DATA MINING TEMPORAL WORK PATTERNS OF PROGRAMMING STUDENT POPULATIONS

Dale E. Parson, Lori Bogumil, Allison Seidel
Department of Computer Science and Information Technology, Kutztown University of PA
parson@kutztown.edu

ABSTRACT

This paper reports the second stage of a study of the correlations between the temporal work patterns of computer programming students and their success or failure as measured by programming project assignment grades and related metrics. The first stage confirmed the importance for most students of getting an early start on a programming project, and it also uncovered the fact that some student groups perform well with late starts, suggesting the likelihood that they engage in the productive practice of *active procrastination*. The second most important factor for success is the average length of assignment work sessions. Session lengths from 60 to 120 minutes appear to be optimal for most students. Other contributing factors include total time spent on a project and working more day than night sessions. This second stage more than doubles the amount of data collected and analyzed. It finds that procrastination and session length remain prominent, while secondary factors become slightly less prominent. Its primary contribution is the analysis of within-student patterns for students who perform significantly better on some assignments than others, finding that for these students, starting early and maintaining appropriate work session lengths and times of day correlate with better performance.

KEY WORDS
data mining, programming assessment, student programming, time management.

1. Introduction

The work reported here is the second stage of a study begun in 2013 and reported in 2014 [1,2]. The initial stage collected 90 attributes of data including when, for how long, how often, and with what magnitude of effort and accomplishment, students engaged in work to complete programming assignments. Most of the data collection was automatic, participation was voluntary, and data from auxiliary sources, including a questionnaire on conflicting demands on time, complemented automatically collected data. Analyses revealed that procrastination and excessively brief work sessions were the main indicators of problems for students with inadequate prior success in earlier computer science courses.

Some students with successful track records knew when they could afford late starts. We added four attributes to the study in order to sort out students who could afford late project starts from those who could not, namely, number of overall undergraduate credits, undergraduate grade point average, number of computer science undergraduate credits, and computer science grade point average. The latter attribute proved especially useful, but not deterministic, in distinguishing successful from unsuccessful procrastinators. We hope to use the data to investigate the phenomenon of active procrastination in a future study. *Active procrastination* is intentional, constructive deferral of work, while *passive procrastination* is destructive work avoidance. The two leading theories are that active procrastinators are better time managers, deferring project work until needed in order to address competing tasks earlier [3], and that active procrastination increases the degree of positive stress or *eustress*, closely related to *flow*, forcing focused, efficient, and pleasurable engagement of the cognitive system [4]. One advantage of the current study over most studies of active procrastination is that this study works almost entirely with automatically collected, objective data, while those studies work largely with subjective surveys. However, the more immediate goal is to construct semi-automated aids for at-risk students, a topic appearing in the section on future work.

Distribution of work sessions across the day was a contributing factor to success. Additional analysis presented here indicates that working a greater percentage of daytime than nighttime sessions yields beneficial results for students whose performance varies significantly across projects. Also, using a few marathon sessions to attack complex problems does not work well for most students, so the only way to increase the overall work time without resorting to long sessions is to have multiple sessions, necessitating distribution across the day with breaks in between. The ability to distribute requires a relatively early start in the project cycle. Otherwise, time becomes short and the ability to schedule numerous work sessions of productive length becomes impossible.

The initial stage of the study found that conflicting demands on student time did not correlate strongly with

the degree of project success. Students used a short questionnaire to report conflicting programming projects from other courses as well as reporting exams in any course. Data collection did not include time spent on jobs or other extracurricular activities. Answers in the questionnaire comprised the primary form of subjective data in the study. They did not correlate well with student performance, either in specific student subgroups or in the student dataset as a whole. While task conflicts undoubtedly affect some students' performance, there was no clear correlation. It is likely that appropriate scheduling of programming work sessions by a student, as measured and analyzed by the study, lessens the impact of competing activities.

2. Related work

Edwards, et. al. have previously reported late starts in programming projects as clearly associated with poorer results on such projects [5]. That study utilized data from three programming courses for over five years. It confirmed results from earlier studies about the correlation of earlier project starts with better rates of project success. 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 projects. Portions of the current stage of the present study also take that approach, as reported in a section below. However, the present study mostly investigates all students, with special attention to at-risk students who perform poorly either part or all of the time. Consistently well-performing students are also of interest as they relate to active procrastination. 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 [6]. 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 [7]. 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 [8]. 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 and Temporal Patterns

3.1 Data collection and preparation

This subsection gives an outline of the process for data collection and extraction and the classes of data attributes collected for the study. Readers interested in more details of this process should consult the report of the initial stage of the study [1].

GNU makefiles [9] supplied by the instructor automated all three phases of compilation, testing, and deployment to the instructor of student code in the study. A student types **make build** for compilation, **make test** to run tests, and **make turnitin** to bundle the assignment directory into a compressed archive and send it to the instructor. Testing automatically invokes **make build** to update any object files whose source files are more recent due to coding changes. Some projects require students to write additional tests that are triggered by **make test**. Students may run **make turnitin** multiple times when they decide to over-write earlier submissions. The instructor's repository keeps only the most recent submission.

