Broad Awareness of Unseen Work on a Concurrency-Based Assignment

Prasun Dewan
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
dewan@cs.unc.edu

Samuel George
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
sdgeorge@cs.unc.edu

Bowen Gu
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
gubowen2@live.unc.edu

Zhizhou Liu
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
yiwk321@cs.unc.edu

Hao Wang
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
harrywh@live.unc.edu

Andrew Wortas
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599
ajwortas@cs.unc.edu

***Abstract*— During the Covid pandemic, we gave a Java assignment that exercised threads, synchronization, and coordination and wrote tests to check each concurrency aspect of the assignment. We used four different technologies to record events related to work on this assignment: the Piazza discussion forum, the Zoom conferencing system, an Eclipse plugin, and a testing framework. The recorded data have given the instructors of the course broad awareness of several aspects of student work: How much time did a student spend on an assignment? How many attempts students made on thread, synchronization, and coordination tests before they reached their final scores? How many times did they go to Piazza or use Zoom-supported office-hour visits to fix concurrency problems, and what was the nature of these problems? How effective was Zoom transcription to classify the office hour problems? How long and effective were the office hour visits, and to what extent was screen sharing used during these visits? To what extent did students use the tests to determine if they had met assignment requirements? These data, in turn, have provided us with preliminary answers to a variety of questions we had about unseen work and the concurrency aspects of the assignment. While the answers may be specific to our assignment, the questions answered by these mechanisms can be expected to apply to other settings.**

Keywords—awareness, metrics, concurrency, education, defects, distance education, hands-on learning

I. Introduction

Programming assignments are critical in courses on concurrency as they allow students to see its non-deterministic impact on performance [1] and/or the user-interface [2]. For this reason, "nifty" assignments have been an important component of many workshops on concurrency education.

Traditionally, the only objective metric available to judge the impact of an assignment is the total score a student received on the submitted solution. This information does not indicate several aspects of the assignment, such as how challenging its various requirements were on students who completed it, and the extent and nature of the help they sought to implement it. Such awareness could potentially be used to offer help to shy struggling students, change the assignments, and adapt the presentation of the underlying concepts. These applications of awareness are particularly important for an early course on concurrency as there is less experience with them, and concurrent programs are believed to be difficult to implement and debug. Feedback about the difficulty of concepts is particularly important in remote instruction and large classes as visual clues of student confusion are absent.

Such awareness requires technology to record events relevant to the work. In summer 2021, during the Covid pandemic, we offered a remote course on object-oriented programming that covered concurrency. The last assignment exercised threads, synchronization, and coordination, and was accompanied with tests to check concurrency and non-concurrency aspects of the assignment. We used four different technologies to record events related to work on this assignment: Piazza for asynchronous discussions, Zoom for synchronous office hour visits, an Eclipse plugin called Fluorite [3], and a testing framework. Use of each of these recording technologies was voluntary.

In this paper, we describe the awareness mechanisms we derived from these events and the kind of concurrency questions they answered about the assignment. We first identify related work on awareness. Next, we give the context for this case study, outlining the nature of the course and the concurrency assignment. We then provide an integrated discussion of our awareness mechanisms and associated questions and answers. We end with conclusions and future directions.

This work was funded in part by NSF awards OAC 1829752 and 1924059.

II. RELATED WORK

In collaborative systems, awareness is defined as information about the activities of one or more actors that can influence the activities of the observers of this information [4]. In our context, the actors are students and the observers are instructors.

The Codeopticon [5] system was designed for exactly this context. It allows instructors to see the current programming windows of several students all at once. Whenever a student gets an error, this is flagged in the view of the students, and they can engage in chats with the instructors to help them solve their problems. The windows displayed in the instructor view are also grouped according to the number of errors a trainee has generated so that the instructor can focus on the students in need of the most assistance. The instructor can replay the history of actions of any student to further investigate the nature and severity of the difficulty.

Digital Show-How [6] also targets the education context. It is designed to provide scalable help when a student's task is to follow an instructor's demonstration in class. Thus, it is not designed for tracking homework on assignments. The system uses a programming environment called Bricks to give the instructor a summary of students' progress in terms of how many steps they have followed. Unlike Codeopticon, it supports solicited rather than unsolicited help. Whenever students have trouble, they can submit their code and the instructor can work to correct it in class.

