Identifying different student clusters in functional programming assignments: From quick learners to struggling students

Anonymous Author(s)

ABSTRACT

Instructors and students alike are often focused on the grade in programming assignments as a key measure of how well a student is mastering the material and whether a student is struggling. This can be, however, misleading. Especially when students have access to auto-graders, their grade may be heavily skewed.

In this paper, we analyze student assignment submission data collected from a functional programming course taught at the X[1] university incorporating a wide range of features. In addition to the grade, we consider activity time data, time spent, and the number of static errors. This allows us to identify four clusters of students: "Quick learning", "Hardworking", "Satisficing", and "Struggling" through cluster algorithms. We then analyze how work habits, working duration, the range of errors, and the ability to fix errors impacts different clusters of students. This structured analysis provides valuable insights for instructors to actively help different types of students and emphasize different aspects in their overall course design. It also provides insights for students themselves to understand which aspects they still struggle with and allows them to seek clarifications and adjust their work habits.

KEYWORDS

Online programming platform; computer science education; cluster analysis

ACM Reference Format:

Anonymous Author(s). 2018. Identifying different student clusters in functional programming assignments: From quick learners to struggling students. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Online programming environments, such as RoboProf [8] for C++, DrScheme [13, 14] for Scheme or, more recently, Mumuki [4] , offer immense potential to enhance the students' educational experience in large-scale programming-oriented courses. They not only lower the entry barrier for beginners, but often feature autograding facilities that allow students to run and get feedback on their code while they are developing their programs, giving them the opportunity to fix bugs and address errors in their understanding right away. While having access to immediate feedback on their code has been recognized to significantly improve student learning outcomes and engagement (see, e.g., [15, 25, 29]), instructors and students alike are often too focused on the grade as a key measure of competency. Especially when students have access to auto-graders, the students' grade may be heavily skewed and misleading.

This paper develops a data-driven approach to better understand students' behavior when solving programming assignments in a functional programming course. In addition to the grade, we propose to consider additional factors such as the number of static errors and total time spent on solving programming assignments to identify student clusters. Using this methodology, we analyze student assignment submission data collected in a functional programming course taught at the X1 university which uses the Learn-OCaml online programming platform [5, 6, 17]. This allows us to identify four student cluster: *"Quick learning"*, *"Hardworking"*, *"Satisficing"*, and *"Struggling"*. While the first two clusters can be characterized as maximizers, i.e. students strive to achieve the highest possible grades and continue to improve their work, they still differ in the amount of time and effort spent on completing a given homework. In contrast, satisficing[2] students accept a possibly non-optimal outcome as "good enough" allowing them to adequately achieve their goals by saving time and effort. We further analyze these clusters with respect to work habits and the number and kinds of errors that are prevalent. This leads to four key insights:

- Leveraging the notion of *chronotype* - a circadian typology in human and animals, we confirm that a work pattern where students tend to work in the morning is related to academic success. In particular, quick learners tend to work more in the morning, while other clusters of students rely more on afternoon and evenings.
- Unsurprisingly, starting on the homework early is related, in general, to higher grades. However, we also noticed that satisficing students start relatively late, yet finish the earliest. This further emphasizes that satisficing students make pragmatic choices aiming for a satisfactory result rather than the optimal one. At the same time satisficing students have one of the lowest number of programming errors suggesting that they struggle significantly less with static errors than for example hardworking students.
- Our analysis of static errors shows that syntax and type errors are prevalent among all students. Further, students continue to struggle with these errors throughout the semester. In addition, our analysis points to other common mistakes such as non-exhaustive case analysis and the use of unbound variables.
- Taking into account students' ability to fix static errors, i.e. how many tries a student needs to fix a particular error, we

[1] The name is intentionally anonymized for the double-blind review.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[2] The term "satisficing" was introduced by H. Simon [26] to describe a decision-making process in which an individual makes a choice that is satisfactory rather than optimal.

notice that the failure/success ratio is particularly high for hardworking students. This highlights both their desire and drive to get the best possible grade, but also that their path is full of small stumbling blocks.

We believe our proposed set of features and data-driven analysis can provide instructors with a clearer and more detailed picture of students behaviours and performance. This in turn may be used to adjust how some concepts such as how to avoid static errors are taught. It may also be used to design different strategies for different students to enhance the students' learning experience. Furthermore, this data may be interesting to students themselves to better understand how well they do in a class and identify areas where they can actively make changes and seek help early.