Each of these three make operations runs a data-collecting script that, unlike most make operations, does not echo its steps to the student's terminal. The goal is to maximize transparency of data collection in order to minimize interference with student work habits. Data collection consists of three steps. The first step captures the output of the UNIX "**ls –l**" command. This output shows the student project files, their sizes in bytes, and their most recent modification dates and times. These data reveal time, byte-size, and file-level locality of code changes. The second step captures a copy of all student source files. These data are useful in analysis of the number of sources lines added, changed and deleted using the UNIX **diff** utility. They would also be useful for a subsequent study of classes of student bugs, since the files collected

by this step support reconstitution of each stage of development of a student project. The third step bundles the data of the first two steps into a compressed archive with additional attributes embedded into the archive file name. These attributes include the student ID, the date, the time, and the type of the make action. The types include **start-of-build**, **completion-of-build**, **start-of-test**, **completion-of-test**, and **turnitin-to-instructor**, with start-of-test automatically triggering start-of-build. It is thus possible to observe both initiation and success or failure of compilation and success or failure of testing as distinct data attributes. Upon completion of archive compression, the makefile in the student directory moves the archive to a write-only directory in the instructor's UNIX account.

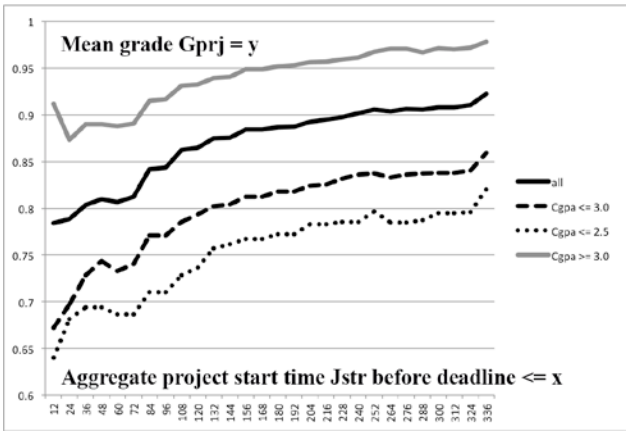
GNU make is available on the UNIX server system used for development of non-graphical projects. For graphical user interface (GUI) projects, many students worked on their own machines. Since the study was for a multi-section Java Programming course, with the Java Runtime Environment installed on student machines, it was straightforward for the instructor to supply a Java utility to drive compilation, testing, and data collection on non-UNIX student machines in a manner comparable to the makefile. The only data not collected were the student source files, in the interest of not overrunning available space on student machines. It was still possible to determine times and changes in file sizes in bytes from the directory listings. Students eventually moved their files back to their UNIX accounts for **make turnitin**. This step moved archives collected on student machines to the instructor's collection directory.

Students also filled in a three-question text form indicating number of programming projects started in other courses during the project timeline, number of programming projects due, and number of exams taken in any course. As noted earlier, these data showed no correlation with student success. The instructor also counted student emails about assignments in two classes, **clued** and **clueless**. Not surprisingly, the former correlated with more successful projects and the latter with less successful ones, but that correlation is trivially obvious to an instructor during project development. The study did not collect data about class attendance or extracurricular activities. Students participating in the study earned a 1% project point for permitting data collection and another 1% for completing the short survey. The IRB (Institutional Review Board) agreement required the instructor to award similar points to students choosing not to participate by invoking **make optout** (which would shut off data collection) and answer "optout" to survey questions. No students chose to opt out; participation was 100%, and many students were interested in the study results.

The most important class of data attributes analyzed is time, including project start time, completion time, and the time of every make action. Data extraction aggregated each series of make actions with fewer than 60 minutes between actions into a composite record for a **work session**. In addition to compressing and thus reducing the amount of data to be analyzed, creating aggregate work session records supported analysis of temporal length of work sessions, size of code changes during sessions, number of compilation and testing tasks started and completed during sessions, times of day of work sessions, and times between work sessions. The study analyzed all repetitive measures of time and project size changes in terms of per-session mean, population standard deviation, median, mode, minimum, and maximum. Data preparation resulted in one flat tuple of attribute values per student-project, with many attributes being statistical summaries of session properties. Course-project-student ID, overall project date-time period, student grade point average and university credit data, and data from the questionnaire also contributed attributes to each student-project tuple.

3.2 High level analysis of project data

This subsection gives very high-level statistical results for stage 2 of the study, based on the attribute selection process from stage 1 of the study [1]. Software tools used included custom Python scripts for data extraction and statistical analysis, along with Excel spreadsheet graphs. All data are for students including sophomores through seniors in an elective Java Programming course that has for a prerequisite the CS I-II sequence taught in C++. All project work was individual homework; there were no laboratory class sessions. There were 29 students with 111 student-project tuples of 91 attributes each in the 2013 stage 1 of the study, combined with two more sections of students in 2014 to yield 64 students with 282 student-project tuples of 95 attributes each in this stage 2 of the study. All results reported here are from stage 2.



