, we are able to differentiate students from different CSS161 classes that worked on the same computer. Although we are unable to reliably track the working habit of a particular student, we are able to monitor how students work with the system during each IDE session. By collecting and accumulating data from individual IDE working sessions we can derive average representative information for the entire CSS161 class.

[4] In this case the id is: *uwbgtcs*.

[5] For example, id of *css161nash* for Professor Nash's class.

Chapter 3
METHODS

3.1 Design

In addition to the design and instructions for students to configure the BlueJ instances, as depicted in Figure 3.1, there are two main functional components to this project: Data Transfer to local server, and Data Analysis from local server. The language of choice for implementation was PHP, over other languages like Ruby, JSP, PERL, or ASP. This is because PHP has a well-supported interface with popular databases like MySQL, and because of existing experience working with the language. Two sets of PHP utilities are created for each of the functional components for transferring and analyzing the collected data.

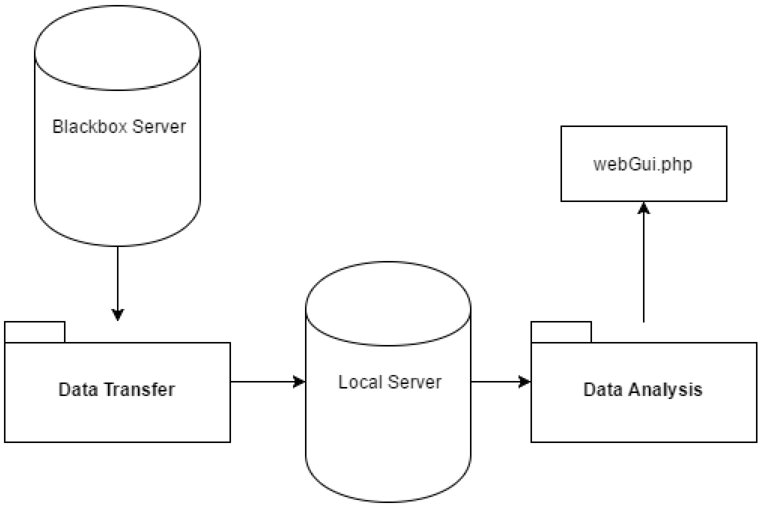
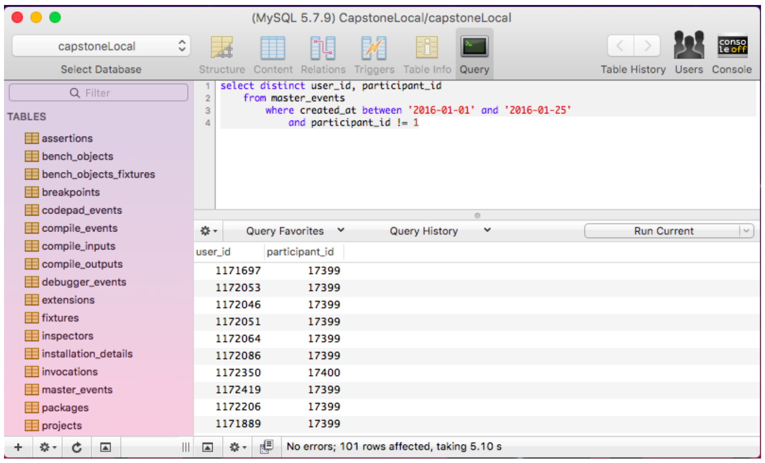


Figure 3.1: Overall Components of This Project

There are two important reasons to filter and transfer data from the remote Blackbox server. First, among millions of BlueJ users from around the world, we are only interested in the results from our own students. Second, the answers to the questions in which we wish to gain insights to require relatively intense data extraction and manipulations. The Blackbox server is a shared resource among all researchers and it is important to avoid computationally intensive processing on their server. For these reasons, we replicated the MySQL database structure onto our local server, and designed the data transfer component to extract and duplicate only those data from our own students. A checkpoint logging system is maintained by the data transfer system to ensure interrupted transfers are recoverable and only new data are transferred.

The data analysis component consists of a suite of PHP-based utilities organized according to the research questions posted by our end-user, Professor Nash. The database access operations, GUI functionality, and graphical visualization support are grouped such that future additional research questions can be answered with minimal coding.



```
select distinct user_id, participant_id
    from master_events
        where created_at between '2016-01-01' and '2016-01-25'
            and participant_id != 1
```

user_id	participant_id
1171697	17399
1172053	17399
1172046	17399
1172051	17399
1172064	17399
1172086	17399
1172350	17400
1172419	17399
1172206	17399
1171889	17399

```
No errors; 101 rows affected, taking 5.10 s
```

Figure 3.2: Screenshot of SequelPro database management software displaying query result

3.2 *Implementation*

3.2.1 *Local MySQL Database storage*

A local MySQL database system is created by replicating the schema of that of the Blackbox project. SequelPro[1] is used to manage the database, and test SQL queries. A screenshot

[1]http://www.sequelpro.com

of the application can be seen in Figure 3.2. SequelPro has a simpler and friendlier user interface than MySQL Workbench, the default MySQL database management tool.

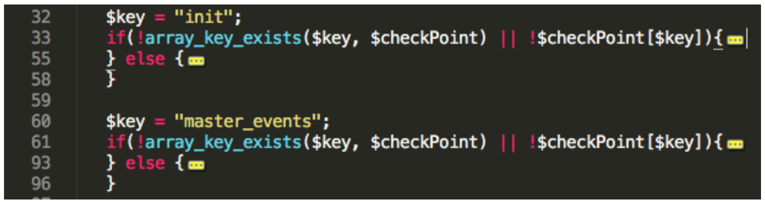
3.2.2 Implementing Data Transfer

The transferring of data from the remote server consist of three phases: connecting to the remote database, identifying all relevant users, and downloading the relevant table entries. To properly support these operations, the remote database tables are carefully studied and the relationships among the sixteen relevant tables are identified.

The Checkpoint Support

Due to the quantity of data and the performance of the remote server, the duration of a data transferring session can be in the order of multiples of days. A per-table checkpoint system is established to ensure reliability and to avoid unnecessary repeats of downloaded information. The system tracks and logs each successfully transferred table and can be interrupted/restarted reliably without repeating the transfer of unnecessary data.

Figure 3.3 shows setting up checkpoints with an array key and initiating table download.

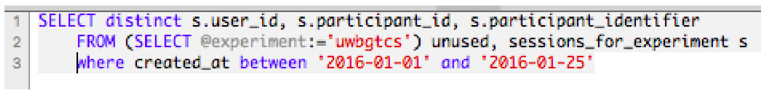


```
$key = "init";
if(!array_key_exists($key, $checkPoint) || !$checkPoint[$key]){
} else {
}

$key = "master_events";
if(!array_key_exists($key, $checkPoint) || !$checkPoint[$key]){
} else {
}
```

Figure 3.3: Screenshot of Setting Up Checkpoints and How To Resume

The Master Event Table

The *master_ events* table records all tracked events, with foreign keys that refer to other tables where details of the events are contained.



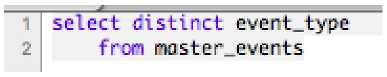
```
SELECT distinct s.user_id, s.participant_id, s.participant_identifier
    FROM (SELECT @experiment:='uwbgtcs') unused, sessions_for_experiment s
    where created_at between '2016-01-01' and '2016-01-25'
```

Figure 3.4: Query provided by Blackbox to retrieve relevant users among the millions of others

As discussed, *experiment_ identifier* and *participant_ id* together identify students in our CSS161 classes with their respective instructors. The *user_ id* identifies unique machine that a BlueJ IDE session is running from. In this way, the unique *uwbgtcs experiment_ identifier* allows us to extract the list of all our students as depicted in Figure 3.4. This query returns a data set with the fields *user_ id*, *participant_ id* and *participant_ identifier*, which is used for analysis and retrieval of data from other related tables.

All Events Triggered By Our Students

The query in Figure 3.5 returns all the unique events that was triggered by our students. An event is tracked action performed by the user in the BlueJ session. For example, an event will be recorded when a user opens the debugger. It is important to note that not all actions performed by the users are tracked, e.g., selection of a menu item is not tracked by Blackbox.



```
select distinct event_type
    from master_events
```

Figure 3.5: Query to return unique events triggered by our students

Each *event_ type* represents a table we need to download. For example, a *event_ type* of *invocations* refers to the *invocations* table which contains detailed information associated with all invoked events. Tables that include events that are triggered by our users include *invocations*, *compile_ inputs*, *sessions*, etc.. As discussed, there are a total of 16 unique tables.

Details of Important Tables

A *session* begins when BlueJ is first opened. Information such as time-stamp and which user opened the session will be recorded in the *sessions* table. The time-stamp allows us to investigate the engagement level of students by the duration or the accumulation of sessions over a period of time.

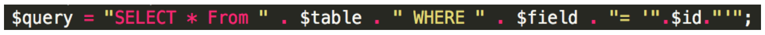
Information regarding compilation is recorded in three tables: *compile_ inputs*, *compile_ outputs*, and *compile_ events*. Information such as the name of a source code file and compile event id are contained in the *compile_ inputs* table. Compile event id is the foreign key that reference back to the main records in the *master_ events* table.

Information such as the name of the method, the session which the method was called, and whether the method call was successful were recorded in the *invocations* table. Using the *invocations* table, we can investigate how often students called functions related to GTCS materials and use the results as an indicator for engagement.