2 RELATED WORK

Analyzing student data in programming courses is a central topic in learning analytics, and it is gaining increasing attention with the recent advances in storing and processing data. One of the core aims of analyzing student data is to understand student behaviour, and in turn improve student learning experience [20].

Over the past decade, there have been several studies that focus on identifying groups of students using cluster analysis. For example, Emerson et al. [12] uses cluster algorithms to identify student misconceptions in a block-based programming environment for non-CS major students based on program structures. Wiggins et al. [28] finds five major clusters of hint requests in a block-based programming system equipped with an intelligent tutor. Hossein et al. [19] leverages clustering analysis to further investigate the correlation between students' programming speed and programming behaviours by collecting programming snapshots whenever an action occurs. They then divide students in two clusters by comparing a student's programming speed to the median speed. Lahtinen et al. [22] uses different levels of Bloom's Taxonomy as features to identify six distinct student groups that instructors should be aware of when teaching introductory programming courses.

In contrast to these existing works, our work considers multi-categorical features involving the grade, total time spent on the assignment, and the number of static errors encountered to identify clusters of students. Hence, our analysis provides a richer picture.

Based on the identified clusters, we follow existing work in understanding work/rest patterns of students. In particular, Claes et al. [7] studies programmers' working patterns using clustering analysis on time stamps of commit activities of 86 large open source software projects. Zavgorodniaia et al. [30] studies the chronotypes of students through cluster algorithms using keystroke data. In our study, we use activity data (such as whether a student compiled or graded their homework) to study work/rest patterns of students. It is the first study in the context of typed functional programming.

We further analyze static errors in typed functional programming assignments and their impact on different student clusters. This is the first such study in this setting. Previous studies focus on compilation events in object-oriented programs written in Java. For example, Ahmadzadeh et al. [1] investigates compiler errors frequencies of student programs and debugging activity patterns in Java. They suggest debugging skills should be emphasized in the teaching of programming. Altadmri et al. [2] collects a large dataset comprising of compilation events of 250,000 students, which provides a robust analysis about errors patterns and time for fixing different errors. Denny et al. [9] also studies various syntax errors frequencies and how long students spend fixing common syntax errors. They also found that certain types of errors remain challenging even for higher ability students. Edwards et al. [11] analyzes 10 million static analysis errors found in over 500 thousand program submissions made by students over a five-semester period. The experimental results suggest errors frequencies made by CS major and non-major students are consistent.

Our analysis is one of the first that investigates in more depth the frequency of various static errors in the typed functional programming assignments. Here, static errors go beyond syntax and simple type errors and but include for example detection of missing branches in a program.

3 STUDY DESIGN

This research aims to gain a deeper understanding of how students develop typed functional programs (TFP). We assume that the grade alone is not a good indicator on how well a student masters basic concepts and achieves competency. Instead, we propose that taking into account the time spent as well the number of errors a student encounters provide a more nuanced picture. Hence, the main question that we tackle in this paper is how can we best identify different clusters of students taking into account grade, time spent and number of errors. We then analyze our clusters with respect to five hypothesis:

- **H1:** Even students with a high grade in programming assignments may significantly struggle with a range of static errors.
- **H2:** Despite a lower grade, students who spend less time and have a low number of static errors do in fact well overall.
- **H3:** Work/rest patterns of students as well as the time a student spends on the homework play a role in students achieving a high grade. It highlights how driven a student is.
- **H4:** Static errors in TFP range from syntax and type errors to detecting unbound variables and missing branches in programs. This wide range of static errors provides a fine-grained picture of concepts students find challenging.
- **H5:** Error fix ratio, i.e. how many tries a student needs to fix a static error, indicates how well students understand basic ideas in TFP and this is correlated to their understanding and performance.

3.1 Course Context

Our study concerns students in a a second-year undergraduate computer science course on programming languages and paradigms at the X university. The course introduces concepts about functional programming and programming paradigms. It is offered every semester with more than 300 registered undergraduate students. The data which we analyze in this study was collected in Fall 2021 when the course was taught in a hybrid flipped classroom format using prerecorded lectures. Students then had the choice to either attend an online Zoom or an in-person practice session that reviewed concepts from the videos. All office hours were online.

The course had six bi-weekly programming assignments each worth 5% of the final grade. The expected time for these assignments

	programming topics	#tasks
HW1	basic expressions, recursion	7
HW2	data types and pattern matching	6
HW3	higher-order functions	11
HW4	references, state, memorization	5
HW5	exception, continuations	5
HW6	lazy programming, toy language	5

Table 1: Overview of six programming assignments.

