Mining Student Time Management Patterns in Programming Projects[1]

Dr. Dale E. Parson and Allison Seidel
Kutztown University of Pennsylvania
Kutztown, PA, 19530, USA

Abstract

Computer science faculty members cite procrastination as one of the key causes of poor student performance in programming projects. In contrast, students cite conflicting demands for time. This study uses a tool-driven process of automated compilation and testing of student programs to collect student-project data. Data include when, for how long, how often, and with what magnitude of effort and accomplishment, students engage in work to complete programming assignments. Participation is voluntary, and data from auxiliary sources, including a questionnaire on conflicting demands on time, complement automatically collected data. Analyses reveal that procrastination and excessively brief work sessions are the main indicators of problems for students with inadequate prior success in earlier computer science courses. Some students with successful track records know when they can afford late starts and short sessions. The time of day that students work is a contributing factor to success. The goal is to build an automated warning system for at-risk students.

Keywords: data mining, programming assessment, student programming, time management.

1. Introduction

The project reported in this paper grew out of an initial offering of a master's level course in data mining using the Weka toolset [1,2] in conjunction with the annual offering of two sections of the Java Programming course in spring 2013. Graduate students offered advice on mining the project work habits of undergraduates, and Java students voluntarily supplied the initial data set. Java Programming at Kutztown University is an elective major course that includes sophomores, juniors and seniors after they complete the CS1-CS2 introductory course sequence using C++. The authors extended data collection to two sections of undergraduate Operating Systems in fall 2013, and to two sections of Java Programming and one section of Programming Languages in spring 2014. The range of courses and student experience levels help to distinguish consistent attributes from incidental ones in predicting student project success as a function of time management and other student work related data. None of the courses include in-class programming time. All project work takes place as homework.

The initial investigation grew out of an interest in quantifying the folklore on both sides of the faculty / student divide. Faculty members cite procrastination as a primary cause of poor results in student programming projects. Students cite conflicting demands on their time from projects and exams in other courses. While both factors contribute to project success to some degree, their contributions are far from simple and linear.

A goal is construction of an advisement program that would warn at-risk students when their patterns of work on programming projects begin to exhibit signs of problems. Anticipated use of this program by students would be voluntary, as is participation in the ongoing study itself.

The authors chose to report preliminary results before completion of the full study because we have uncovered useful information that we are communicating to students in current courses. Encouraging students to avoid pitfalls identified in preliminary results may help to uncover the effectiveness of passing this information along to students.

2. Related work

Edwards, et. al. have previously reported late starts in programming projects as clearly associated with poorer results on such projects [3]. That study utilized data from three programming courses for over five years, in contrast to the present preliminary study. It confirmed results from earlier studies about the correlation of earlier project starts with better rates of project success. Unlike the current project, that study eliminated both consistently well-performing students and consistently poorly-performing students from the analysis in order to focus on intra-student attributes that vary between successful and unsuccessful

[1] This study was supported in part by a Kutztown University Assessment Grant in 2013.

projects. The current study seeks to uncover consistent work habits leading to success and failure, regardless of individual student correlations. In fact, the present study is particularly interested in detecting patterns within at-risk students, including both consistently poorly-performing students and intermittently poorly-performing students. The cited study considered only the latter population.

Results of the present study agree with the results of the cited study, while detecting additional significant attributes of student work habits that alter those basic results.

Edwards and Ly have reported on automating analysis of the specific types of problems that occur in running student programs [4]. That analysis differs in nature from the current project, which focuses on correlations between student work patterns and project success or failure, as contrasted with specific types of project failure.

Mierle, et. al. examined student code repositories, based on file modifications that appear in file change submissions and log files [5]. They found a weak correlation between normalized number of lines of code per revision and student success in terms of final grade, and no correlation with timing of student work. The present study differs in nature by logging and analyzing student work at a much finer temporal grain, that of individual *make* actions within each student's private workspace. The present study uncovers more detailed correlations.

The most recent related study examined confirms the correlation of poor programming project results with late project starts [6]. That study allowed student submission of programs to additional, opaque automated tests (so-called *release tokens*). It concluded that availability of these tests might discourage students from writing their own test cases, and might encourage procrastination because students can count on additional available tests no matter how late they start. The current project takes a different approach to testing, modeled after the industrial experience of the instructor. The instructor supplies test cases, requires students to write additional test cases for some projects, and uses additional test cases not available to the students for grading. The latter sets of tests emulate customer *acceptance testing* not available to software providers. The present study collects data on successful and unsuccessful test runs for instructor-supplied and student-required testing, but it does not use a limited number of opaque test runs as a variable. The current study finds several attributes that correlate with project success, in addition to start time.