CollabVS [7] is a related system built for industrial programming that provides members in a small software engineering team awareness about each other's actions in Visual Studio. Like Codeopticon, it can be used to provide unsolicited help on open-ended tasks. A programmer is provided with tiles for all team members that indicate (a) whether they are active, (b) whether they are editing or debugging, and (c) the file, class, and procedure they last edited. This information can be used by programmers to avoid conflicts and provide help to team members who have spent an undue amount of time on code on which the observers are experts.

None of these systems provide awareness of the kind of problems programmers face. The work by Lonnberg [8] has gathered this information in the context of a visual concurrency assignment in which students (a) create threads for different trains that can share tracks, and (b) ensure that a train changes tracks to avoid collision with another train. They are provided with assignment-specific code to create a visualization of the train movement which is referred to as the assignment environment. Based on the grading of the submitted solutions, accompanying reports, and student interviews, the authors identified defects in the submitted solution and created two classifications. One classification separated them into deterministic and non-deterministic errors. The other separated them into four categories: incorrect algorithm or implementation, assignment requirements, programming environment (language and standard libraries), and assignment environment. The same assignment was given in three different years. The number of errors in each category reduced in subsequent years, perhaps based on countermeasures taken in response to awareness of the information from previous years.

Like these efforts, our work is intended to provide awareness information to both help students and find ways to improve the course. Our contribution is using novel information recorded automatically and a new set of awareness mechanisms based on the recorded information that can, in turn, be used to offer new ways to offer help and improve the assignment.

III. CONTEXT

The assignment discussed here, which we refer to as the target assignment, was the last assignment in a Java-based course on object-based programming, offered in the summer of 2021, in which concurrency was a major component. As it was a summer-course assignment, it had the size of three semester-course assignments. It covered threads (concurrency), synchronization (mutual exclusion), coordination (managing thread interdependencies), assertions, exceptions, and abstract classes. The data structure course was a prerequisite. The first, second, fourth and sixth authors were instructors of the course. The first author was the lecturer. This was his first offering of this course, which is relatively new in our curriculum. The course had three other instructors, and together, they ensured that office-hours were scheduled from 9 am to 5 pm each weekday.

This course was a variation of a previous course (taught several times by the first author), which preceded the data structure course [9]. The concurrency aspects of the assignment of this course were based on three different assignments in the previous version [9]. A major difference was that concurrency aspects were required in this course and optional in the previous version.

In this offering, the assignments in the course together simulated a variation of the bridge scene from the movie *Monty Python and the Holy Grail* (Fig. 1).

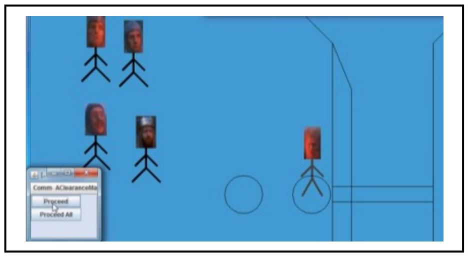


Fig. 1. Holy Grail Bridge Scene

The target assignment had four concurrency requirements checked by instructor-provided tests:

1. Asynchronous Knights: Provide a command to create a thread to animate the movement of a selected Knight such as Arthur or Galahad. The implementation creates a separate animator object for each thread or Knight, which provides an animation method that executes one or more loops that call *sleep()* before each movement.

2. Synchronized Knights: Two executions of a command to move the same (different) Knight are now serialized (execute concurrently). The implementation now has to create a single animator object for each Knight and declare its animation method as **synchronized**.

3. Waiting Knights: A thread is created to animate each Knight, but the threads wait for a start user-command to start the animations. The time between the steps of the animation can be different. The animation method now executes the *wait()* call in a global object before executing the animation loop. It unblocks when a *notifyall()* is executed on the global object in response to the execution of the start user-command.

4. Lockstep Knights: As above a thread is created to animate each Knight, which now makes its Knight move to a beat set by the clapping of the Guard. The animation method of the Guard executes the *notify()* method of a global object after each *sleep()* call and the animation() method for a Knight executes the *wait()* method of the global object before each movement (instead of a *sleep()* call).