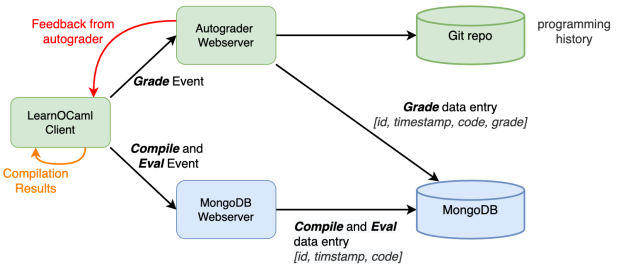

Figure 1: Data collection pipeline. `Grade` and `Compile` and `Eval` events are handled by different server, all submission data are stored in a *MongoDB* database. The components highlighted in light green are original components in the Learn-OCaml platform, while the components highlighted in light blue are newly introduced by us.

was approximately 6h. The bulk of the grade, 70%, was determined through an online midterm and an in-person final exam. Each homework consists of programming tasks which requires students to implement functions and test cases which were evaluated against unseen buggy implementations. Homework (or assignment) information is summarized in Table 1.

All homework assignment were hosted on Learn-OCaml [6], an online programming platform for OCaml which allows students to edit, compile, test, and debug code all in one place. We are using a modified version of Learn-OCaml by Hameer and Pientka [18], which extends the Learn-OCaml platform with features such as style checking and evaluation of test cases written by students.

3.2 Data Collection

Our data collection pipeline is built on top of the Learn-OCaml platform and it can automatically log students' actions. Specifically, we send *local* programming events like compile and evaluation (for testing and debugging) with asynchronous logging requests to our backend server. Figure 1 illustrates the process of collecting the data from the online environment Learn-OCaml.

Around 52.81% (i.e., 169 out of 320) students gave us consent to analyze their interactions with the Learn-OCaml platform. We collect more than 270,000 programming events, each event stores a snapshot of the code as well as feedback information (e.g., timestamp, static errors, grades, etc.).

3.3 Feature Engineering

3.3.1 Feature. For each homework, we collect a sequence of programming *activity events* for each student. The activity events include grade, compile, and evaluation events. This allows us to create an **activity density vector** for each student. It is a four-element vector that represents the percentage of the student's activity events that occurs in different ranges of hours [3-9, 9-15, 15-21, 21-3], which is the same choice of ranges suggested in [30].

In addition, we collect the following data based on the activity event sequence:

- **Start time**. Which day a student start *actively* working on an assignment based on the activity events collected.
- **End time**. The day when a student finishes an assignment, which is the last `Grade` event.
- **Working session**. Defined as the time window where activity events occur. If there is no activity event within 30min, then the working session is assumed to have ended.
- **Total time spent**. Sum over the length of all working sessions.
- **Number of errors**. Number of static errors that a student made while completing the assignment.
- **Grade**. The final grade a student receives for an assignment.

3.4 Feature Engineering

There are two challenges for applying clustering algorithms and statistical tests to our data. The first one is that some of our data is skewed. For instance, the grade distribution feature is highly skewed as students can always improve their grades through interacting with the auto-grader. The second one is the difference between feature scales, which renders the clustering results incoherent. We use two approaches to address these challenges. First, we use non-parametric tests including Spearman correlations and Kruskal-Wallis H-Tests. Second, we apply rank transformation on features to facilitate clustering algorithms.

4 IDENTIFYING STUDENT CLUSTERS

To identify student clusters, we run the K-means clustering algorithm on the aggregation (mean) of three most important features (i.e., *grade*, *number of errors* and *time spent*) over six homework. We use the *elbow method* to determine the optimal k (the number of clusters) to be 4. After determining the optimal k, we re-run the K-means algorithm and report the results in Table 2. We give the time in hours and note that all clusters have a similar size in terms of numbers of students (#*Std*).

To determine whether the resulting four clusters are different, we run a Kruskal-Wallis H-Test, which is a non-parametric equivalent of an ANOVA, on the three features (time spent, #errors, and grade) of each clusters. The results are statistically significant with the statistics of 113.26, 100.87, and 123.02 respectively, and all p-values < 0.0001. This suggests the four clusters are statistically different.