3. Data Collection & Extraction

3.1 Using *make* for attribute collection

Most of the programming projects studied use the GNU *make* utility [7] on a Unix server for compilation, testing, and submission of student programs. A student types *make test* in a project directory, leading to automatic compilation when object or executable files predate their corresponding source files, followed by automated tests supplied by the instructor, and then by automated tests written by students when required. Automated submission of completed programs takes place via the *make turnitin* command. For projects where students offload code from the Unix server to laptops for development of graphical user interfaces (GUIs), the instructor supplies an executable Java archive that serves as an alternative, project-specific makefile.

Using either mechanism, the makefile captures and compresses data from the project directory and archives it for later analysis. Analysis defers until after course completion and grading as part of the agreement between the instructor and students. Data collected for the study play no role in grading. A student earns one bonus point on a scale of 100 project points for participating in the automated study, and an additional point for completing a short survey explained in the next section. Students can also earn these two points by performing actions that remove their data from the study. Institutional ethical standards preclude giving bonus point incentives only to participating students. Participation is voluntary, and students must not feel grade pressure to participate.

Listing 1 shows the first category of collected data, the actual zip archives containing detailed student information. An archive file name encodes the student identifier (obscured here), the date and time of the triggering *make* action, and an identifier for that action. The first two BUILD lines of Listing 1 show two failed attempts to compile the assignment, followed by a BUILD leading to a BUILT record, signifying successful compilation. The TESTING line signifies the start of automated tests, and the TESTED line signifies successful execution of tests. Typically, the student triggers compilation, and testing when compilation succeeds, by invoking *make test*, which triggers *make build* for updated source files. It is possible to determine the time and frequency of a student's attempts at compilation and testing, and the success of these attempts, simply by decoding the archive file names. Data logging activities remain invisible in the interest of minimizing impact on student workflow.

```
idN_2014-02-12-12-37-43-EST_BUILD.zip
idN_2014-02-12-12-38-48-EST_BUILD.zip
idN_2014-02-12-12-41-50-EST_BUILD.zip
idN_2014-02-12-12-41-53-EST_BUILT.zip
idN_2014-02-12-12-41-53-EST_TESTING.zip
idN_2014-02-12-12-41-55-EST_TESTED.zip
```

Listing 1: Archives of student *make* data

Each archive contains two files. Listing 2 shows an example Unix listing file extracted from an archive. It is a listing of the student's project directory at the time of invoking make. Each line shows student ID, the number of bytes in the file, the file's most recent modification date

and time, and the file's name. By comparing file sizes and modification timestamps from successive archives, it is possible to determine which files have changed, and the net change in byte size for those files, since the previous make action for that student project.

The other file extracted from each archive of Listing 1

```
TESTED 2013-03-24-21-45-38-EDT
~idN/JavaLang/FillWord2
idN     5836 Mar  3 15:43      FillWordTest.java
idN     57 Mar  3 16:01        testjava.txt
idN     6248 Mar  3 16:01      testjava.ref
idN     70 Mar  3 16:06        testjava2.txt
idN     2037 Mar  3 16:07      makefile
idN     8951 Mar  3 16:07      testjava2.ref
idN     1183 Mar 13 17:33      IFillWord.java
idN     3474 Mar 14 12:59 FillWordGrows.java
idN     8849 Mar 24 21:33 FillWordBasic.java
idN     5027 Mar 24 21:37 FillWordHelper.java
```

Listing 2: Listing of student project directory

contains the contents of multiple text files. Each logged make action concatenates all text files of interest in the project, in this case Java source files, into a single file with markers giving start of each file, file name, logging time, and file contents. The initial use for collecting source files has been to use the Unix *diff* utility to determine number of lines *added*, *changed*, and *deleted* for each source file since the previous make action. These files could also support analysis of types of student solution and error mechanisms in a subsequent study.

In summary, archiving collects all available project data for a student every time a student compiles, tests, or turns in a project, including the success or failure of compilation and testing. Since compilation and testing are automated, it is possible to determine, automatically, exactly which files fail compilation and which tests fail testing. For the current study the focus is on time management patterns, the magnitude of changes per work session, time of day of each session, and related data discussed in the next section.