Lastly, there are tables which we do not need because they are information unrelated to events triggered by students such as *tests* and *version_ control_ events*. The *tests* table is triggered when users execute a test from a test class. Using of the test class is not a part of the CSS161 introductory programming course. The *version_ control_ events* table records events when users maintain their code using a SVN or CVS protocol. Assignments in CSS161 are usually small in scale and typically do not require repository support.

Query for Data Transfer

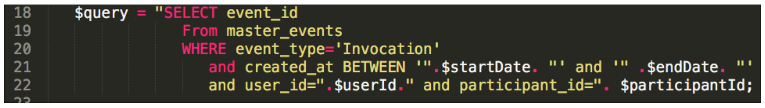
The most important part about downloading a table is creating the correct query. Figure 3.6 shows the query that was used to download many tables. This query returns all columns from *$table* where the *$field* is of value *$id*.



```
$query = "SELECT * From " . $table . " WHERE " . $field . "= '".$id."'";
```

Figure 3.6: Generic Query for Data Transfer

Figure 3.7 illustrates the query that retrieves the *invocations* table. An *invocation* record in the *master_events* table has an unique *event_id* that served as the foreign key to obtain details from the *invocations* table. In MySQL database, data can be quickly retrieved with primary keys. Unique keys for different tables are listed in the Blackbox instruction document.



```
$query = "SELECT event_id
            From master_events
            WHERE event_type='Invocation'
              and created_at BETWEEN '".$startDate. "' and '" .$endDate. "'
              and user_id=".$userId." and participant_id=". $participantId;
```

Figure 3.7: Query to retrieve all "invocations" event from the "master_events" table

The term following the *SELECT* clause determines the desired field to retrieve from where the * symbol represents all fields in the table. We specified the name of the table after the *FROM* clause. Any parameters we add after the *WHERE* clause determines the specific condition for returning the desired data.

Implementing Data Analysis

Figure 3.8 shows the structure of the data analysis component. Core and shared functions including many database queries are organized in the *coreFunctions.php* file, the *graphFunctions.php* source file contains common graphing visualization functionality, and general user interface and display supports are defined in *guiPage.php*. Each sub component decomposes posted research questions into relevant SQL queries for efficiency and associated computations.

All posted research questions are answered based on scripts constructed in a similar process as described in the following. We begin by identifying which are the tables that contain the relevant information, followed by finding out the unique keys that correlate to the events in the *master_events* table, then constructing simple queries that access individual tables, and finally aggregating and computing results based on the retrieved data. In this process, one of the key principles is to use multiple simple queries instead of a single complex query, because complex query is often more resource intensive and slow [12]. A complex query searches for more than one parameters in more than one table. Whereas a simple query only

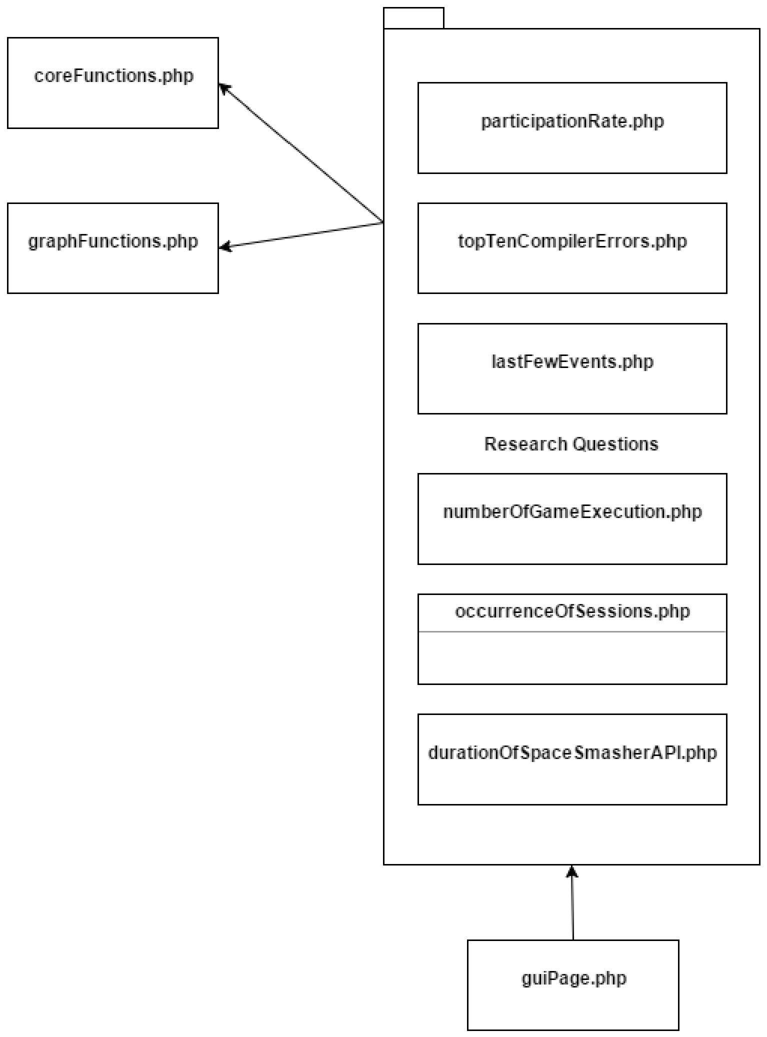
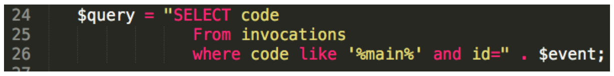


Figure 3.8: Data Analysis and Graph Generation Scripts

searches for parameters in one table. Iteration of simple queries was used to ensure code readability for future researchers and maintainability.

The following is a demonstration on how the above process is followed in answering one of the posted research question: how many times students ran the GTCS game.

In BlueJ, students ran the GTCS game by calling the *main()* method. A record was generated in the *master_ events* table with the *event_ type* as *invocations*. Followed by additional records generated in the *invocations* table with more details when a function is called. Relevant *invocation* events are retrieved using the query as shown in Figure 3.7. To count the number of method calls that contains the word *main*, we first need the list of unique users. The list of unique users will be used to fill out the *user_ id* and *participant_ id* fields. The *created_ at* provide a time frame to better locate students data according to when the assignments were assigned. Next, we matched each invocation event to the corresponding record in the *invocations* table using *event_ id* as depicted in Figure 3.9.



```
$query = "SELECT code
            From invocations
            where code like '%main%' and id=" . $event;
```

Figure 3.9: Query to match each invocation events to look for "main" in the "code" column

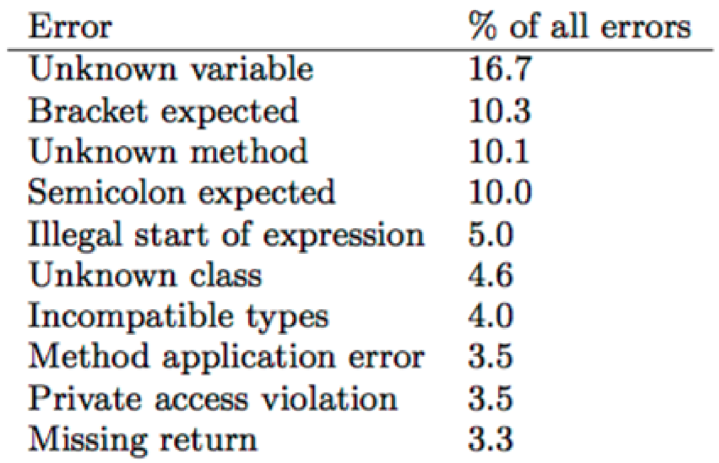
All other research questions are answered following similar process of constructing iterations of simple queries to retrieve the required data for analysis (see appendix A.2).

Chapter 4
RESULTS / IMPLICATIONS

Findings based on answers to the research questions posted by Professor Nash are presented in this section.

4.1 Top Ten Compiler Errors

Based on the data collected by the Blackbox tracking system, the top ten compiler errors can be summarized as shown in Figure 4.1 [7].



Error	% of all errors
Unknown variable	16.7
Bracket expected	10.3
Unknown method	10.1
Semicolon expected	10.0
Illegal start of expression	5.0
Unknown class	4.6
Incompatible types	4.0
Method application error	3.5
Private access violation	3.5
Missing return	3.3

Figure 4.1: Top ten errors from previous researcher Jadud

We would like to verified that the tracking is functioning properly for our students and thus is able to produce similar results. We were able to demonstrate similar errors tracked as shown in Figure 4.2.

A major difference between Figures 4.1 and 4.2 is the duration of data tracked, the previous researcher worked with results from participants over a much longer period of time. An interesting observation is that while missing semicolon is the top on our list, it is lower

Errors	% of all errors
Semicolon expected	15.3%
Illegal start of expression	9.8%
Class, interface, or enum expected	6.6%
Not a statement	3.9%
Bracket expected	3.7%
Identifier expected	3.5%
Reached end of file while parsing	2.7%
Illegal start of type	2.6%
'.class' expected	1.8%
Missing return statement	1.2%

Figure 4.2: Top ten errors we generated

when compared to previous result. The table shows errors that our students encountered most and can vary according to the instruction given. This result is a demonstration that the tracked results are well within anticipation.