Clusters	#Std	Time (Hours)	# Error	Grade
A - Quick learning	46	5.30 (± 0.94)	66.11 (± 26.95)	95.24 (± 3.25)
B - Hardworking	46	8.24 (± 1.52)	148.67 (± 63.26)	94.25 (± 3.90)
C - Satisficing	31	4.47 (± 1.01)	52.26 (± 21.89)	74.43 (± 11.31)
D - Struggling	46	6.49 (± 0.94)	118.14 (± 35.32)	72.81 (± 11.03)

Table 2: Student clusters

Students in cluster A have the highest average grade (95.24), while spending less than the expected 6h on solving the homework. This suggests that while they achieve their goal with relative ease. In fact, students in this cluster outperform students in other clusters by a large margin. We characterize this cluster as **quick learning**.

Students in cluster B have the second-highest average grade (94.25). Yet, they also have the highest average number of errors (148.67) and with 8.24h spend significantly more time on the homework than any other group. In particular, they spend significantly more time than expected. This suggests that they face many difficulties which they manage to overcome by spending a significant amount of time. These students are driven to improve their work and to achieve the highest possible grade. Hence, we characterize them as **hardworking**. This data supports our hypothesis **H1**.

Cluster C has the lowest average number of errors (52.26) and spent the least amount of time (4.47h) on the homework. With the average grade of 74.43 they still achieve a "good enough" result. These students achieve their goal by saving time and effort. At the same time, these students seem to reach a satisfying level of competency as evidenced by their low number of average errors. We describe these students as **satisficing** students. This supports our hypothesis **H2**.

Students in Cluster D are in fact closely related to students in cluster B be showing a similarly high average number of errors (118.14) and significant amount of time (6.49h). Yet, compared to students in cluster B, they fail to overcome the difficulties along their path. These students are **struggling**.

5 UNDERSTANDING STUDENT CLUSTERS

5.1 How do work habits vary for different student clusters?

To investigate our hypothesis **H3**, we consider when students are active based on our activity data. Prior research suggests that chronotype, a person's preference in carrying out activity at certain periods in a day, is governed by the circadian cycle which is controlled by clock genes [10, 24]. In this section, we are interested in investigating the chronotypes, or in other words, work habits of students. In particular, it has been observed that "morningness" is positively correlated with academic achievement [23, 30].

To identify potential chronotypes, we run K-means clustering algorithm on the feature space spanned by activity density vectors. The elbow method yields $k = 3$, suggesting three possible chronotypes, which is different from four chronotypes reported in [30]. We report centroids of each chronotype cluster in Table 3.

Chrono clusters	3 - 9	9 - 15	15 - 21	21 - 3	Chronotype
Cluster 1	8%	14%	26%	**52%**	Evening (Eve)
Cluster 2	4%	**26%**	20%	50%	Morning (Mor)
Cluster 3	2%	19%	**37%**	42%	Afternoon (Aft)

Table 3: Centroids of each chronotype.

As we can see, most activities occur from 18:00 - 00:00 for all three clusters. This is not surprising as most students may have classes during the day. Based on this observation, we aim to define chronotypes by considering secondary activity peaks as well. We notice that Cluster 2 has its secondary activity peak (26%) in 6:00 - 12:00 whereas Cluster 3 has the secondary activity peak (37%) in 12:00 - 18:00. Thus, we define the Cluster 2 and 3 as the **morning** (Mor) and **afternoon** (Aft) type. The cluster 1 seems to have only one activity peak in 18:00 - 00:00, thus we define it as **evening** (Eve) type.

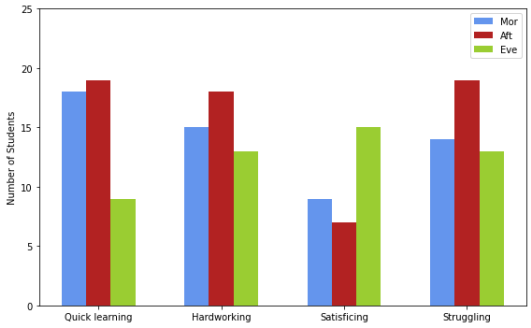

Figure 2: Chronotype distribution in each student cluster.

We plot the distribution of chronotypes in each student cluster. Figure 2. As the figure suggests, quick learning students usually tend to work in the morning and afternoon whereas satisficing students worked on their homework in the evening. This suggests quick learning students were driven, motivated and had possibly better time management skills. In general, satisficing students were the only group to have a strong incline to work in the evening. This could point to other commitments that students have or a high course load.

The afternoon type occurs most frequently in struggling and hardworking clusters. This may be because they were seeking help during office hours that were offered during the day or they simply required more time in general. Overall, our results confirm previous findings that certain chronotypes are related to academic achievement[23, 30].