3.2 Collection of auxiliary attributes

There is no means for automatically collecting data about conflicting demands on student time. The study uses a survey with three questions: How many computer science projects from other courses were given out during the project period? How many computer science projects from other courses were due during the project period? How many exams from any course took place during the project period? This study does not measure non-course time conflicts such as jobs or extracurricular activities.

Since responses are subjective and accuracy of the answers is less reliable than the makefile-collected data of the previous section, the study treats survey answers as student perceptions of these potential sources of time conflicts. Correlation of student perceptions with project success is a measure that the study can inspect objectively.

Student data from other sources include *year* (freshman, sophomore, etc.), *track* (the department has software development and information technology tracks for students), *course*, *semester*, *project number*, and *project start* and *end dates and times*. The instructor collects *student email questions* about projects, classifying them into the following categories: 1) informed and detailed, 2) uninformed and vague, and 3) the total of (1) and (2). Email provides one sample measure of student interaction with the instructor. This study does not measure classroom attendance or office hour meetings with the instructor.

So-called *target attributes* include project numeric and letter grade, final course numeric and letter grade, and centile ranking of project and course numeric grades. The latter measures are useful in distributing clumped grades during analysis. A script extracts these data from a grading spreadsheet after the semester ends.

3.3 Student cumulative grade data

The initial plan was not to use cumulative student grade data coming into the course. The intent was to base analysis strictly on conditions and actions relating directly to the projects. However, initial analysis determined that some students could perform well with seemingly bad works habits, while others could not. That fact may account for some of the lack of correlation between project start time and project success in the study by Mierle, et. al. [5]. Consequently, this study incorporates four additional attributes into the dataset: 1) cumulative grade point average in computer science courses, 2) number of credits in computer science courses, 3) overall cumulative grade point average for all courses, and 4) total credits earned. These four attributes are recorded at the start of the course. Analysis reveals that including these attributes helps to sort out students who can afford to use what might be considered "bad habits" from those who cannot. These attributes are helpful in identifying potentially at-risk students who could subscribe to an automated warning system derived from this study.

4. Data Analysis

4.1 Distinguishing attributes

The study uses 90 attributes concerning student project activities and related data discussed in the previous section. These attributes fit into the following categories.

1. **Student data** include ID, major track, and year in university.
2. **Course-project data** include course number, semester, project number, and project start and end date-time.

3. **Date-times for start and completion of student work** come in five forms: A) hours from handout until student start, B) hours from completion until project due, C) hours from student start until project due, D) hours from handout until student completion, and E) hours from student start until completion.
4. Other time management attributes relate to the concept of a *work session*, which for this study is a contiguous period of captured make actions with no intervals between actions >= 60 minutes in length. Attributes record the min, max, mean, sample standard deviation, median and mode for the following data: **length of session time** in minutes, and **time between sessions** in hours.
5. Additional temporal attributes are **total session time**, total **number of sessions**, and **session time-of-day**. Graduate students suggested the latter data category, which comes in six divisions: sessions centered between 12 and 3:59 AM, between 4 and 7:59 AM, between 8 and 11:59 AM, and their three afternoon-evening counterparts.
6. The study also analyzes size of work (min, max, mean, sample standard deviation, median and mode) for the following per-session data attributes: **number of source file bytes changed**, **number of source files modified**, number of source **lines added**, **lines deleted**, and **lines changed**.
7. Informed and detailed **email messages** to the instructor, uninformed and vague emails, and the total of these two classes comprise another class of data as previously discussed.
8. The *target attributes* of project numeric and letter grade, final course numeric and letter grade, and centile ranking of project and course numeric grades constitute the final category.

Analysis uses only one target attribute at a time because these attributes are partially redundant. Project numeric grade trivially determines project letter grade, for example, without considering other attributes. After some analysis the study dropped use of overall course grades because these course grades are a weighted average of course projects and exams, and some students are habitually poor test takers. Project preparation patterns showed no clear correlation to exam results. The study also dropped use of letter grades in favor of using binned (discretized) numeric grades in order to allow exploration of binning strategies. The primary target attribute is *project numeric grade*, with *project centile ranking* also considered in detail.

Data mining has two different modes of operation for investigators. The first is to *make data relationships clearer to human investigators*. The second is to support creation of *automated programs for pattern recognition and response*. The former is the mode of the current study. The latter is the mode for the anticipated automated early warning system for at-risk students.