Graph 1: Project grade as a function of start time

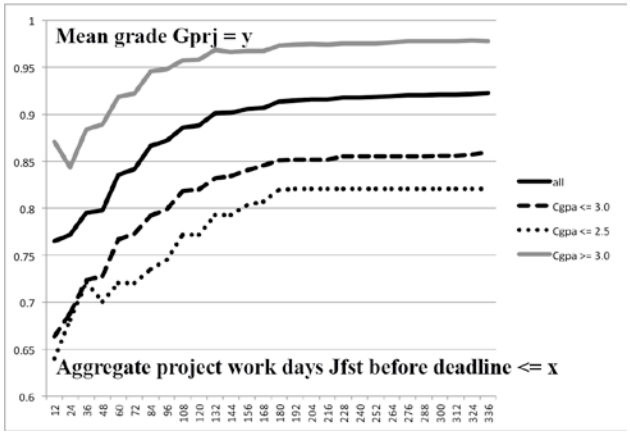
Graph 1 shows the average project grade (attribute **Gprj** on the Y axis) for students of four categories of incoming computer science GPA (**Cgpa** all, <= 3.0, <= 2.5, and >=

3.0) as a function of the time between the start of a project and the project deadline in hours (**Jstr** on the X axis). Note that the **Jstr** value is cumulative in reading the graph from left to right. For example, a **Jstr** value of 180 shows all student-projects with project start times <= 180 hours before the deadline, including all students to the left at **Jstr** 12 through 108. This approach to graphing eliminates localized peaks and troughs that often obscure trends. There are 37 student-project measurements at the left side of the **Cgpa** = all graph, 141 (half of all student-project records) at the **Jstr** = 156 hours point, and all 282 student-project records at the right side.

The span of time is 14 days (maximum **Jstr** = 336 hours) because most contributing projects had a length of 14 days. Phase 1 of the study confirmed the instructor's experience that most students in this course do not take advantage of more than two weeks of available time. In this dataset, all students are included in the **Jstr** = 336 average for **Gprj** at the right side of the graphs.

The detailed data show a 19% (~2 letter grades) spread for **Jstr** range [12, 336] in the **Cgpa** <= 2.5 graph, and a 14% (~1.5 letter grade) spread in the **Cgpa** <= 3.0 graph. There is a 7% grade spread for **Cgpa** >= 3.0, and a 14% spread for the **Cgpa** all group (all students). Average grades increase monotonically for all but the **Cgpa** >= 3.0 group. Clearly, the most at-risk population of students with respect to computer science GPA take the biggest hit from procrastination.

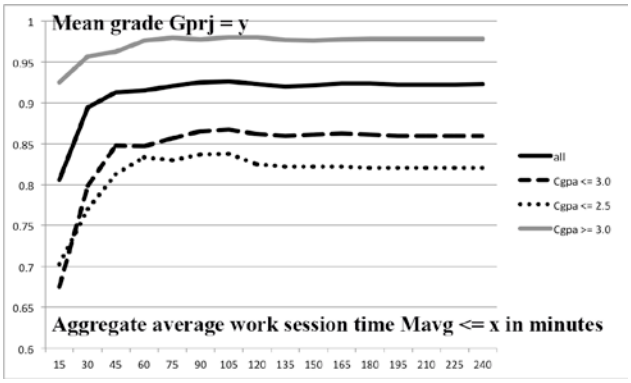
Graph 1 shows only averages; there are exceptions to these. The left side of the **Cgpa** >= 3.0 graph illustrates a sample of 15 of the total 152 student-project tuples of that graph with **Jstr** <= 12 hours and an average grade of 91.2%. These are successful procrastinators, and probably active procrastinators. In fact, eliminating their contributions to the graph results in a 10% spread in **Jstr** for **Cgpa** >= 3.0. These are definitely not at-risk students, and therefore not the primary subpopulation of interest in the current stage of the study. Nevertheless, they constitute a significant and interesting subpopulation.



Graph 2: Project grade as a function of workdays

The current, second stage of this study extracted a related set of curves that appear in Graph 2, average project grade **Gprj** as a function of the number of hours **Jfst** in days on which students actually worked on a project. The need for **Jfst** became apparent after detailed visual inspection of raw data following the initial report. Some students copied assignments into their accounts, put in some amount of work, and then left the projects sit untouched for days. **Jfst** is simply **Jstr** – 24 hours for each calendar day in which the project experienced no student make actions. The curves are similar to those of Graph 1, with similar **Gprj** ranges. Most significantly, all curves level off at around 192 hours (8 days). This set of graphs does not show procrastination in starting. Cramming all work into one day early in the project cycle would give a **Jfst** value of 24 or less, since all subsequent, non-work days would subtract out. What this graph really shows is that distributing work across up to 8 days leads to grade improvements, after which there are diminishing returns. There is slight growth in the actual **Gprj** values up through **Jfst** = 336 for all except the **Cgpa** <= 2.5 group, which is literally flat after 8 days. None of those students worked more than 11 days.