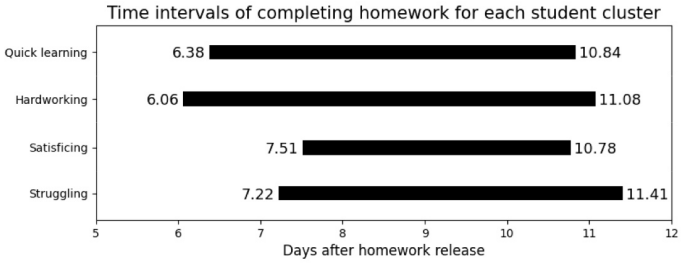

Figure 3: Clustering result of different types of students The start of a time interval stands for the average start time whereas the end represents the average end time.

5.2 How long do different clusters of students work on their homework?

To further investigate hypothesis **H3**, we investigate when students in a given cluster start and finish their homework. We report the

Error Groups	Error Categories	HW1	HW2	HW2	HW4	HW5	HW6
A. General Static Errors	1. Type Error	38.12%	30.94%	40.93%	32.65%	36.90%	34.83%
	2. Syntax Error	42.33%	21.54%	21.79%	32.68%	17.80%	25.66%
	3. Unbound value	10.42%	7.19%	9.06%	13.42%	7.02%	7.27%
	4. Missing else branch	1.92%	0.75%	0.43%	0.08%	1.03%	1.07%
B. Imperative Thinking Errors	5. Unused variable	0.74%	0.65%	0.63%	6.37%	21.34%	7.23%
C. Pattern Matching Errors	6. Pattern matching type error	0.84%	5.24%	2.13%	0.62%	1.37%	1.40%
	7. Non-exhaustive pattern matching	1.02%	16.78%	15.74 %	2.47%	4.62%	11.92%
D. Function Applications Errors	8. Wrong number of arguments	1.67%	2.19%	3.38%	1.17%	2.09%	1.89%
	9. Misuse of non-function values	2.50%	2.10%	2.07%	1.72%	1.50%	1.77%
	10. Others	0.88%	12.6%	5.89%	8.83%	6.33%	6.96%
	Total number of errors	7,850	27,519	14,331	19,859	22,467	26,681

Table 4: Error Groups and error categories together with their distribution of HW

average start time and end time for each cluster in Figure 3. In addition, the Kruskal-Wallis H-Test suggests start date was statistically significantly different ($stat$ = 22.59, $p\text{-}value$ < 0.0001) whereas end date was not ($stat$ = 3.12, $p\text{-}value$ = 0.37). Despite that, we can still observe some interesting patterns.

We note that both satisficing and struggling students start relatively late on their homework, at 7.51 and 7.22 average day respectively. However, satisficing students finish the earliest (10.78). This underscores the fact that they accept a "good enough" result rather than striving for the highest possible outcome. Further, satisficing students had the shortest working duration. This substantiates our claim that these students achieve their goals by saving time and effort.

Struggling students experienced the many difficulties as evidenced by high number of static errors that they encounter. These students finish indeed last (finish time (11.41)). This indicates that these students are struggling, although the do try their best until the very end. Yet, they lack the skills or support to overcome their difficulties.

Hardworking students have the longest time interval. While they start the earliest (6.06), they finish the second latest (11.08). This shows the commitment and dedication they bring to their work. Yet, we suspect that they are aware of their difficulties which may be one of the reasons for starting early.

Quick learning students tend to start quite earlier (6.38), although not as early as hardworking students. This seems to suggest that these students have confidence in their abilities to finish the homework smoothly.

We ran Spearman correlations to examine the correlation between start time and homework grade, the statistically significant result ($correlation$ = -0.42, $p\text{-}value$ < 0.0001) suggests procrastination affect negatively on student learning outcomes, which has been widely reported [3, 16, 21].

5.3 How do static errors affect students in different clusters?

Compilers for typed functional programming languages such as OCaml provide a wealth of errors and feedback to programmers. It not only reports on syntax and type errors, but also reports for example unused variables, missing branches in case-statements and if-expressions. This provides a basis to better understanding with what basic concepts students struggle with the most.

5.3.1 Overview of static errors. To investigate our hypothesis **H4**, we analyze the types of errors of each failed compile event and group errors into four main categories: general static errors (eg. group A), errors due to imperative thinking (Group B), and errors related to pattern matching and function (eg. groups C and D). We also include how often a particular errors occurred in assignment submissions (see Table 4).