The study used two complementary approaches to thin the 84 non-target attributes down to a set that exhibits correlation with target attributes in a way that makes the data relationships clearer to human investigators. The first was repeated manual application of Weka's OneR rule-based machine learning algorithm [1,2] that uses the minimum-error attribute for prediction of discrete numeric attribute bins. OneR selects the single most accurate non-target attribute to predict the target attribute. Our process was to use OneR repeatedly to select the next-most predictive attribute, then remove that attribute, and then repeat the process. This iterative process also eliminated partially redundant attributes, along with attributes for which there were significant numbers of missing values. For example, the initial study of the Java course did not capture source files for GUI development projects run on laptops, so it was necessary to discard diff-based line add/change/delete attributes when analyzing those projects.

The second approach was to use Weka's "Select attributes" capability, applying the "CfsSubsetEval" algorithm that evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them, in conjunction with a "BestFirst" search method that searches the space of attribute subsets by greedy hill climbing augmented with a backtracking facility. This is essentially the automated version of the semi-manual approach of the previous paragraph. Both yielded the same set of predictive attributes.

Jstr hours from student start until due deadline
Mavg average minutes of a work session
Mdev sample standard deviation of Mavg
Snum number of work sessions
Mtot total minutes spent on the project
Cgpa computer science GPA at semester start
GprjRank centile ranking of project grade

If Jstr < 24.0 then 0 <= GprjRank <= 13.2
Elseif Jstr < 79.0 then GprjRank > 90.4
Elseif Jstr < 181.0 then 80.7 < GprjRank <= 90.4
Elseif Jstr >= 181.0 then GprjRank > 90.4
(36/111, 32.4%, instances correct)

Listing 3: OneR prediction from 6 attributes

Listing 3 shows Weka results for six of the most predictive attributes in predicting the centile ranking of the project grade. OneR shows that **Jstr**, the number of hours from the time the student started the project until the project deadline, is the single most predictive attribute. Other analyses duplicate this result. **Jstr** is more useful than the time from project handout until start, because project time periods vary by project, sometimes because of complexity, but also because of incidental reasons such as spring break or an exam in the course. **Jstr** is a measure of

Gprj project grade
If Cgpa < 2.24 then 0 <= Gprj <= 0.625
Elseif Cgpa < 2.635 then 0.625 < Gprj <= 0.815
Elseif Cgpa < 2.945 then 0.815 < Gprj <= 0.925
Elseif Cgpa >= 2.945 then 1.015 < Gprj <= 1.04
(28/111, 25.2% instances correct)

Listing 4: OneR prediction to project grade

procrastination because it shows how close to the project deadline a student has waited before starting.

Listing 3 shows that starting a project within the last 24 hours leads to the lowest centile bin, between 0 and 13.2 percentile. This observation accords with previous studies such as Edwards, et. al. [3], which states, "approximately two-thirds of the lower scores were received by individuals who started on the last day or later." However, the fact that OneR's predictions based on **Jstr** are correct only 32.4% of the time, and that one-third of the lower scores of [3] were received by individuals who started earlier than the last day, show that procrastination is not the only important attribute at work in determining project quality.

Listing 3 shows a problem with a simple monotonic interpretation of start time before deadline as the main attribute of interest. OneR predicts that students starting between 24 and 79 hours before the due date will attain the top centile bin (greater than 90.4 percentile), 79-to-181 hour starters will attain the next lower centile bin, and students starting >= 181 hours will attain the top bin. The problem is with the 24-79 group attaining the top bin. This result resonates with the instructor's experience that some students can start during the last few days, sometimes send very focused email questions that may not receive immediate responses, send subsequent "never mind, I found it" messages, and complete the project with good results. It is because of such students that the authors decided to include the computer science GPA and related attributes in the study. In fact, replacing centile ranking with project grade as the target attribute leads to the OneR rule of Listing 4, with computer science GPA (**Cgpa**) replacing **Jstr** as the most predictive attribute.

This fatalistic prediction of prior GPA as the most predictive attribute would be discouraging, were it not for the fact that this OneR rule is wrong 74.8% of the time for the Java Programming dataset. **Cgpa** is important, but its contribution is not deterministic. The next section takes up the detailed analysis used to find countervailing attributes.