Thus, the first requirement exercises threads, the second one synchronization, and the last two coordination. 31 Students submitted solutions to the assignment. Our discussion focuses on 20 of them for whom we have all of the recorded data needed by our awareness mechanisms. The assignment was given as homework, so students' work on it was not visible to the instructors.

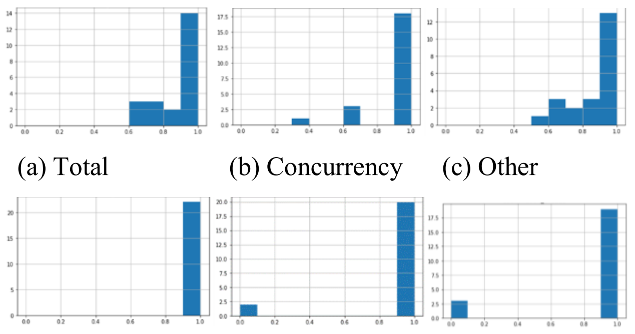
The student solutions had to conform to expectations set by the tests. For instance, they had to use a Java annotation to identify the method that animated Arthur's movement. Similarly, they had to ensure that the time an animation method took was bounded by lower and upper limits set by the tests. We refer to these constraints as test requirements to distinguish them from the assignment requirements above.

IV. Awareness Mechanisms

Our awareness mechanisms were developed in response to several questions we had about the unseen work in the submitted solutions. While we focus on those that are relevant to concurrency, many of these also apply to any unseen homework. The answers these mechanisms gave us illustrate and motivate the mechanisms. As they are specific to a particular assignment given to a small set of students, the answers may not apply to other settings. Thus, the awareness mechanisms are expected to be general but not the illustrative answers provided by them. Before we present an awareness mechanism, we pose the questions that motivate it.

A. Standard Total and Topic-based Scores

Since concurrency was now a required element of the assignment, we naturally wished to know the absolute and relative performance of students on the concurrency and non-concurrency aspects of the assignment. The test scores on the submitted assignments give this information readily.



(a) Total (b) Concurrency (c) Other

(d) Threads (e) Synchronization (f) Coordination

Fig. 2. Topic-Based Percentages

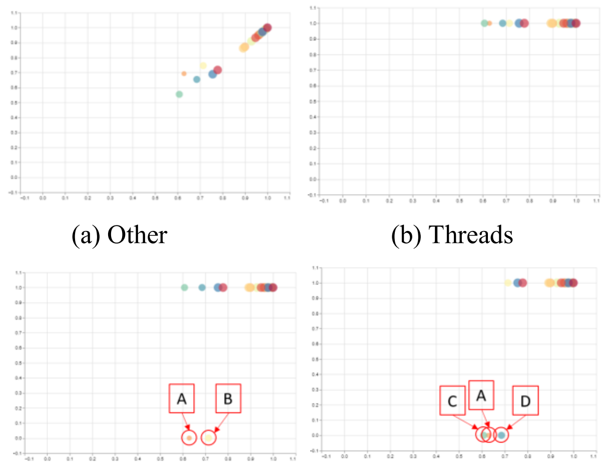
Fig. 2 shows by topic score histograms, with the X-axis being score percentages and the Y-axis being the number of students with a score range. Other stands for non-concurrency topics. Fig. 2 (a) shows that the vast majority of students did very well on the assignment. Fig. 2 (b) shows that a greater number did very well on concurrency aspects, though there were a few extremely low scores on these aspects. Fig 2 (d) shows that low scores on concurrency were not caused by the thread component- all students did very well on it. Fig. 2(e) and (f) show that synchronization and coordination posed problems for the students who received low scores on concurrency, with coordination posing more problems. This trend is consistent with the fact that the three concurrency concepts are inherently layered in the order: threads, synchronization, and coordination, with each topic building on the topics that precede it.

The high scores here are inconsistent with the number of defects in student submissions of the assignment reported by Lonnberg [8], which like our assignment, also visualized concurrency. Perhaps the reason is the use of instructor-provided tests to check the requirements before submission. We later discuss how extensively they were used.

Did students who struggled with some concurrency topic also struggle with other topics? If not, then perhaps the concurrency topic was especially difficult for some students.