The first homework shows a significant spike (42.33%) in syntax errors encountered. This is unsurprising, as the first homework includes "fix-me" exercises where students need to fix syntax and type errors in a given program. It is also the first time students attempt to write a program in a new language. However, it may be surprising that 20% to 30% of the errors encountered are related to syntax and type errors (Group A) throughout the semester. In fact, these errors constitute around 60% of errors for every homework assignment in Table 4. This may point to the fact that type errors in TFP catch conceptual errors in the programmer's thinking early rather than later during testing. Yet, it seems surprising that type errors play such a significant role. This may also suggest that instructors want to dedicate more time towards demystifying type error analysis. Presently, this knowledge is taught more implicitly along with other concepts such as higher-order functions or pattern matching.

For some key concepts from typed functional programming such as pattern matching, our error analysis indicates that students do improve and gain a better understanding of it. When pattern matching is first introduced in HW2, pattern matching errors and non-exhaustive pattern matching errors (Group C) consist 22% of total static errors. After practicing HW2 and HW3, the proportion of Error Group C drops greatly, which suggests that students gain a deeper understanding with more programming practice.

One of the prerequisites of this course is taking an introductory CS course, which is taught in Java or Python at our university. This implies that all of the participants had experience in programming before and had to deal with conceptual transfer from imperative/object-oriented programming (Python or Java) to functional programming (OCaml). Students usually report to transition smoothly between procedural language and object-oriented language for concepts such as *If-Conditionals* and *functions and*

scope[27]. From our observations, students seem to struggle more when transitioning to functional programming. In particular, they struggle with the concept of bound vs unbound variables, missing branches in if-expressions, or function application errors. Although these errors occur less frequently than syntax and type errors, we believe it highlights that students struggle with thinking recursively and considering all cases in such a recursive program (Error No.4,7). Therefore, if-else expression without an else branch also often leads to type errors in a language like OCaml.

Moreover, imperative programming supports variables declared in the local or global state, while in functional languages as OCaml we distinguish between stateful variables that can be updated and bound variables. While the concept of free variables and bound variables and the difference to stateful variables is discussed frequently in this course, students continue to encounter errors related to variables. In particular, the unbound value error occurs throughout the semester. This seems to be a sign that the concept of stateful variable declarations as used in imperative programming is persisting in how students think about a given problem. The most essential concept of functional programming is that functions are first-class citizens. Therefore, higher-order functions, which takes a function as an argument, or returns a function, are used frequently, especially in HW3 and subsequent homework. If functions are not used correctly, it would most frequently be flagged as a type error. However, OCaml also provides other error reporting. In particular, it may report on the incorrect number of arguments (Error NO.8) and use a function value instead of applying arguments on a non-function value (Error NO.9). These errors form a non-negligible class indicating where students stumble.

5.3.2 How efficiently do students in each cluster fix errors? Last, we investigate hypothesis **H5** and aim to understand how students in different clusters vary in their ability to fix errors quickly. Table 5 shows the average number of success compile events and failure ones experienced by different student clusters throughout the semester. The Failure/Success ratio x can be roughly interpreted as *debugging efficiency* or *error fix rate*— it on average costs a student x failure compile events to get a successful one.

	Quick learning	Hardworking	Satisficing	Struggling
Success	37.9	60.4	28.1	40.7
Failure	85.7	162.3	66.9	118
F/S	2.26	2.67	2.38	**2.90**

Table 5: Average success, failure and failure/success ratio (F/S) of compile events in each student cluster

Struggling students have the most difficulty in fixing static errors, requiring 2.9 failure compilations to fix the error on average. By contrast, quick learning students have the best ability of debugging with only a 2.26 failure compilation to get a successful one. Further more, the gap between their debugging efficiency is more significant, if we look at their average failure and success. While the average success for struggling students (40.7) and quick learners (37.9) are close, their average failure have a substantial gap: a struggling student has around 30 more failure compilations than quick learners.

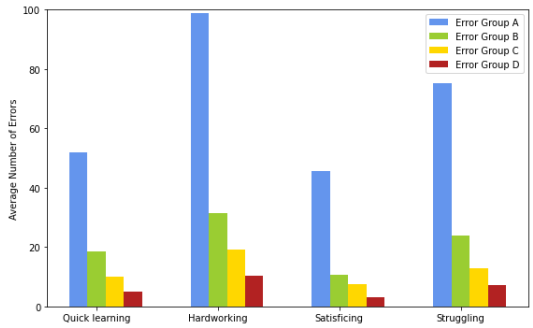

Figure 4: Distribution of static errors in each student cluster.