4.2 Patterns for indication of success

The previous section deals with the search through available attributes. Lack of space precludes inclusion of strategies for the search through the space of available machine learning algorithms and correlation techniques. The guiding principle in this search was intelligibility. Algorithms and tools whose results are straightforward to interpret lead to the results reported in this section.

Table 1 shows the results of Simple K-means clustering [2] for 6 clusters, along with the full programming dataset, for the six most predictive attributes for project grade **Gprj** in Java Programming. Several relationships in the data appear at the surface.

First, the lowest average project grade of 57.19% in cluster 4 pairs with the latest starting time **Jstr** of 36.1875 hours before the project deadline. Cluster 4 is the smoking gun for the negative effects of procrastination.

However, cluster 0 with the second smallest **Jstr** of 57.875 pairs with a grade of 93.83%. Comparing these two clusters shows that the computer science GPA for cluster 0 is 3.3392 compared to 2.255 for cluster 4, suggesting that the cluster 0 students are better prepared for a somewhat late start; cluster 0's **Jstr** value is 60% greater than cluster 4's. There are other important differences. Cluster 0's average work session time in minutes **Mavg** is 68.5097, compared to cluster 4's 36.6927, and cluster 0's total session minutes **Mtot** is 174.125, compared to 4's **Mtot** of 100.6875. Cluster 0 students work in sessions that are 87% longer, and for overall time that is 75% greater, than cluster 4 students.

Cluster 5 with the second lowest **Gprj** value at 88.86% is significantly better than cluster 4 at 57.19%. The **Cgpa** at 2.6143 is also the second lowest, and **Mavg** is relatively low at 49.4403 minutes. However, cluster 5 students have a

attribute	full data 111 records	cluster 0 24 = 22%	cluster 1 9 = 8%	cluster 2 27 = 24%	cluster 3 7 = 6%	cluster 4 16 = 14%	cluster 5 28 = 25%
Jstr	167.9459	**57.875**	155.2222	249.7778	238	**36.1875**	245.25
Mavg	58.1449	**68.5097**	138.8896	42.4935	63.0163	**36.6927**	49.4403
Mdev	48.4055	36.8001	116.7501	44.7172	78.8199	21.9392	47.4618
Yavg	5153.661	4435.4986	20419.4735	2605.7539	5061.4645	5635.8594	3066.7782
Snum	5.4324	2.9167	4.4444	5.7407	14.1429	2.625	7.0357
Mtot	302.045	**174.125**	589.7778	242.2963	884.8571	**100.6875**	346.1786
Cgpa	3.055	**3.3392**	3.3411	3.6156	3.1429	**2.255**	2.6143
Gprj	90.01%	**93.83%**	99.56%	100.89%	102.29%	**57.19%**	88.86%

Table 1: Simple K-means clusters for the six most predictive attributes for project grade **Gprj** in Java Programming

high **Jstr** value of 245.25 hours, the second-highest number of sessions **Snum** of 7.0357, and a higher than average total session minutes **Mtot** of 346.1786. An early start and repeated application of time to the project pay off in giving cluster 5 students a **Gprj** that is 55% greater than that of cluster 4.

Finally, Table 1 shows that for most clusters, larger values for mean session minutes **Mavg** correlate with larger values for mean source bytes (characters) modified per session **Yavg**, but for cluster 4, the second highest **Yavg** of 5635.8594 bytes pairs with the smallest **Mavg** value. Cluster 4 students attempted, unsuccessfully, to complete a large amount of work in very little time.

Other algorithms such as the J48 decision tree and Bayesian inference [2] give results that are compatible with the K-means clusters, but that are harder to read and integrate into a paper because of the amount of detail in their logic. One additional example is the Pruned **M5P** model tree, which uses decision tree-like structure to select from among a set of linear regression formulas. Below is the M5P model tree for all Table 1 attributes except **Yavg**.