Graph 2 student-project values at the left side of **Cgpa** >= 3.0 are the same student-projects as the successful procrastinators of Graph 1. Their low **Jfst** work time took place during the last hours of the project timeline.



Graph 3: Project grade as a function of session length

Graph 3 shows project grade **Gprj** as a function of average length of a work session **Mavg**, again accumulating student-project records going left to right, where a session consists of multiple make actions with fewer than 60 minutes between adjacent actions. For all subgroups the minimum amount of time needed to achieve good results is about 60 minutes, and results fall off slightly after 105 to 120 minutes. Too little session time appears to result in inadequate engagement with the programming project, and too much may result in mild fatigue for these groups of programmers. Clearly, the decline is not as pronounced as the rise to 60-minute sessions.

Cluster	All data	**0**	1	**2**	3	4	5	6	7
Count	282	55	31	14	35	42	38	5	62
Count %	100.0%	19.5%	11.0%	5.0%	12.4%	14.9%	13.5%	1.8%	22.0%
Jstr	178.55	**52.60**	255.26	**54.86**	78.97	186.36	361.58	96.20	225.23
Jfst	91.40	**33.40**	108.16	**8.57**	50.86	82.93	205.58	72.20	113.35
Mavg	55.81	49.68	29.40	**32.26**	116.06	80.32	45.21	107.00	31.53
Mdev	44.52	29.91	30.85	25.29	57.49	**85.12**	50.17	19.79	32.35
Yavg	10229.25	**10103.84**	6191.93	**17188.36**	14658.20	**6885.83**	6573.48	75482.73	7530.67
Snum	5.36	3.20	7.06	2.93	2.77	7.31	8.84	1.40	5.29
Mtot	267.99	**148.13**	215.00	**113.07**	307.00	**530.64**	387.82	154.20	171.56
Cgpa	3.05	2.87	2.01	2.50	3.71	2.57	3.40	3.09	3.56
Gprj	0.92	**0.95**	0.94	**0.19**	0.98	**0.87**	1.03	0.95	1.00

Table 1: Simple K-means clusters for primary attributes that correlate with project grade Gprj

3.3 Detailed analysis of project data

The three statistical trends of Graphs 1 through 3 are the initial findings of this stage of the study. However, since they show cumulative averages, there are a lot of missing details. This subsection delves into some of the details.

Table 1 shows a set of K-means clusters in the columns that associate attributes in the rows that correlate strongly with project grade **Gprj** as extracted using the Weka data analysis toolkit [10,11]. The initial report describes the process of primary attribute selection [1]. The following list defines these attributes.

Jstr is deadline minus start time in hours.
Jfst is **Jstr** minus 24 for each non-work calendar day.
Mavg is the average number of minutes per work session.
Mdev is the sample standard deviation of **Mavg**.
Yavg is the average change in source code character count (bytes) per session.
Snum is the total number of work sessions.
Mtot is the clustered total number of minutes worked.
Cgpa is the computer science grade point average.
Gprj is the project grade. Note that with two bonus points for participating in the study, 1.02 is a "perfect score." Some projects also carried optional bonus points, 20% for the final project, giving **Gprj** an upper bound of 1.22.

In the actual data **Mtot** = **Mavg** x **Snum**. Clustering moves actual measurements to cluster centroids, hence the slight discrepancy in **Mtot** values.

Cluster 2 shows that K-means clustering associates the lowest **Gprj** value with the second-lowest values for **Jstr** and **Mavg**, and the lowest for **Jfst** and **Mtot**. **Jfst** is substantially lower than its value in the other clusters. This cluster has the second-highest **Yavg** value, showing that the worst performing student-projects crammed the second highest magnitude of code changes per session into around 3 very short sessions. The overall impression for cluster 2 is one of trying to "get it over with" in as little time as possible.

Cluster 0 has the lowest **Jstr** and second lowest **Jfst** values, but its **Gprj** is 95%, 5 times that of cluster 2. The **Cgpa** is somewhat higher for cluster 0, but most significantly, **Jfst** is about 3.9 times greater than cluster 2. **Mavg** is about 1.5 times greater (still on the low side), and **Mtot** is about 1.3 times greater. Each of these two attributes is a contributor to cluster 0's higher **Gprj**, but **Jfst** appears to be the primary attribute to correlate with the massive increase in project grade.