4.2 Participation rate per instructor

By knowing which section users are from, it allows us to collect more data from the least participated class section. The distribution of our students is shown in Figure 4.3. During the first week of class, we instructed all students who are willing to participate to setup for data collection. An interesting observation is that there are a few instructor who only had one participating student. This could be a result from a class where no students want to participate in the research. The low participation rate could also suggest students might have a higher preference to do work on their machines than on school's lab machines.

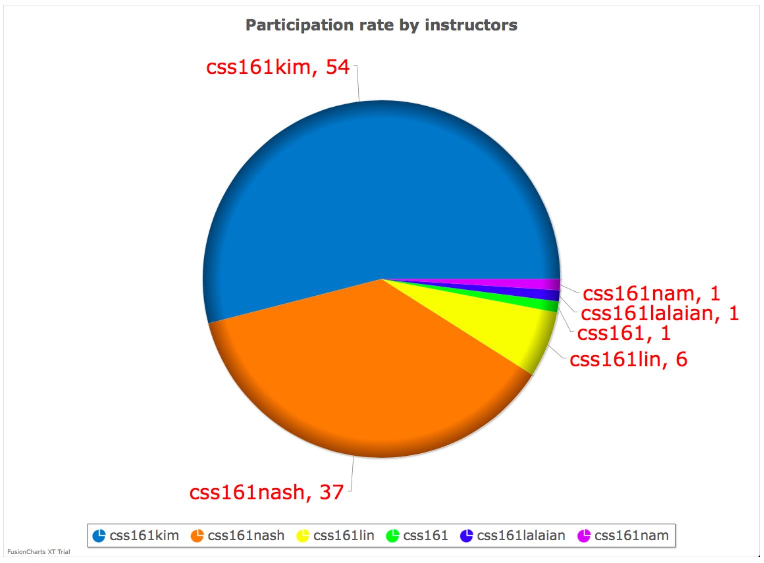


Figure 4.3: Participation rate per instructor

4.3 Number of sessions per user

We are interested in finding out if the GTCS course would lead to higher level of engagement with programming. The number of sessions invoked by students during an assignment period could indicate efforts. Our results shown in Figure 4.4, has an average of 16 sessions per day throughout a month of development while students worked on the first assignment.

The number of average sessions per day was 16, and that is considerably high. A session is when BlueJ opens. Having 16 sessions means that BlueJ has been opened on a machine 16 times in a day. Possible reasons for that many opening could be distractions from other things. However, it could also be true that the GTCS assignment have influenced students to open the assignment more often in a day.

4.4 Timing of sessions throughout the assignment

Knowing that BlueJ has been opened multiple times in a day, we would like to know when students usually work on the assignment. We hope the GTCS assignments would increase students' interest to work more often. Our graph in Figure 4.5 shows the occurrence of sessions per day through out the period when assignment one was assigned.

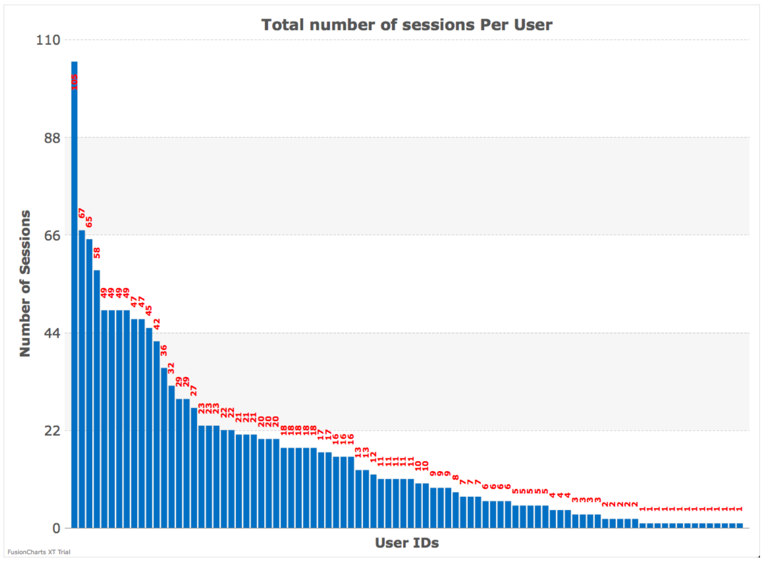


Figure 4.4: Total number of sessions per user in assignment 1

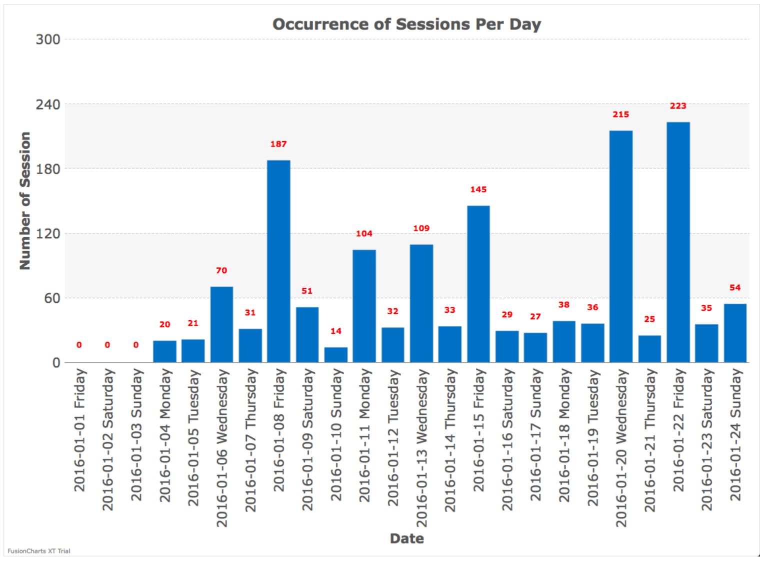


Figure 4.5: Occurrence of sessions per day

The highest occurrence of sessions were Friday when there was a lab session and assignments were due. Follow by Saturday with the second highest of occurrence that suggested many students were working due to late submission.

4.5 Number of times the game was executed

After getting a sense of when students work, we would like to know how many times they ran the game throughout the development. A high number of method calls could have a correlation with students' level of interest in the assignment. Our data showed a rather interesting outcome. Figure 4.6 is a graph of the total number of times when a user runs the game.

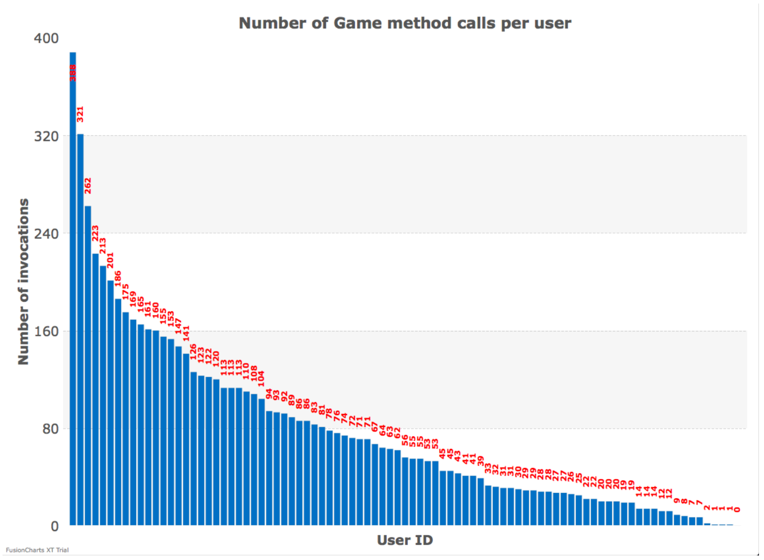


Figure 4.6: Number of Game Method Calls Per User

The average number of game invocations was 77 times. This suggests that, on average, the game was started 77 times on each of the tracked machine during the assignment period. The average translates to running the game about twice a day, which is rather low even if the assignment was not a game. In reality, the average should have been higher because students would have to debug or test the game. Another factor that might have affected the data is that the majority of the data could be coming from the lab machines in school. Which could be the low number of calls during a lab session. Nonetheless, there are over 50% of the students from our data to have ran the game for more than 77 times. With one particular students that ran the game for 388 times. Therefore, the GTCS assignment might have some influence in engaging students to work more often.

4.6 Last few events before user closed BlueJ

By understanding what students do right before they close BlueJ, we might be able to see if GTCS motivates students to work more. Our result in Figure 4.7 displayed the last 20 events before BlueJ was closed.

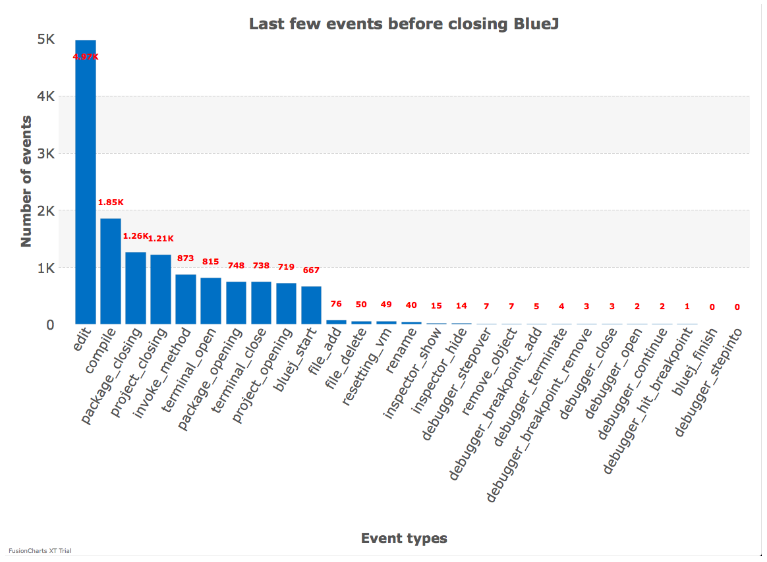


Figure 4.7: Last 20 Events Before BlueJ Closed

The most frequent event was *edit* which is normal considering the majority of any programming session would be editing. An *edit* event in BlueJ is generated when there are changes to multiple lines of code or returning of a caret. A more obvious indication that the GTCS assignment has increased students' interest should be a higher number of *invoke_method* events. The result could suggest potential patterns in a game development with more data.