Cgpa <= 3.205 :
| Jstr <= 66 :
| | Mavg <= 36.167 : **LM1** (5/29.636%)
| | Mavg > 36.167 : **LM2** (18/106.448%)
| Jstr > 66 : **LM3** (40/57.247%)
Cgpa > 3.205 : **LM4** (48/31.383%)
LM num 1: Gprj =
0.0012 * Jstr + 0.0009 * Mavg
+ 0.0214 * Snum - 0.0002 * Mtot
+ 0.2908 * Cgpa - 0.2251
LM num 2: Gprj =
0.0001 * Jstr + 0.0009 * Mavg
+ 0.0214 * Snum - 0.0002 * Mtot
+ 0.2278 * Cgpa + 0.0515
LM num 3: Gprj =
0.0001 * Jstr + 0.0007 * Mavg
+ 0.028 * Snum - 0.0001 * Mtot
+ 0.226 * Cgpa + 0.0893
LM num 4: Gprj =
0.0008 * Mavg + 0.0143 * Snum
- 0.0001 * Mtot + 0.1052 * Cgpa + 0.5199

Listing 5: Pruned M5P model tree

LM4: When the tree finds a **Cgpa** > 3.205, it goes to a linear formula with the highest constant value for the **Gprj** (51.99%) and the lowest weight for the **Cgpa**. These students are not typically at risk.

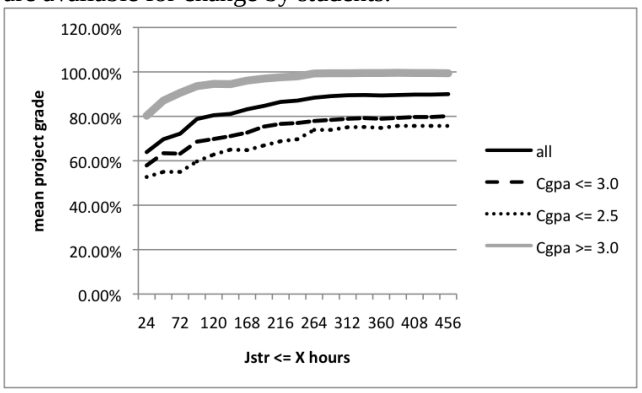
LM3: Otherwise, when **Cgpa** <= 3.205 and the start time **Jstr** is greater than 66 hours before the deadline, the tree goes to a linear formula with a much smaller constant value and greater dependence on the **Cgpa**.

LM2: With **Jstr** <= 66 and **Mavg** work time that exceeds 36.167 minutes, the tree goes to a formula with more weight on the **Cgpa** and a lower constant value.

LM1: Finally, with **Mavg** <= 36.167, the tree goes to a formula with the greatest dependence on **Cgpa** and a negative constant value. A low mean average session time is an automatic handicap.

There is nothing a student can do to change an incoming **Cgpa**, but the tree and formulas of Listing 5 indicate that students with lower **Jstr** and **Mavg** values correlate with lower constant values and more deterministic ties to the **Cgpa** within linear predictors, than students with higher **Jstr** and **Mavg** values. At-risk students need to start earlier and engage in sessions that exceed 36 minutes. Apparently, 36 minutes are not enough to get properly immersed in software development. Other Weka models suggest that **Mavg** should exceed 60 to 75 minutes.

M5P is one of the better predictors applied to these attributes, predicting about 55% of the test cases correctly for this dataset, with a mean absolute error of 13.04% on a **Gprj** scale of 100%. Predicting incorrectly 45% of the time, with an average miss of more than a 10% letter grade, is not great prediction. A student is a hard nut to crack. There are, seemingly, hidden variables not measured by the study. However, **Cgpa**, **Jstr**, and **Mavg** have strong correlation with project grade, and the latter two attributes are available for change by students.

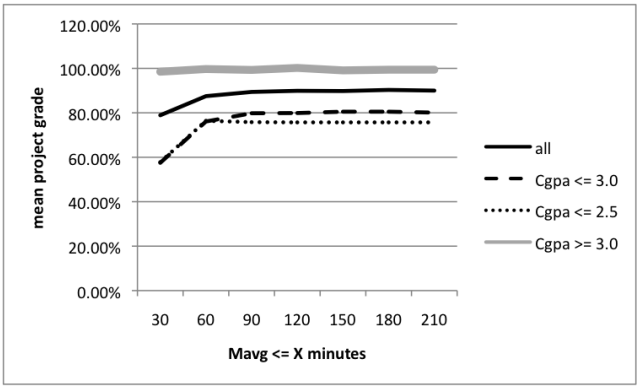


Graph 1: **Gprj** as a function of cumulative **Jstr** ranges

Graph 1 shows average relationships between **Jstr** value ranges and mean project grades **Gprj** for students in various **Cgpa** ranges. **Jstr** values to the right in the graph subsume **Jstr** values to the left. For example, 72 on the X axis means **Jstr** <= 72, a range that includes **Jstr** <= 24. The maximum grade of 127% in the dataset includes optional bonus points for one project. The authors computed Graph 1 by using the Weka preprocessor as a query tool to eliminate from the dataset **Cgpa** and **Jstr** values outside a desired range, and then entered the resulting mean **Gprj** value into an Excel spreadsheet to plot the graph. All four **Cgpa** curves show a minimum spread of 20%, i.e., two letter grades, across the range of