Cluster 4 is an interesting case. It has **Jstr** and **Jfst** values that approximate those of the "All data" column, but its **Gprj** value is 5% lower, constituting the second-lowest cluster grade. Its **Mavg** value of 80.32 appears to be well into the "success zone" until we consider its **Mdev** of 85.12, the highest. Cluster 4 has extremely high variations in session time, i.e., many extremely short sessions averaged with a few marathon sessions in **Mavg**. It has a low ratio of code size change in characters per session (**Yavg** / **Snum**) of 942, but clusters 5 and 1 come in lower for this metric at 743 and 876 characters respectively. The telling attribute for cluster 4 is the highest total work minutes **Mtot** of 530.64. This cluster has the lowest ratio by far of coding changes per minute at (**Yavg** / **Mtot**) of 13. Students in this cluster are not working consistently in sessions of 60 minutes or more and they are not accomplishing as much coding per minute as students in other clusters.

Cluster 1 is another case to consider. It has the lowest **Cgpa**, the lowest **Mavg** session time of 29.4 minutes (with a **Mdev** value that shows that some sessions meet or exceed 60 minutes), and the lowest **Yavg** code changes per session, and yet its **Gprj** value is 94%. What appear to be helping these students are high **Jstr** and **Jfst** values, i.e., lack of procrastination. They apply themselves in a fairly high number of sessions, with an adequate number

of 60-minute-or-greater sessions, while starting early and getting some work done on many days after starting.

Cluster 5 shows the greatest values for **Jstr** and **Jfst** correlating with the greatest **Gprj**. Session minutes **Mavg** are a little low, but the **Mdev** value that is greater than **Mavg** and the substantially highest number of sessions **Snum** of 8.84 show that these students are starting the earliest, working consistently across days, working an adequate number of minutes-per-session for an adequate number of sessions.

Co-variation of **Jstr** and **Jfst** with **Gprj** as seen in Graphs 1 and 2 and Table 1 correlates with distributing work across days. Other attributes that do not appear in Table 1 show average number of sessions centered during different 4-hour intervals of the 24-hour day, e.g., midnight until 3:59 AM, 4 until 7:59 AM, and so on. A linear regression formula derived by Weka [10, 11] for the attributes of Table 1 plus these six 4-hour time-of-day work periods is:

Gprj = 0.0007 * Jfst + 0.0012 * Mavg + 0.0088 * Snum
+ -0.0002 * Mtot + 0.0335 * S0003 + 0.0129 * S1215
+ 0.087 * Cgpa + 0.5166

The correlation coefficient of this formula is 0.35 with a mean absolute error of 0.13, i.e., its "guesses" are about 35% accurate, with a mean error of 1.3 letter grade. **Cgpa** carries the strongest weight of 0.087, with **Jfst** and **Mavg** contributing something (but not a lot). The small negative weight for **Mtot** corrects for small over-contributions of **Mavg** and **Snum**, noting that **Mtot** = **Mavg** x **Snum**. What is interesting here is that **S0003**, the total number of sessions centered between midnight and 3:59 AM, and **S1215**, the total number of sessions centered between noon and 3:59 PM, contribute the second and third largest weights, weights that are significant fractions of **Cgpa**'s. Weka does not extract weights for other 4-hour intervals, but those intervals are in the data used in the analysis that extracts this regression formula. Those intervals apparently do not contribute deterministic correlations with **Gprj**.

The instructor and a student collaborator in this study have been debating the significance of the time-of-day numbers for some time. Time-of-day contributions show up as significant in other analyses, but they do not show up in exactly this way. For example, one analysis showed that students who worked consistently only between 8 PM and midnight tended to perform poorly. That conclusion would agree with a conclusion about the importance of working in the early AM and early PM hours, but other analyses do not pin **Gprj** correlations to exact working hours.

Our current conclusion is that, in addition to distributing work sessions across multiple days as indicated by relatively high, co-varying **Jstr**, **Jfst** values, distributing work sessions across multiple times of the day, and having at least half of those work sessions be at least 60 minutes in duration, is better than attempting a large amount of work during a few sessions in a few days. Cramming work into a few days or a few sessions per day is not an effective strategy for programming for most students. That statement may seem like common sense to most computer science instructors. The objective data examined to this point confirm it.

Cluster	3	13
Count	22	18
Count %	7.8%	6.4%
Jstr	47.41	64.33
Jfst	37.59	45.67
Mavg	132.49	114.72
Mdev	38.90	101.41
Yavg	18985.54	8256.90
Snum	1.77	4.61
Mtot	235.86	505.94
Cgpa	3.68	2.56
Gprj	.98	.76

Table 2: Simple K-means clusters with procrastination

Table 2 represents a brief attempt to address the topic of active procrastination. It comes from the same dataset as Table 1, this time partitioned into 16 clusters. Table 2 elides six time-of-day attributes for work sessions that contribute to cluster formation, because these attributes show no discernable patterns. The two clusters in Table 2 have the second and third lowest values for **Jfst**, with the lowest **Jfst** cluster (not shown) failing abysmally. The key distinction between the more successful procrastinators from cluster 3 in Table 2 compared to cluster 13 is the higher **Cgpa**. Cluster 13 students work more hours across more sessions with less success than cluster 3 students. Students in both of these clusters are cramming, but cluster 13 students are doing so less effectively. The ability to procrastinate successfully typically correlates with a high computer science GPA, but a high **Cgpa** does not guarantee success in procrastination. We have not yet found attributes that distinguish the successful procrastinators among students with high **Cgpa** values. It is an established observation, however, that at-risk students with lower **Cgpa** values usually make poorly performing procrastinators.