4.7 Time spent in the SpaceSmasherAPI package

We provided students with the basic packages required to develop each game. The package include basic functions that generated different objects in the game. We would like to know if students were intrigued enough by the assignment that they start modifying the basic package. The graph in Figure 4.8 shows the total amount of time in minutes that students have spent in the package.

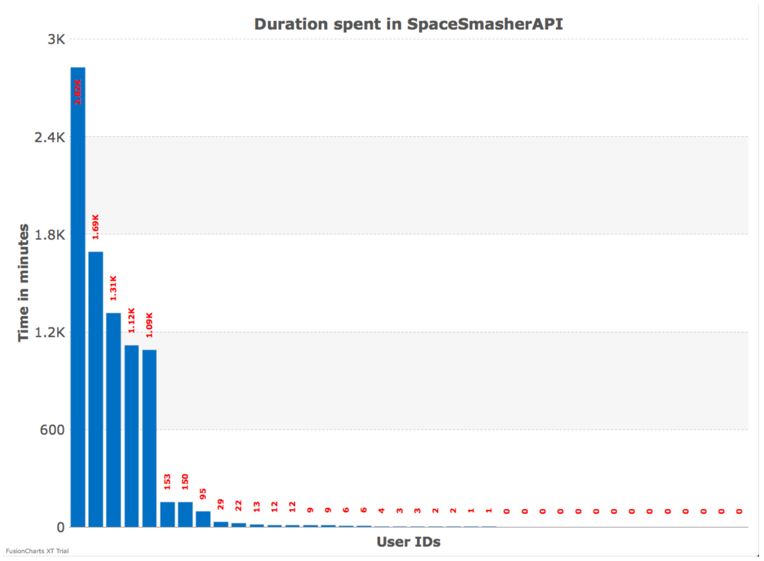


Figure 4.8: Duration of Spent in the SpaceSmasher Package

There are a few students who have spent over 2800 minutes within the *SpaceSmasher* package throughout the first assignment. Though we don't know what students were doing in the package, it is an encouraging sign that some students were interested in modifying the game beyond the given sets of instructions.

Chapter 5
CONCLUSION AND FUTURE WORK

We set out to investigate the possibilities of studying the effectiveness of the GTCS course materials through instrumentation of the IDE. After initial researches and experimentation with creating a custom plugin from the ground up, we concluded that given the project time frame adopting an existing IDE instrumentation system is a better solution. The Blackbox system is adopted because of its completed and stable functionality, and because BlueJ is the IDE of choice in our existing CSS161 classes.

Base on the Blackbox tracking system, an infrastructure was developed to demonstrate the potentials of studying effectiveness of course materials based on IDE event tracking, and to support future researches into GTCS coursework contents. The developed infrastructure includes the support for reliable data transfer from the remote server, local storage and maintenance, and answering research questions. The effectiveness of the results from this project is demonstrated using the infrastructure to answer sample research questions posted by Professor Nash. While designing answers to the posted questions, we identified and established a template approach to construct queries for analyzing the collected data. We are confident that the continuation of this project will prove useful in studying the detailed student interaction with the GTCS materials.

5.1 Future Work

The results from this project demonstrated feasibility of using the tracking data to provide insights into students' interaction with the GTCS materials when they work on the assignments. This work is just beginning and there are many areas for improvement. For example, complete automation of the data transfer process, extending and refinement of the user interface, parameterization of data analysis functions, etc.

Data transfer process from the remote server could be improved in several ways. One way is to parallelize the PHP script to distribute the download tasks into multiple threads. This will require redesigning the process to handle the returned data set. Another way to improve the downloading process is to parallelize the MySQL query to return result in smaller chunks. Further investigation is required to verify if there will be an increase in network traffic by maintaining connection until all result chunks are received.

More data is needed in order to make implications more accurate in regards to GTCS. During this research, we weren't able to collect data from a control class where introductory programming was taught without the GTCS curriculum for comparison. Data from the control class could serve as a great contrast to differentiate the implications of GTCS materials.

The collected data can show the overall performance of the class, but not of individuals. We could choose to focus on individual students by assigning each student a separate identifier instead of a common identifier for the whole class. However, it is crucial to follow IRB protocol and ensure anonymity of the participants.

It is also possible that the students may have forgotten to enable tracking on some systems. We can improve the reliability of the data by ensuring that they enable data tracking on all computers they work on, including lab machines and personal computers.

The current GUI web page presents the data with one graph per research question with a limited number of parameters to work with. Better data visualization can be improved by adding more parameters to the queries. Addition of different data visualization could produce more perspectives into GTCS courses.

Lastly, further investigation on the tracking data in Blackbox could inspire new questions regarding the context of GTCS. There are tables with data that we have not used due to unfamiliarity with the internal table relations. We had difficulties in attempting to trigger some of the events as described in the Blackbox handbook. Without understanding the relations, it is difficult to see the usability of those tables.

BIBLIOGRAPHY

[1] Amjad Altadmri and Neil CC Brown. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 522–527. ACM, 2015.

[2] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. Software traceability with topic modeling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 95–104. ACM, 2010.

[3] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.

[4] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. Blackbox: A large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 223–228. ACM, 2014.

[5] Gregory Dyke. Which aspects of novice programmers' usage of an ide predict learning outcomes. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 505–510. ACM, 2011.

[6] James B Fenwick Jr, Cindy Norris, Frank E Barry, Josh Rountree, Cole J Spicer, and Scott D Cheek. Another look at the behaviors of novice programmers. *ACM SIGCSE Bulletin*, 41(1):296–300, 2009.

[7] Matthew C Jadud. *An exploration of novice compilation behaviour in BlueJ*. University of Kent, 2006.

[8] Christian Murphy, Gail Kaiser, Kristin Loveland, and Sahar Hasan. Retina: helping students and instructors based on observed programming activities. *ACM SIGCSE Bulletin*, 41(1):178–182, 2009.

[9] Cindy Norris, Frank Barry, James B Fenwick Jr, Kathryn Reid, and Josh Rountree. Clockit: collecting quantitative data on how beginning software developers really work. *ACM SIGCSE Bulletin*, 40(3):37–41, 2008.

[10] Daniel Olivares. Exploring learning analytics for computing education. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 271–272. ACM, 2015.

[11] Antonio Sabatini, Nathan Jarus, Pushp Maheshwari, and Sahra Sedigh. Software instrumentation for failure analysis of usb host controllers. In *Instrumentation and Measurement Technology Conference (I2MTC), 2013 IEEE International*, pages 1109–1114. IEEE, 2013.

[12] Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko. *High performance MySQL: Optimization, backups, and replication.* " O'Reilly Media, Inc.", 2012.

[13] SC Shaffer. A brief overview of theories of learning to program. *Psychology of programming interest group newsletter*, 2005.

[14] Kelvin Sung, Cinnamon Hillyard, Robin Lynn Angotti, Michael W Panitz, David S Goldstein, and John Nordlinger. Game-themed programming assignment modules: a pathway for gradual integration of gaming context into existing introductory programming courses. *Education, IEEE Transactions on*, 54(3):416–427, 2011.

[15] Kelvin Sung, Rob Nash, and Jason Pace. Building casual game SDKs for teaching CS1/2, a case study. Proceedings of the Annual CCSC-NW Conference, To Appear.

[16] Ian Utting, Neil Brown, Michael Kölling, Davin McCall, and Philip Stevens. Web-scale data gathering with bluej. In *Proceedings of the ninth annual international conference on International computing education research*, pages 1–4. ACM, 2012.

[17] Jason Vandeventer and Benjamin Barbour. Codewave: a real-time, collaborative ide for enhanced learning in computer science. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 75–80. ACM, 2012.

[18] Christopher Watson and Frederick WB Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 39–44. ACM, 2014.

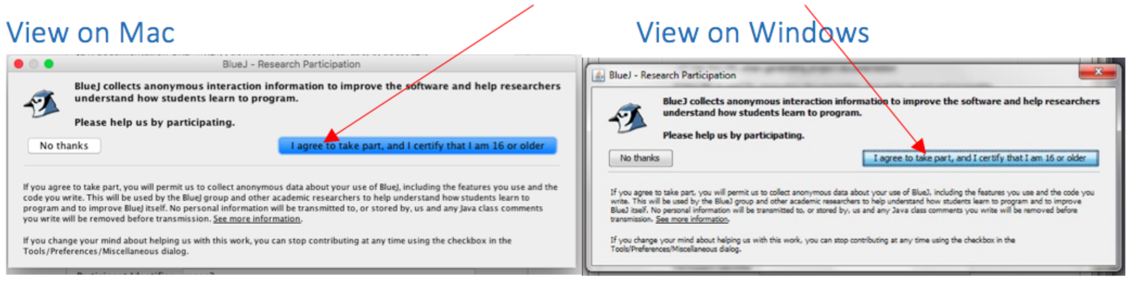
[19] Craig Watson, Frederick WB Li, and Jamie L Godwin. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 319–323. IEEE, 2013.