The *concurrency-centered plots* of Fig. 3 provide answers to this question. In such a plot, each circle represents a student whose color identifies the student and size is proportional to the student's concurrency percentage The X-coordinate of the circle gives the total assignment score as a fraction of 1 for the student. The Y-coordinate of such plots vary. In Fig. 3(a), (b), (c), and (d), it gives the scores on the portion of the assignment characterized as other, threads, synchronization, and coordination. . *In all such plots the Y-axis 0 is the line above the X-axis. as circles on the X-axis look bad.*

These plots confirm that all students did very well on threads. Almost all of them (did well on synchronization and coordination – the exceptions are labelled A, B, C, and D. Of these four students, one of them did poorly only on synchronization (B), two did poorly only on coordination (C and D), and one student did poorly on both synchronization and coordination (A).



(a) Other (b) Threads

(c) Synchronization (d) Coordination

Fig. 3. Concurrency-Centered Plots for Topic Scores (Y-Axis Increment = 0.1)

The students in the other plot were more closely clustered together. However, students who did poorly on synchronization or coordination also received lower scores on other and overall. Based on these data, synchronization and coordination made the spread more extreme than other and threads, which is consistent with the intuition that these are more advanced topics than threads.

B. *Help Awareness*

Was the good performance of the vast majority of students on concurrency a result of seeking help? Was the poor performance of some students a result of not seeking help? Which concurrency topics required more help? What was the nature of the problems for which students sought help?

As a step towards answering these questions, we classified each concurrency-based Piazza help post and office-hours visit along the topic and type dimension. The topic dimension indicated whether it had to do with threads, synchronization, or coordination. The type dimension had the following categories:

1. Assignment requirement: Did the student need help in the initial understanding of some assignment requirement. An example was a Piazza post that resulted from a wrong assumption that the same class had to implement the animators for unsynchronized, synchronized, waiting, and lockstep animations.

2. Conceptual: Was the problem based on a failure to understand some theoretical concept or develop an algorithm to implement some understood requirement using an understood concept? An example was an office hour visit to understand how to implement the lockstep animations.

3. Test-unrelated: Was the problem based on unexplained behavior uncovered by the student's own debugging rather than an instructor-provided test?

4. Test-uncovered: Did the student seek help to interpret test output that uncovered some misunderstanding of assignment requirements or mistake with the implementation? An example was a Piazza post that resulted from the test for lockstep animation failing because the *wait()* calls were not called correctly.

5. Test-requirement: Did the student seek help to interpret some implicit or explicit test requirement? An example was a Piazza post resulting from an animation taking more than the upper limit of 5 seconds set by tests for synchronized animations.

The classifications were done by the third and fifth authors, who first did them independently, and then worked together to resolve their differences. Classifying the office-hour transcripts was particularly a problem as a result of transcription errors. These were of three kinds:

1. Word errors: The wrong word appeared in the transcript. Example: "lockstep: was translated into "lock semester."

2. Sentence errors: The context of a specific sentence was not enough to understand it. Example: "Yes, so your enemy they're basically your elevator store or the animation steps that you have to take in your strength right so, which means that yeah that's it."

3. Transcript errors: The context of the full transcript of an office-hour visit was not enough to classify it. There was no example of such an error.

To determine the usefulness of transcripts as a mechanism for help awareness we made some estimates of these errors after processing the transcripts using the Grammarly system for checking grammar problems. The number of grammatically incorrect sentences was considered a count of the number of word errors. The third author then looked at the sentences with these alerts and manually determined the number of sentences that were not understandable. The third and fifth authors then determined from all the sentences in a transcript whether they could classify it. As mentioned above, there was no such transcript. In 11 transcripts of office visits related to our target assignment containing 46651 words and 5441 sentences, we found 2842 grammatically incorrect sentences and 170 sentence errors.

Table I shows the number of posts for each topic and nature combination. A combination with 0 posts is not included. What is most striking is the small number of these posts in each concurrency row. This number indicates either (a) that students did not have problems represented by these rows, or (b) that the types of these problems were few, so each answer was helpful to many, or (c) that the problems were so severe that office-hour visits were used to resolve them.

The first three columns of Table II show the same information for office-hour visits. The numbers are higher now for the more advanced topics. For example, coordination was associated with 11 visits and 5 posts, while threads was associated with 3 posts and 2 visits. Unlike Piazza posts, information in office-hour visits is not shared. Hence a repeated problem results in repeated visits.