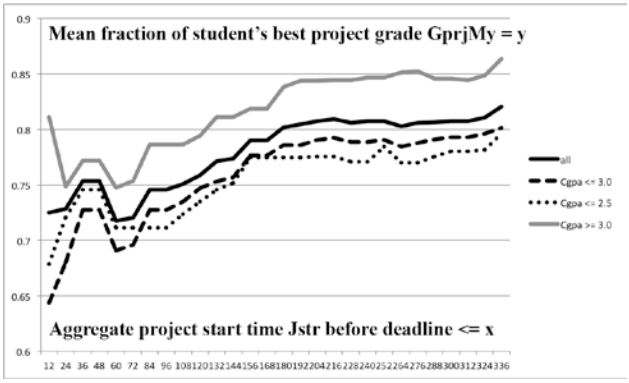
Other statistical analysis tools within Weka [10,11] including decision trees, model trees (decision trees with linear regression formulae at the leaves), and Bayesian correlation tables give similar results for important attributes to those presented here. A J48 decision tree [11] that considers only **Jfst**, **Mavg**, **Cgpa** and **Gprj** accurately predicts **Gprj** for 70.9% of the student-project tuples of Tables 1 and 2, with a mean absolute error of 0.14, about 1.4 letter grade. Adding a few attributes lowers this error to 0.12. This is the most accurate

predictor we have achieved to date, but the structure of the decision tree is too big to fit into a paper. It is filled with little local pockets of behavior that have as much to do with the details of our dataset and particular students as with general trends. The overall trends in the decision tree, however, reflect those of the graphs, tables, and linear regression formula appearing above.

3.4 Within-student analysis for students with large grade variance

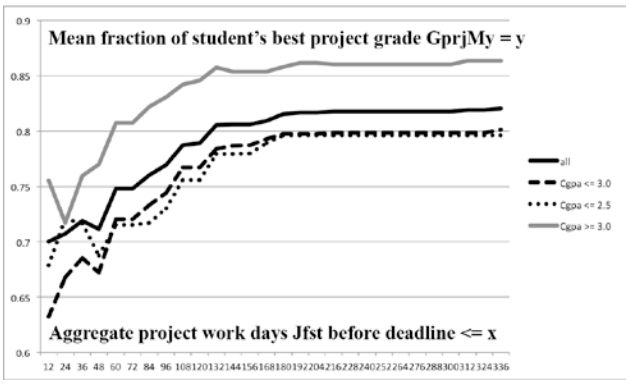
This section reports an entirely new portion of the study inspired in part by Edwards, et. al. [5]. It considers student-project tuples only for students with a project grade **Gprj** spread of at least 20% between their best and worst project grades. Different students occupy different overall grading bands, but each shares the fact that the difference between his or her best and worst **Gprj** is at least 0.2. This section introduces one new attribute, **GprjMy**, which is the ratio of a student-project grade to that student's best grade. A **GprjMy** value of 1.0 indicates that student's best **Gprj** grade, whatever it may be. The goal of this part of the study is to understand what students who perform better in some cases than others do differently in those sets of cases.

Graphs 4 through 6 are the **GprjMy** counterparts to the **Gprj** Graphs 1 through 3 at the start of this paper. Graph 5 irons out the noisy fluctuations of Graph 4 that come about because of various distributions of skipped available workdays after starting a project. Graph 5 levels out at about **Jfst** <= 7 days (168 hours), one day earlier than Graph 2 for these populations of grade-varying students. Otherwise, the overall trends, including successful procrastination for some **Cgpa** >= 3.0 students, remain the same.

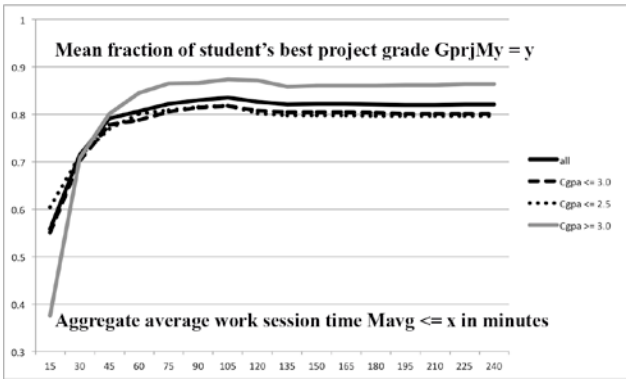


Graph 4: GprjMy as a function of start time

In Graph 6, all four curves peak at an **Mavg** value of 105 minutes per session, at the upper end of the 60 to 120 minute range, remaining fairly level thereafter. Students with substantially varying grades per project should plan to spend an average of around 105 minutes per work session to achieve best results.