Appendix A

A.1 Enable Tracking on BlueJ

1. Install BlueJ (www.bluej.org)

2. Upon first execution after installation, click on the "I agree to take part, and I certify that I am 16 or older" button when prompted. If this prompt is missing, move on to step 3.



BlueJ - Research Participation

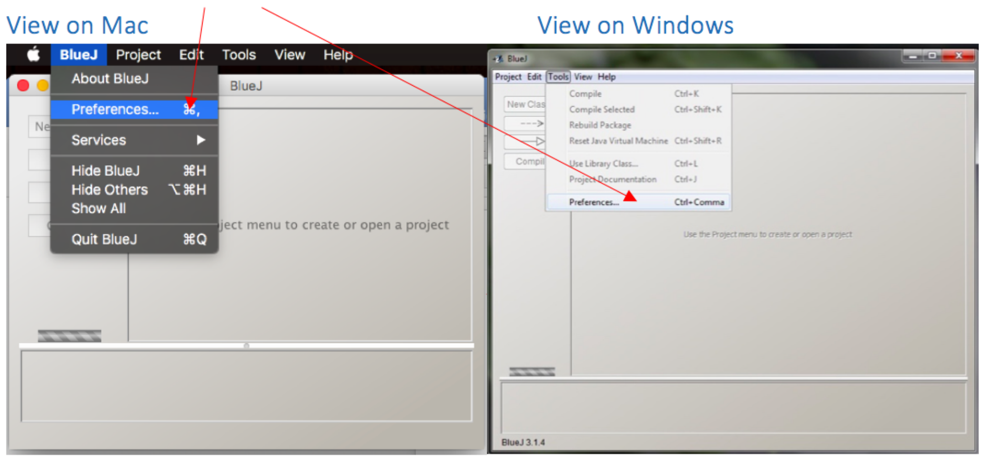
BlueJ collects anonymous interaction information to improve the software and help researchers understand how students learn to program.

Please help us by participating.

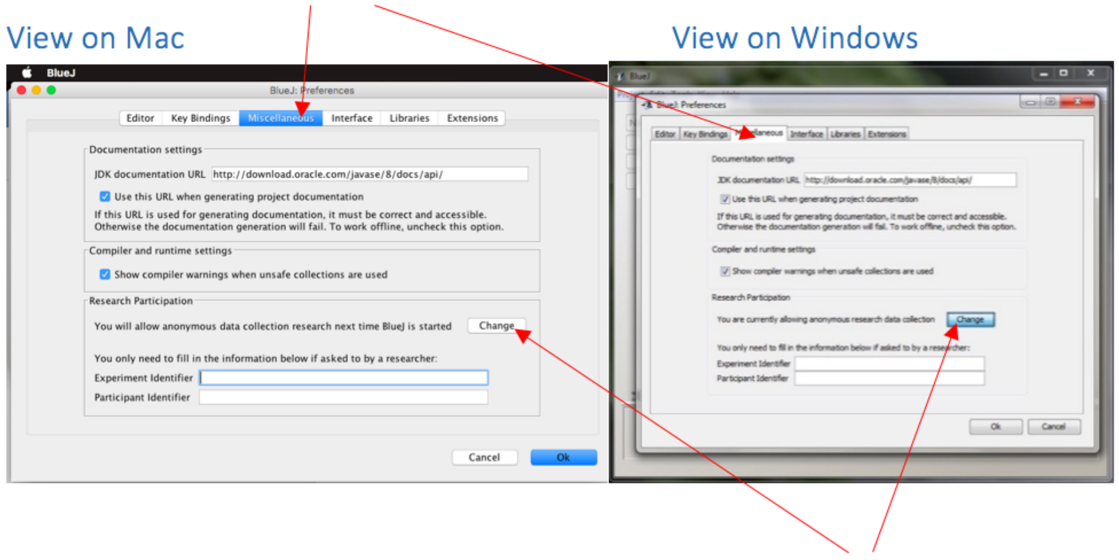
If you agree to take part, you will permit us to collect anonymous data about your use of BlueJ, including the features you use and the code you write. This will be used by the BlueJ group and other academic researchers to help understand how students learn to program and to improve BlueJ itself. No personal information will be transmitted to, or stored by, us and any Java class comments you write will be removed before transmission. See more information.

If you change your mind about helping us with this work, you can stop contributing at any time using the checkbox in the Tools/Preferences/Miscellaneous dialog.

3. Click on "Preferences" from the BlueJ drop down main menu (under Tools for Windows)


View on Mac
BlueJ
Project
Edit
Tools
View
Help
About BlueJ
Preferences... ⌘,
Services
Hide BlueJ ⌘H
Hide Others ⌥⌘H
Show All
Quit BlueJ ⌘Q
View on Windows
Project Edit Tools View Help
Compile Ctrl+K
Compile Selected Ctrl+Shift+K
Rebuild Package
Reset Java Virtual Machine Ctrl+Shift+R
Use Library Class... Ctrl+L
Project Documentation Ctrl+J
Preferences... Ctrl+Comma
Use the Project menu to create or open a project
BlueJ 3.1.4

4. Click on the "Miscellaneous" Tab



Documentation settings

JDK documentation URL http://download.oracle.com/javase/8/docs/api/

☑ Use this URL when generating project documentation

If this URL is used for generating documentation, it must be correct and accessible.
Otherwise the documentation generation will fail. To work offline, uncheck this option.

Compiler and runtime settings

☑ Show compiler warnings when unsafe collections are used

Research Participation

You will allow anonymous data collection research next time BlueJ is started

You only need to fill in the information below if asked to by a researcher:

Experiment Identifier

Participant Identifier

Documentation settings

JDK documentation URL http://download.oracle.com/javase/8/docs/api/

☑ Use this URL when generating project documentation

If this URL is used for generating documentation, it must be correct and accessible.
Otherwise the documentation generation will fail. To work offline, uncheck this option.

Compiler and runtime settings

☑ Show compiler warnings when unsafe collections are used

Research Participation

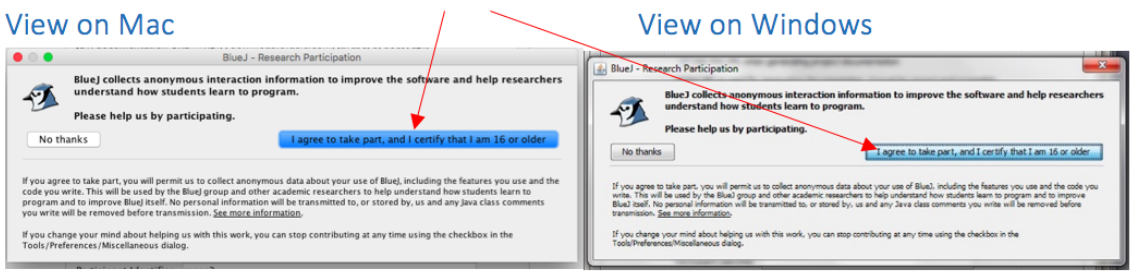
You are currently allowing anonymous research data collection

You only need to fill in the information below if asked to by a researcher:

Experiment Identifier

Participant Identifier

5. Skip this step if already prompted in step 2, else click on the "Change" button and then click on "I agree to take part, and I certify that I am 16 or older" button


View on Mac
View on Windows
BlueJ - Research Participation
No thanks
I agree to take part, and I certify that I am 16 or older

BlueJ collects anonymous interaction information to improve the software and help researchers understand how students learn to program.

Please help us by participating.

If you agree to take part, you will permit us to collect anonymous data about your use of BlueJ, including the features you use and the code you write. This will be used by the BlueJ group and other academic researchers to help understand how students learn to program and to improve BlueJ itself. No personal information will be transmitted to, or stored by, us and any Java class comments you write will be removed before transmission. See more information.

If you change your mind about helping us with this work, you can stop contributing at any time using the checkbox in the Tools/Preferences/Miscellaneous dialog.

BlueJ collects anonymous interaction information to improve the software and help researchers understand how students learn to program.

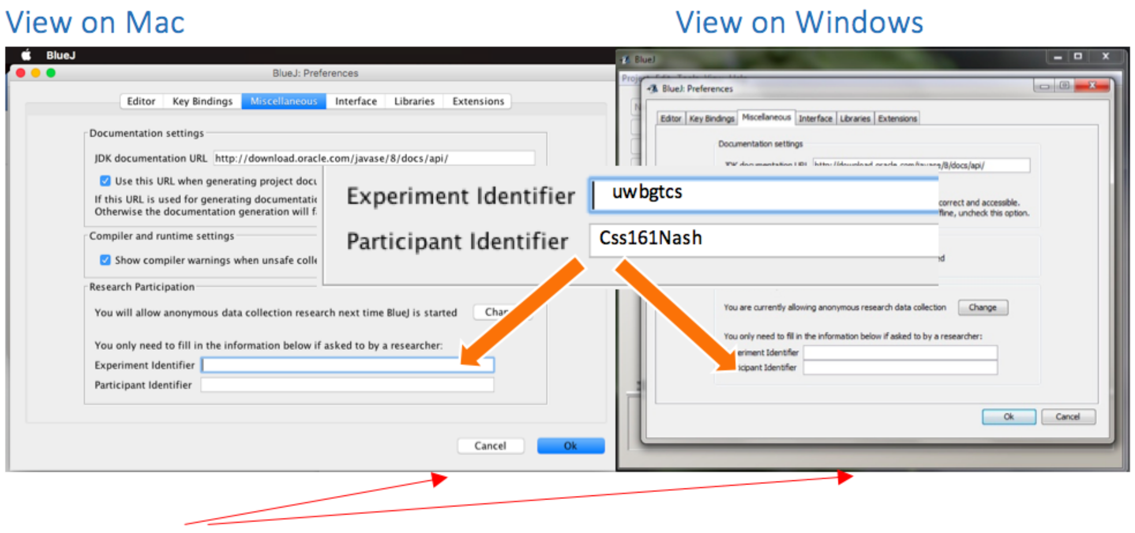
Please help us by participating.