The data show that tests uncovered more problems than they introduced in the form of misunderstanding of the requirements they introduced.

TABLE I. CLASSIFYING ALL POSTS

Topic	Nature	Number
Other	Test-unrelated	9
Other	Assignment-requirements	7
Other	Test-uncovered	5
Other	Conceptual	1
Other	Test-requirements	1
Threads	Test-requirements	4
Threads	Test-uncovered	2
Threads	Conceptual	1
Coordination	Assignment-requirements	1
Coordination	Test-unrelated	1
Coordination	Test-uncovered	4
Synchronization	Test-uncovered	2

Based on the office--hour transcripts, we give in the last three columns of Table II the number of visits involving screen sharing (SS), the average duration of a visit, and the average fraction of this duration used for screen sharing. The visit duration indicates the severity of the problem. Screen sharing is a necessary condition for debugging so it may imply debugging.

TABLE II. CLASSIFYING ALL VISITS

Topic	Nature	#	# SS	Avg. Duration (Min)	SS Pct. Duration
Other	Test-unrelated	1	1	7.5	99.6%
Other	Assignment-requirements	3	3	23.8	99.6%
Other	Test-uncovered	4	4	9.9	85.1%
Other	Conceptual	4	4	23.8	97.9%
Other	Test-requirements	3	3	16.9	97.6%
Threads	Test-uncovered	2	2	7.4	92.9%
Coordination	Test-uncovered	10	8	19.9	91.8%
Coordination	Conceptual	1	0	1.1	0%
Sync.	Test-uncovered	7	6	12.6	85.3%

Some offerings of large courses known to the first author limit an in-person office-hour visit to 15 minutes. These data seem to indicate that this time would be sufficient even for online meetings. Our data showed that meetings longer than 20 minutes usually did not resolve the problem, and typically, these required the student to debug on their own. The fact that screen sharing was used so extensively, even for conceptual problems, is somewhat dismaying. It seems to indicate that students were not able to abstract out the nature of the problem in words and perhaps that the instructor did not try to make them do that. More investigation of these visits is necessary to determine if screen sharing was not warranted. If all of these problems required debugging, then, of course, screen sharing is desirable and these data show the benefits and usability of such sharing to solve concurrency and other problems in remote visits.

These data do not answer a question we raised earlier. Did the students who did poorly on a concurrency topic seek help through Piazza on that topic? The concurrency-centered plots of Fig. 4 and 5 address this issue. They mirror Fig. 3 except that in Fig. 4 the Y-axes show the number of posts of each student and in Fig. 5 they show the number of office-hour visits.

Continuing with our investigation of the low-performing students on concurrency, Fig. 4 shows that the low-performing student D was the only student to post anything on Piazza, and that was a single post not related to concurrency topics. Fig. 5 shows that only one of these four students, B, made an office-hour visit for any reason, and that visit was related to the topic of coordination, a topic on which B scored well. Thus, these plots indicate the low-performance on the advanced concurrency topics did not occur despite seeking help.

Fig. 4 and 5 show an example of a student, F, who performed well and had high posts (8) and office hour visits (16). We also see another high-performing student, E, who had no posts and only 4 office hour visits. We will investigate E in more depth later.

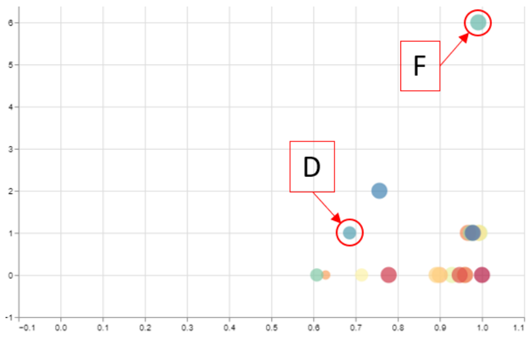
C. *Task Completion-Time Awareness*

As we see above, even high-scoring students made posts and office-hour visits. This, in turn, seems to indicate that they achieved success with some struggle. Help requests indicate problems students could not solve in the time available to them. Task completion time is an indication of struggle with problems solved by programmers on their own.