Graph 5: GprjMy as a function of workdays



Graph 6: GprjMy as a function of session length

Cluster	All data	0	1	2	3
Count	116	29	18	26	43
Count%	100%	25.00%	15.52%	22.41%	37.07%
Jstr	152.26	293.62	69.67	63.81	144.98
Jfst	71.98	127.28	44.33	23.19	75.77
Mavg	56.78	51.82	118.92	27.40	51.87
Mdev	50.64	55.83	110.71	21.33	39.73
Yavg	9036.18	5351.37	10184.80	13526.58	8325.34
Snum	5.03	8.48	4.17	3.08	4.26
Mtot	268.83	440.17	471.83	85.77	178.98
Cgpa	2.79	2.49	2.64	2.31	3.34
GprjMy	0.82	0.86	0.76	0.65	0.92

Table 3: Clusters for students with varying Gprj

Table 3 gives condensed, 4-cluster Simple K-Means clusters for students whose **Gprj** grades vary 20% or more between projects. The average, "all data" cluster shows an average **GprjMy** performance of 82% of a student-project's best **Gprj** grade.

Cluster 2 with the worst intra-student performance correlates with the lowest values in the table for **Jstr**, **Jfst**, **Mavg**, **Snum**, **Mtot** and **Cgpa** as defined in the previous subsection. For this cluster and clusters 0 and 1 (**Cgpa** < 2.65), **Yavg**, the average number of source code characters modified per session, correlates negatively with **GprjMy**. Procrastination, cramming, and working

excessively short sessions all serve to hurt students with lower **Cgpa** values the most.

Cluster 3 with the best intra-student **GprjMy** correlates with the lowest **Mdev** / **Mavg** ratio (i.e., the lowest variation in a fairly substantial average session time) and the second-highest **Jstr** and **Jfst**. The implication is that maintaining consistent work session times, starting early, and distributing work across days can help to raise **Gprj** within a semester and therefore **Cgpa** across semesters.

Cluster	All data	0	1	2	3
Count	116	29	30	51	6
Count%	100.00%	25.00%	25.86%	43.97%	5.17%
Jstr	152.26	238.21	260.87	43.59	117.50
Jfst	71.98	126.48	105.67	22.41	61.50
Mavg	56.78	51.71	51.77	63.61	48.25
Mdev	50.64	55.53	53.04	42.78	81.87
Yavg	9036.18	7691.54	5665.92	12394.23	3843.15
Snum	5.03	5.69	7.73	3.00	5.67
Mtot	268.83	273.79	397.73	187.45	292.00
S0003	0.28	0.21	0.33	0.29	0.17
S0407	0.05	0.00	0.00	0.00	1.00
S0811	0.52	1.10	0.50	0.22	0.33
S1215	1.41	1.97	1.77	0.80	2.00
S1619	1.40	1.59	2.40	0.67	1.67
S2023	1.39	0.83	2.73	1.02	0.50
S00dev	0.63	0.77	1.16	0.39	0.75
night	1.72	1.03	3.07	1.31	1.67
day	3.32	4.66	4.67	1.69	4.00
n/ses	0.34	0.18	0.40	0.44	0.29
Cgpa	2.79	3.38	2.27	2.73	3.06
GprjMy	0.82	0.93	0.86	0.75	0.75

Table 4: Clusters for students with time-of-day distributions with varying Gprj

Table 4 reorganizes the dataset of this section into four clusters that include the S003 (00:00 through 3:59) to S2023 (20:00 through 23:59) attributes showing number of work sessions centered in six 4-hour time periods of the day. Derived attributes include **S00dev**, which is the sample standard deviation of S003 through S2023, **night**, which is the sum of S003, S0407 and S2023 for its column, **day**, which is the sum of S0811, S1215 and S1619 for its column, and ratio **n/ses**, which is the percentage of night work, i.e., **night** / (**night** + **day**). Some of the derived attribute values show a unit rounding error in the least significant digit because of rounding of their dependent attributes to 2 fractional digits for display purposes. The underlying, double precision calculations are accurate to approximately 15 decimal digits.

Student-projects in cluster 0 achieve 93% of intra-student potential (**GprjMy** of 100% being every student's top score in this dataset). Cluster 0 has the highest **Jfst** value and a substantial **Mavg** value in Table 4, but the attribute of interest here is **n/ses**, the percentage of night work sessions, which is by far the lowest at 18%. When performing their best in terms of **GprjMy**, students work mostly during the day. Clusters 1 through 3 have lower **Jfst** values – they do not distribute their work across days to the degree that cluster 0 does – and they have higher **n/ses** percentages – they do more of their work at night. It appears from this analysis that distribution of work session time in terms of minimizing **S00dev** is not as important as performing most work between 8 AM and 8 PM. Cluster 3 with the second-highest **Cgpa** and second-lowest **n/ses** percentage also has the lowest **Mavg** session time and the second lowest **Jstr** and **Jfst** values. Its **GprjMy** results suffer as a result of procrastination and extremely short work sessions.