The row of Failure in Table 5 can be further represented by calculating the average number of each group of static errors for four student clusters in Figure 4. While for all clusters, type and syntax errors (Group A) dominates, there are noteworthy differences. Quick learners have fewer errors in all groups, not only general static errors, but also errors specific to functional programming. Satisficing students have the fewest errors in Group B, C, and D. This seems to indicate that they in fact achieve competency. Last, hardworking and struggling students have significant more errors in all error groups. In particular they seem to struggle more with basic concepts such as bound or unused variables, missing branches, and the proper use of functions.

6 CONCLUSION

In this article, we aim to understand how students develop functional programming assignments based on data collected through the Learn-OCaml programming platform. Our analysis considers grade, total time spent, and the total number of static errors to identify four student clusters: *"Quick learning"*, *"Hardworking"*, *"Satisficing"*, and *"Struggling"*. Using statistical tests we validate our clustering results along with other analysis results. This provides a nuanced picture of students behaviour and also exposes different paths towards achieving academic success in the course. Our analysis of chronotypes confirms that students who work in the morning reach a highest grade most quickly and smoothly. The total amount of time students spend on the homework also highlights the difference and similarity between the different student clusters. Although this part of the analysis was done in the context of a functional programming course, we expect our methodology to be applicable to other programming courses and help identify clusters of students (in particular hardworking and struggling students) who would benefit from additional support.

Our detailed analysis of static errors in typed functional programming also highlights areas where instructors can adjust their course content and possibly revisit topics. We believe our analysis also provides insights for students themselves, in particular the hardworking students, to understand which aspects they still struggle with and to seek clarifications. This would possibly allow them to become more efficient debuggers, spend less time on the homework assignment, and improve their conceptual understanding.

REFERENCES

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An analysis of patterns of debugging among novice computer science students. *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '05.* https://doi.org/10.1145/1067445.1067472

[2] Amjad Altadmri and Neil C.C. Brown. 2015. 37 million compilations. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education.* https://doi.org/10.1145/2676723.2677258

[3] Rahim Badri Gargari, Hossein Sabouri, and Fatemeh Norzad. 2011. Academic procrastination: the relationship between causal attribution styles and behavioral postponement. *Iranian journal of psychiatry and behavioral sciences* 5, 2 (2011), 76–2.

[4] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J. Gomez. 2018. The Effect of a Web-Based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 2–7. https://doi.org/10.1145/3159450.3159579

[5] Benjamin Canou, Roberto Di Cosmo, and Grégoire Henry. 2017. Scaling up functional programming education: under the hood of the OCaml MOOC. *Proceedings of the ACM on Programming Languages* 1, ICFP (aug 2017), 1–25. https://doi.org/10.1145/3110248

[6] Benjamin Canou, Grégoire Henry, Çagdas Bozman, and Fabrice Le Fessant. 2016. Learn OCaml, An Online Learning Center for OCaml.

[7] Maëlick Claes, Mika V. Mäntylä, Miikka Kuutila, and Bram Adams. 2018. Do programmers work at night or during the weekend? *Proceedings of the 40th International Conference on Software Engineering.* https://doi.org/10.1145/3180155.3180193

[8] Charlie Daly. 1999. RoboProf and an Introductory Computer Programming Course. In *Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education* (Cracow, Poland) *(ITiCSE '99)*. Association for Computing Machinery, New York, NY, USA, 155–158. https://doi.org/10.1145/305786.305904

[9] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All syntax errors are not equal. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12.* https://doi.org/10.1145/2325296.2325318

[10] Charna Dibner, Ueli Schibler, and Urs Albrecht. 2010. The Mammalian Circadian Timing System: Organization and Coordination of Central and Peripheral Clocks. *Annual Review of Physiology* 72, 1, 517–549. https://doi.org/10.1146/annurev-physiol-021909-135821

[11] Stephen H. Edwards, Nischel Kandru, and Mukund B.M. Rajagopal. 2017. Investigating static analysis errors in student Java programs. *Proceedings of the 2017 ACM Conference on International Computing Education Research.* https://doi.org/10.1145/3105726.3106182