If you agree to take part, you will permit us to collect anonymous data about your use of BlueJ, including the features you use and the code you write. This will be used by the BlueJ group and other academic researchers to help understand how students learn to program and to improve BlueJ itself. No personal information will be transmitted to, or stored by, us and any Java class comments you write will be removed before transmission. See more information.

If you change your mind about helping us with this work, you can stop contributing at any time using the checkbox in the Tools/Preferences/Miscellaneous dialog.

6. Enter "uwbgtcs" into "Experiment Identifier"

7. Enter "course + instructor LAST name" into "Participant Identifier", e.g. "css161Nash"



Documentation settings

JDK documentation URL http://download.oracle.com/javase/8/docs/api/

☑ Use this URL when generating project docu

If this URL is used for generating documentatio
Otherwise the documentation generation will f

Compiler and runtime settings

☑ Show compiler warnings when unsafe coll

Research Participation

You will allow anonymous data collection research next time BlueJ is started

You are currently allowing anonymous research data collection

You only need to fill in the information below if asked to by a researcher:

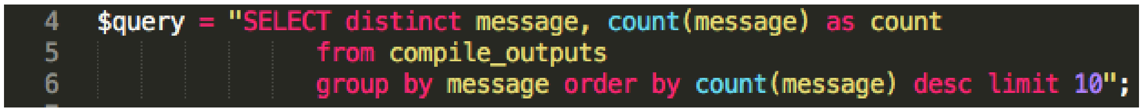
Experiment Identifier uwbgtcs

Participant Identifier Css161Nash

8. Click "Ok"

A.2 Queries for Research Questions

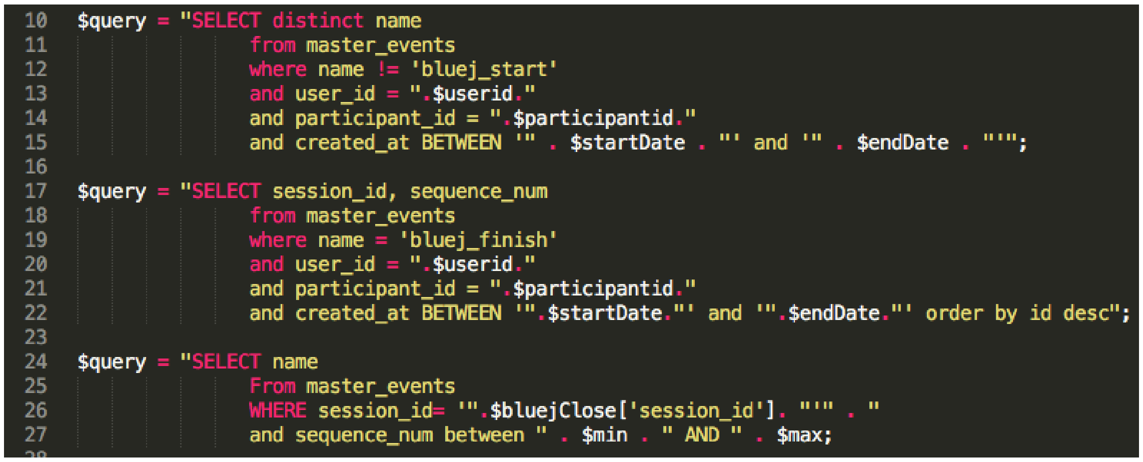
Top Ten Compiler Errors



```
$query = "SELECT distinct message, count(message) as count
              from compile_outputs
              group by message order by count(message) desc limit 10";
```

Figure A.1: Main Query for Answering Top Ten Compiler Errors

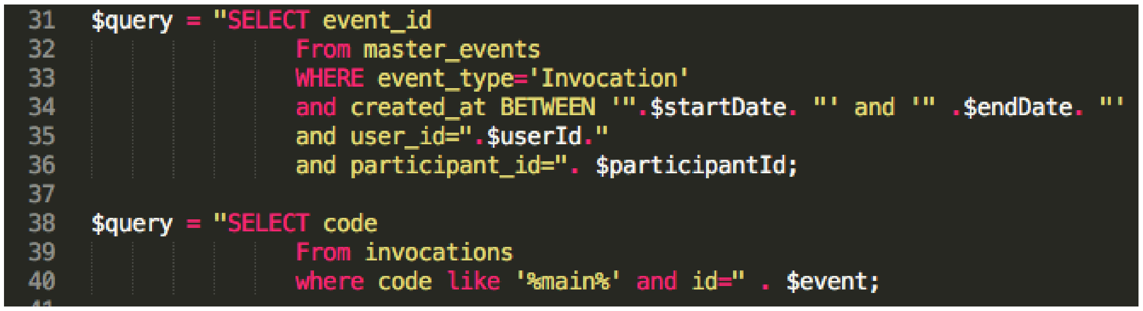
Last Few Events Before Closing BlueJ



```
$query = "SELECT distinct name
              from master_events
              where name != 'bluej_start'
              and user_id = ".$userid."
              and participant_id = ".$participantid."
              and created_at BETWEEN '" . $startDate . "' and '" . $endDate . "'";

$query = "SELECT session_id, sequence_num
              from master_events
              where name = 'bluej_finish'
              and user_id = ".$userid."
              and participant_id = ".$participantid."
              and created_at BETWEEN '".$startDate."' and '".$endDate."' order by id desc";

$query = "SELECT name
              From master_events
              WHERE session_id= '".$bluejClose['session_id']. "'" . "
              and sequence_num between " . $min . " AND " . $max;
```

Figure A.2: Main Query for Answering Last Few Events Before BlueJ Closed

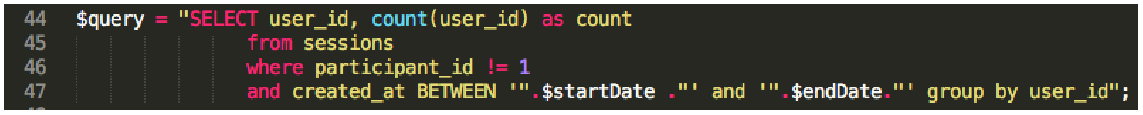
Number of Game Calls



```
$query = "SELECT event_id
                From master_events
                WHERE event_type='Invocation'
                and created_at BETWEEN '".$startDate. "' and '" .$endDate. "'
                and user_id=".$userId."
                and participant_id=". $participantId;

$query = "SELECT code
                From invocations
                where code like '%main%' and id=" . $event;
```

Figure A.3: Main Query for Answering Number of Game Calls Per User

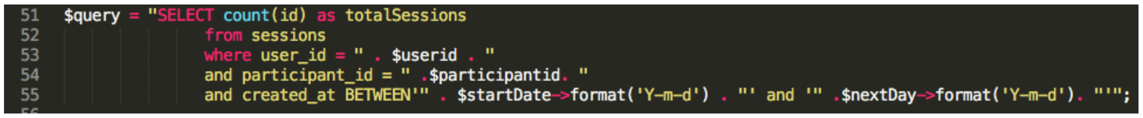
Total Number of Sessions Per User



```
$query = "SELECT user_id, count(user_id) as count
                from sessions
                where participant_id != 1
                and created_at BETWEEN '".$startDate ."' and '".$endDate."' group by user_id";
```