4. Conclusions and Future Work

Phase 2 of this study as reported here adds several important aspects to phase 1 [1,2]. It more than doubles the number of student-project tuples, and it considers intra-student performance of students against their own best project scores for students with variations from highest to lowest project grade of 20% or more. The primary goal is to find ways to assist at-risk students in improving performance in computer programming projects.

- Procrastination in starting a programming project hurts performance. Likely reasons include inability to go to the instructor for help, inability to distribute work evenly across days (avoiding cramming in the last few project days), and inability to schedule around the conflicting demands of other courses by leaving too few days in which to perform useful scheduling of workdays.
- Skipping multiple workdays after starting can hurt performance. Eight actual workdays is a minimum for students of various computer science GPA levels to reach their respective potentials.
- Working fewer than about 60 minutes per work session can hurt performance. Some students can get by with 50-minute work sessions, but students with varying degrees of project success tend to do better in those projects in which they work at the upper end of the [60, 120] minute range. Increasing work session length beyond 2 hours leads to diminishing returns, especially when it combines with procrastination. It then takes the form of cramming work into the last few days of a project period.
- Distributing work across multiple sessions during a day, rather than cramming all work into one or two sessions, helps performance. Waiting until the last day or two of a project eliminates the possibility of intentionally distributing work across the day.
- For students with worst-to-best project grade differentials greater than 20%, no more than about 20% of the work should occur between 8 PM and 8 AM (night). Student-projects who have about 80% of their work between 8 AM and 8 PM (day) outperform other student-projects in the intra-student portion of

the study that measures each student-project's performance against that student's best performance in terms of project grade.

The second year of the study also collected and analyzed data from a junior-to-senior Programming Languages course taught using Python for assignments. The results of the analysis on that limited data set do not appear in this report because the student population differed substantially from students in the Java Programming course examined here. There was only one at-risk student who had problems completing assignments successfully. In fact, the CS II course that uses C++ eliminates many at-risk students before they get an opportunity to take the Java Programming course. It is the authors' conclusion that the CS II course would be the best place to continue data analysis and to apply experimental tools for assisting at-risk students.

Future work includes using data visualization tools to attempt to find patterns that remain elusive. Out best attempts are about 71% accurate in predicting student performance as a function of the attributes discussed, with an average error of about 1.4 grade letter. We are hoping that data visualization will help us to notice more patterns that we can then explore.

The plan for assisting at-risk students is to conduct a study that compares three alternative approaches: 1) construct an email-based "automated nag" that detects problems in work patterns during project timelines and then sends email messages to students advising them on how to change behaviors in order to improve likelihood of project success; 2) construct an interactive graphical computer game similar to Snakes and Ladders, driven by incremental, automatic data collection, that mirrors each student's improvement or decline in projected project score as a function of attributes; student's could observe their own projected success during a project cycle and compare them with anonymous classmates; 3) do nothing (control group). We would rotate students among each of the three approaches in different projects, and measure effect as indicated by participation versus lack of participation by the students. Students like to play on-line graphical games, and we conjecture that they would improve their work habits just to see their graphical avatars climb ladders. The instructor is awaiting an appropriate programming course in which to apply these techniques.

Investigation directed at the phenomenon of active procrastination is a final area for future work. Almost every research project into active procrastination uses subjective surveys. The present study uses concrete data collected by automated software compilation and testing. If we can figure out ways to use our datasets to investigate active procrastination, we may be able to make concrete advances in that area.

References:

[1] Anonymous, "Mining Student Time Management Patterns in Programming Projects," *Proceedings of FECS'14: 2014 Intl. Conf. on Frontiers in CS & CE Education*, Las Vegas, NV, July 21 - 24, 2012. The paper is available at ANONYMOUS_URL.

[2] Anonymous, "Using Weka to Mine Temporal Work Patterns of Programming Students," tutorial at FECS'14: 2014 Intl. Conf. on Frontiers in Computer Science & Computer Engineering Education, Las Vegas, NV, July 22, 2-3 PM.

[3] Angela H. C. Chu and Jin N. Choi, "Rethinking Procrastination: Positive Effects of 'Active' Procrastination Behavior on Attitudes and Performance," *The Journal of Social Psychology*, 2005, 145(3), p. 245-264.

[4] E. Kim and E. H. Seo, "The Relationship of Flow and Self-regulated Learning to Active Procrastination," *Social Behavior and Personality*, 2013, 41(7), p. 1099-1114.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Free Software Foundation, GNU Make home page, http://www.gnu.org/software/make/, February 2015.

[10] Machine Learning Group at the University of Waikato, "Weka 3: Data Mining Software in Java", http://www.cs.waikato.ac.nz/ml/weka/, February 2015.

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