[12] Andrew Emerson, Andy Smith, Fernando J. Rodriguez, Eric N. Wiebe, Bradford W. Mott, Kristy Elizabeth Boyer, and James C. Lester. 2020. Cluster-based analysis of novice coding misconceptions in block-based programming. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education.* https://doi.org/10.1145/3328778.3366924

[13] Mattias Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. 1998. The DrScheme Project: An Overview. *SIGPLAN Not.* 33, 6 (June 1998), 17–23. https://doi.org/10.1145/284563.284566

[14] Robert Bruce Findler, John Clements, Cormac Flanagan, Matthew Flatt, Shriram Krishnamurthi, Paul Steckler, and Matthias Felleisen. 2002. DrScheme: A Programming Environment for Scheme. *J. Funct. Program.* 12, 2 (March 2002), 159–182. https://doi.org/10.1017/S0956796801004208

[15] Vincent Gramoli, Michael Charleston, Bryn Jeffries, Irena Koprinska, Martin McGrane, Alex Radu, Anastasios Viglas, and Kalina Yacef. 2016. Mining Autograding Data in Computer Science Education. In *Proceedings of the Australasian Computer Science Week Multiconference* (Canberra, Australia) *(ACSW '16)*. ACM, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/2843043.2843070

[16] T. Hailikari, N. Katajavuori, and H. Asikainen. 2021. Understanding procrastination: A case of a ;study skills course. *Social Psychology of Education* 24, 2 (2021), 589–606. https://doi.org/10.1007/s11218-021-09621-2

[17] Aliya Hameer and Brigitte Pientka. 2019. Teaching the art of functional programming using automated grading (experience report). *Proc. ACM Program. Lang.* 3, ICFP (2019), 115:1–115:15.

[18] Aliya Hameer and Brigitte Pientka. 2019. Teaching the Art of Functional Programming Using Automated Grading (Experience Report). *Proc. ACM Program. Lang.* 3, ICFP, Article 115 (July 2019), 15 pages. https://doi.org/10.1145/3341719

[19] Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Exploring Problem Solving Paths in a Java Programming Course. In *PPIG*. Psychology of Programming Interest Group, 9.

[20] C. D. Hundhausen, D. M. Olivares, and A. S. Carter. 2017. IDE-Based Learning Analytics for Computing Education. *ACM Transactions on Computing Education* 17, 3, 1–26. https://doi.org/10.1145/3105759

[21] Irshad Hussain and Sarwat Sultan. 2010. Analysis of procrastination among university students. *Procedia - Social and Behavioral Sciences* 5 (2010), 1897–1904. https://doi.org/10.1016/j.sbspro.2010.07.385

[22] Essi Lahtinen. 2007. A categorization of Novice Programmers: a cluster analysis study. In *Proceedings of the 19th annual Workshop of the Psychology of Programming Interest Group, Joensuu, Finnland.* 32–41.

[23] Franzis Preckel, Anastasiya A Lipnevich, Sandra Schneider, and Richard D Roberts. 2011. Chronotype, cognitive abilities, and academic achievement: A meta-analytic investigation. *Learning and Individual Differences* 21, 5 (2011), 483–492.

[24] Steven M. Reppert and David R. Weaver. 2002. Coordination of circadian timing in mammals. *Nature* 418, 6901, 935–941. https://doi.org/10.1038/nature00965

[25] Mark Sherman, Sarita Bassil, Derrell Lipman, Nat Tuck, and Fred Martin. 2013. Impact of Auto-grading on an Introductory Computing Course. *J. Comput. Sci. Coll.* 28, 6 (June 2013), 69–75. http://dl.acm.org/citation.cfm?id=2460156.2460171

[26] Herbert Simon. 1956. Rational Choice and the Structure of the Environmen. *Psychological Review.* 63, 2 (1956), 129–138.

[27] Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 227–237. https://doi.org/10.1145/3372782.3406270

[28] Joseph B. Wiggins, Fahmid M. Fahid, Andrew Emerson, Madeline Hinckle, Andy Smith, Kristy Elizabeth Boyer, Bradford Mott, Eric Wiebe, and James Lester. 2021. Exploring novice programmers' hint requests in an intelligent block-based coding environment. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.* https://doi.org/10.1145/3408877.3432538

[29] Chris Wilcox. 2015. The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education.* ACM, 90–95.

[30] Albina Zavgorodniaia, Raj Shrestha, Juho Leinonen, Arto Hellas, and John Edwards. 2021. Morning or evening? an examination of circadian rhythms of CS1 students. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)* (2021). https://doi.org/10.1109/icse-seet52601.2021.00036