Figure A.4: Main Query for Answering Number of Sessions Per User

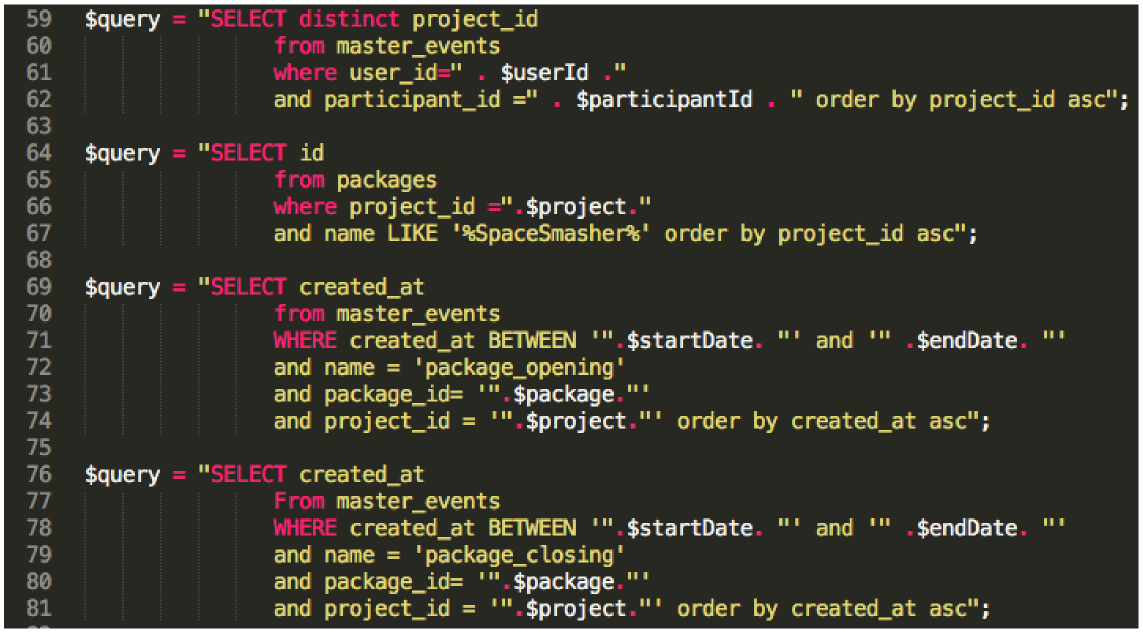
Occurrence of Sessions



```
$query = "SELECT count(id) as totalSessions
                from sessions
                where user_id = " . $userid . "
                and participant_id = " .$participantid. "
                and created_at BETWEEN'" . $startDate->format('Y-m-d') . "' and '" .$nextDay->format('Y-m-d'). "'";
```

Figure A.5: Main Query for Answering Occurrence of Sessions Per User

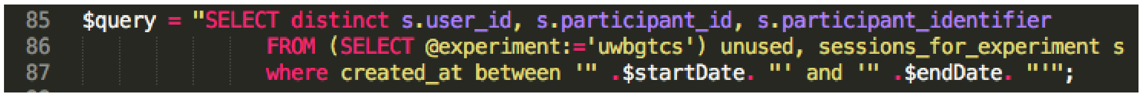
Duration Spent In SpaceSmasherAPI



```
$query = "SELECT distinct project_id
                from master_events
                where user_id=" . $userId ."
                and participant_id =" . $participantId . " order by project_id asc";

$query = "SELECT id
                from packages
                where project_id =".$project."
                and name LIKE '%SpaceSmasher%' order by project_id asc";

$query = "SELECT created_at
                from master_events
                WHERE created_at BETWEEN '".$startDate. "' and '" .$endDate. "'
                and name = 'package_opening'
                and package_id= '".$package."'
                and project_id = '".$project."' order by created_at asc";

$query = "SELECT created_at
                From master_events
                WHERE created_at BETWEEN '".$startDate. "' and '" .$endDate. "'
                and name = 'package_closing'
                and package_id= '".$package."'
                and project_id = '".$project."' order by created_at asc";
```

Figure A.6: Main Query for Answering Duration Spent in SpaceSmasherAPI

Participation Rate Per Instructor



```
$query = "SELECT distinct s.user_id, s.participant_id, s.participant_identifier
                FROM (SELECT @experiment:='uwbgtcs') unused, sessions_for_experiment s
                where created_at between '" .$startDate. "' and '" .$endDate. "'";
```

Figure A.7: Main Query for Answering Participation Rate Per Instructor

A.3 Requirements to run scripts

All instruction below are for setting up the project in MAC OS.

Source Code & Repository

The source code for the PHP scripts can be found here with other helpful information.

- https://bitbucket.org/hccjerrychen/assessingstudentengagement

Operating System

The PHP scripts mainly runs on MAC OS, but running on Windows is theoretically possible though not tested.

Required modules

- PHP 7.0

  Uses latest PHP 7.0 on MAC and was written using *Sublime 2* text editor. To install PHP on MAC using *brew*, follow the next steps.

  1. Open terminal
  2. Install *brew*, if it is not installed, by using the enter the follow command in the terminal
  3. /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/ins
  4. Install PHP by enter the following command once *brew* has been installed
  5. brew install php70

- MySQL Local Storage

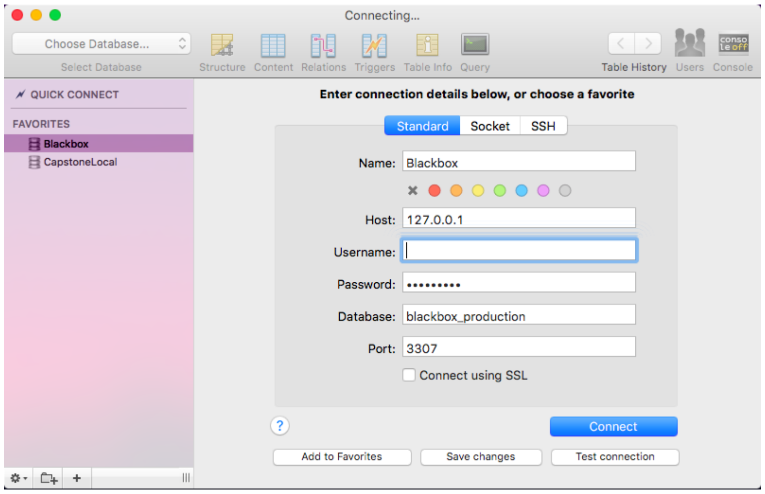
  Uses Ver 14.14 Distrib 5.7.9, for osx10.9 (x86_64) for MAC OS and was installed using the guide from the following website. MySQL is required for hosting a local database to store our research data.

https://coolestguidesontheplanet.com/get-apache-mysql-php-and-phpmyadmin-working-on-osx-10-11-el-capitan/

- Database Structure Setup

  The white machine runs MySQL on the standard port (3306) on the localhost interface. Once a SSH tunnel has been established, you can access Blackbox through terminal or via database management software, the later is demonstrated in the following. The MySQL login iformation will be provided separately when you request for an account on the machine.

1. Open terminal
2. Enter the following command to setup SSH tunnel
3. ssh -L 3307:localhost:3306 ssh_username@white.kent.ac.uk
4. Open Sequel Pro
5. Enter the information as shown in A.8 then click Connect



Enter connection details below, or choose a favorite

Name: Blackbox

Host: 127.0.0.1

Username:

Password: •••••••••

Database: blackbox_production

Port: 3307

☐ Connect using SSL

Figure A.8: Information to setup sequelPro

6. Select *Export* under *File* menu as shown in Fig. A.9

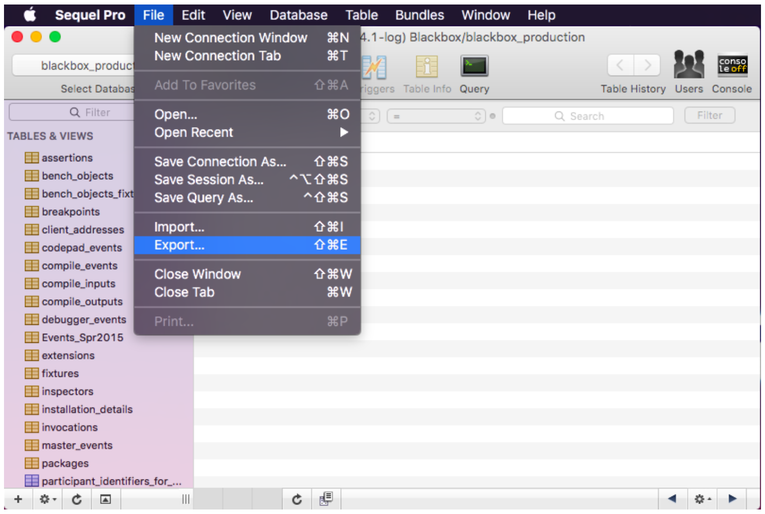
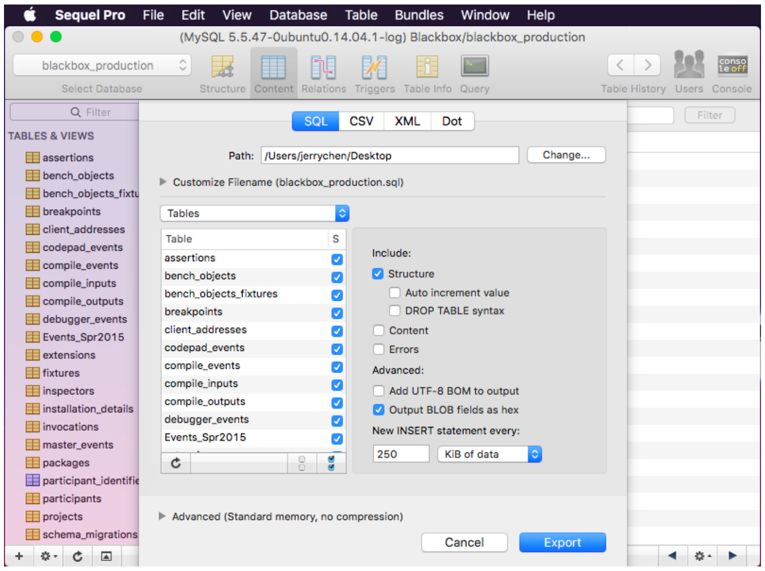


Figure A.9: Export option in Sequel Pro

7. Set the export detail as Fig. A.10 and click *Export*. A warning of no authorization to these tables are normal: *participant_identifiers_for_experiment*, *sessions_for_experiment*, and *sessions_for_experiment_participant*