Jstr values. The graph cannot capture the nonlinear relationships of **Jstr** to **Cpga** illustrated by Table 1, but nevertheless it shows the overall importance of **Jstr**. All four curves level out at about 10 to 11 days. It is the instructor's experience that most students do not utilize project periods greater than two weeks in length very well, requiring the instructor to break big projects in a number of smaller assignments, each fitting in a two-week interval. Graph 1 confirms the validity of this practice.

Graph 2, also captured using Weka's value range filtering and Excel, shows related curves for the relationship of cumulative session minutes **Mavg** in relation to mean project grade **Gprj**. Work sessions less than 60 minutes in length lead to problems for most students, and sessions at least 75 minutes in length are better for at-risk students.



Graph 2: **Gprj** as a function of cumulative **Mavg** ranges

4.3 Secondary & problematic patterns

Weka's "Select attributes" capability suggests possible importance for work sessions with midpoints between 4 AM and 11:59 AM, i.e., morning sessions. Students in higher **Gprj** ranges did tend to work in the mornings, but they also worked more in the afternoons, took an evening break, and worked again during the four hours before midnight. At-risk students fell into two groups. One group started within the last 36 hours, leaving themselves no opportunity for time management. They did not work in the morning. The other group started well in advance of the deadline, performing most of their work during the four hours before midnight. Our conclusions are that some morning work after a night's rest helps to improve a project's quality, and that consistently working only at day's end, when a student is tired, leads to poor quality.

No other attributes correlate strongly with project success, including student survey data about conflicting demands on time. Also, the data collected in the fall 2013 Operating Systems classes were not useful because student projects consisted of modifying state machines drafted by the instructor that emphasized analysis and design, with very little coding, similar to solving a proof. Most of the work was in the pre-coding stage, and therefore not visible to data collection via make. Three spring 2014 programming courses will provide much more useful data.

5. Conclusions and Future Work

Procrastination is a culprit in poor results for programming projects, but it is not a simple, linear one. Some students who have performed well in past computer science courses apparently know their limits. However, for many students, and especially for at-risk students with lower computer science grade point averages at the start of a programming course, starting at least 11 days before the due date of a two-week project yields demonstrable benefits. Also, the minimum length of a programming work session should be at least 60 minutes, and preferably 75. Time of day of work is a contributing factor. In addition to starting early in the project cycle, students should not work solely at the end of long days, when they are tired.

There are additional data available for mining. Data collection in spring 2014 will more than double the size of the dataset. The M5P model tree algorithm promises to provide an accurate basis for constructing an early warning system for at-risk students. The plan is to implement a prototype system after completing analysis in summer 2014, with voluntary use by students to follow.

References

[1] Machine Learning Group at the University of Waikato, "Weka 3: Data Mining Software in Java", http://www.cs.waikato.ac.nz/ml/weka/, May 2014.

[2] Witten, Frank and Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Third Edition, Morgan Kaufmann, 2011.

[3] Edwards, Snyder, Pérez-Quiñones, Allevato, Kim and Tretola, "Comparing Effective and Ineffective Behaviors of Student Programmers", *Proceedings of ICER '09: International Computing Education Research Workshop*, Berkeley, CA, August 2009.

[4] Edwards and Ly, "Mining the Data in Programming Assignments for Educational Research", *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications* (EISTA'07), Orlando, FL, July 12-15, 2007.

[5] Mierle, Laven, Roweis and Wilson, "Mining Student CVS Repositories for Performance Indicators", *Proceedings of the 2005 International Workshop on Mining Software Repositories*, St. Louis, May, 2005.

[6] Spacco, Fossati, Stamper and Rivers, "Towards Improving Programming Habits to Create Better Computer Science Course Outcomes", *Proceedings of ITiCSE 2013, the 18th Annual Conference on Innovation and Technology in Computer Science Education*, Canterbury, UK, July 1-3, 2013.

[7] Free Software Foundation, *GNU Make* home page, http://www.gnu.org/software/make/, May 2014.