Path: /Users/jerrychen/Desktop

▶ Customize Filename (blackbox_production.sql)

Tables

Table	S
assertions	✓
bench_objects	✓
bench_objects_fixtures	✓
breakpoints	✓
client_addresses	✓
codepad_events	✓
compile_events	✓
compile_inputs	✓
compile_outputs	✓
debugger_events	✓
Events_Spr2015	✓

Include:
- [✓] Structure
  - [ ] Auto increment value
  - [ ] DROP TABLE syntax
- [ ] Content
- [ ] Errors

Advanced:
- [ ] Add UTF-8 BOM to output
- [✓] Output BLOB fields as hex

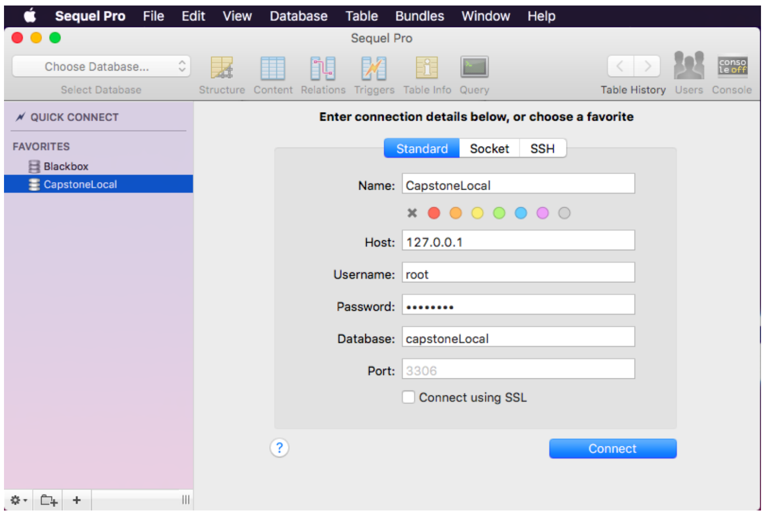
New INSERT statement every: 250 KiB of data

▶ Advanced (Standard memory, no compression)

Figure A.10: Export details

8. Open a new tab in Sequel Pro by pressing *command* + *T* for a new connection view.

9. Setup local database connection by entering the information as shown in Fig. A.11 and click *Connect*. Use your updated *Username* and *Password*, otherwise Username is *root* and password should be *empty* by default.



Enter connection details below, or choose a favorite

Name: CapstoneLocal
Host: 127.0.0.1
Username: root
Password: ••••••••
Database: capstoneLocal
Port: 3306
Connect using SSL

Figure A.11: Local Database Connection Information

10. Click *Add Database* from the dropdown menu below *File* A.12

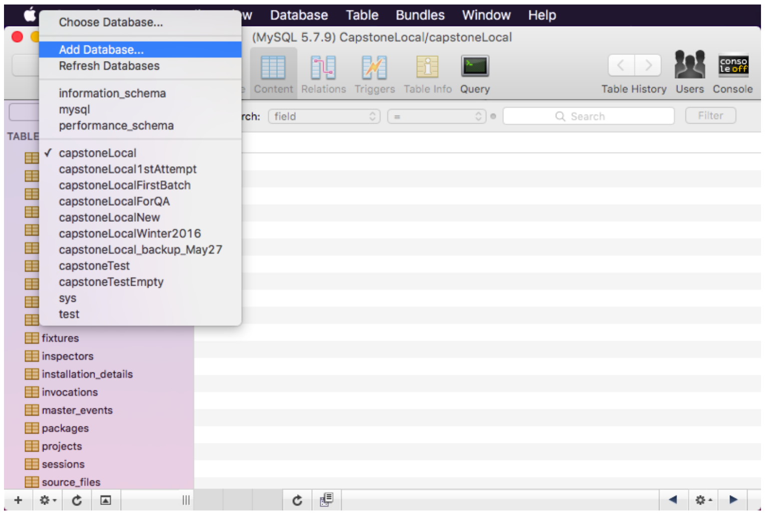


Figure A.12: Local Database Connection Information

11. Select *Import* from *File* menu A.13

12. Select the exported file generated from step 7 and click *Open* A.14

13. After import has completed, the left hand side of Sequel Pro should be populated

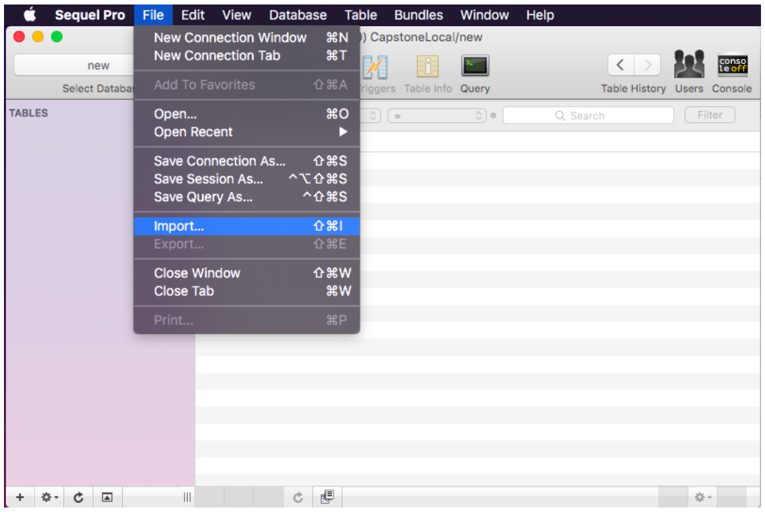
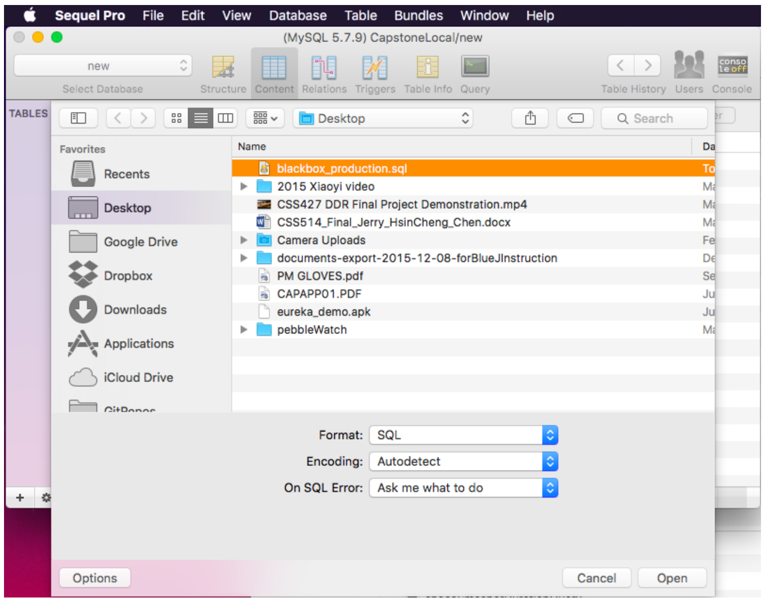


Figure A.13: Local Database Connection Information



Format: SQL
Encoding: Autodetect
On SQL Error: Ask me what to do

Figure A.14: Local Database Connection Information

with empty tables as shown in Fig. A.15

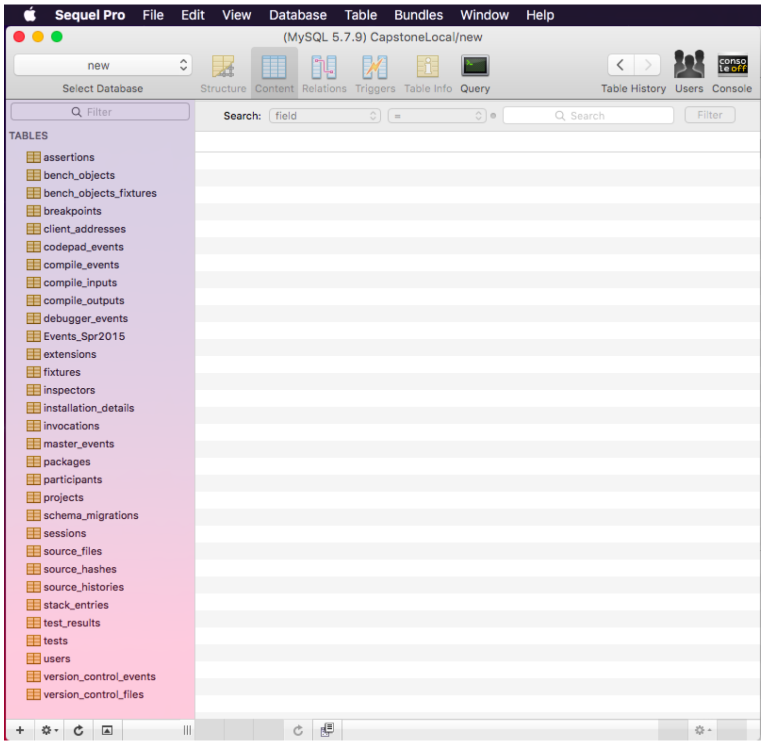


Figure A.15: Local Database Connection Information

- PHPUnit

  Uses PHPUnit 5.2.12 for unit testing.

  1. Open terminal
  2. Install *brew*, if it is not installed, by using the enter the follow command in the terminal
  3. /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/ins
  4. Install PHPUnit by enter the following command once *brew* has been installed
  5. brew install phpunit