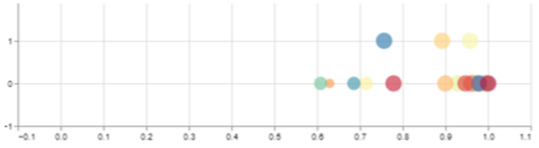
To estimate this time, we processed students' Eclipse timestamped commands logged by the Fluorite [3] Eclipse plugin. Assuming that a five-minute pause is a break, we calculated from these logs the total work-time of each student on a submitted assignment. Fig. 6 shows a concurrency-centered plot in which the Y coordinate gives the student's active work-time in minutes.

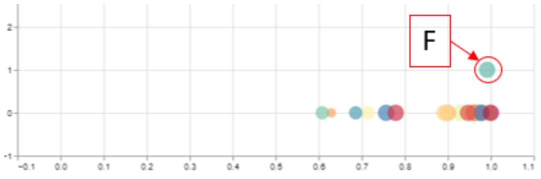
As we see here, students had widely varying work-times. Continuing with the low-performing students, their active time spent working was over 500 minutes, which is around the average. Thus, their poor performance cannot be explained by less than average work.

Among the students who had a high work-time is student F, who also asked for a relatively high amount of help. The highest work-time is of student E, who as mentioned earlier, sought very little help in comparison, which is consistent with research that shows some people are shy about asking for help. F took around 1500 minutes of active time to complete the assignment, while E took around 1800 minutes. Both students scored over 90% on the assignment. This information seems to indicate that some help might have reduced E's work-time, perhaps at the cost of some learning.

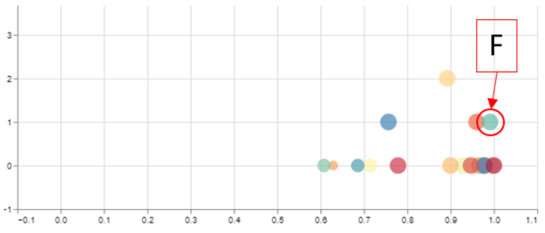


(a) Other


(b) Threads



(c) Synchronization



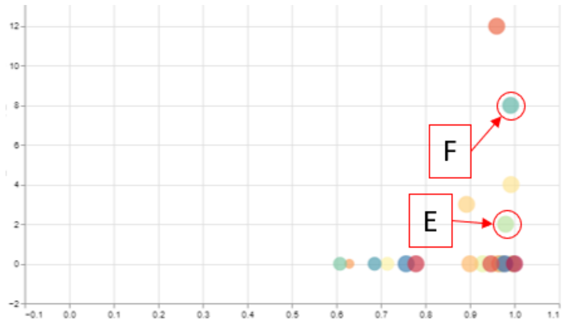
(d) Coordination

Fig. 4. Concurrency-Centered Plots for Posts (Y-axis Increment = 1)

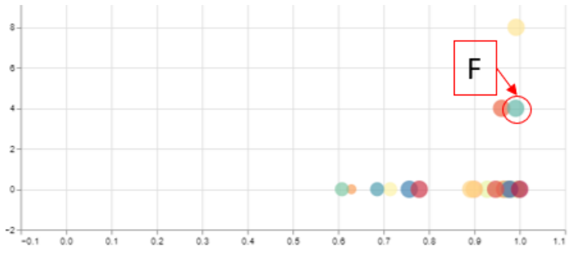
As instructors, we are told that a course worth X credits should require 3X hours per week outside class lectures. The data show here a large variance in these times, which is consistent with research that shows estimating project completion time is difficult. The fact that programming assignments take a high toll on some students may be a result of high variance and lack of data about task completion times. This awareness mechanism provides these times.

D. Topic Test-Attempts Awareness

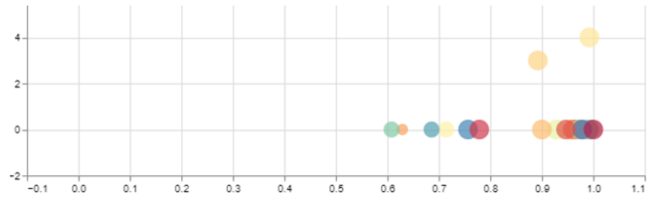
Task-completion time does not give us information about how much students struggled with individual concurrency aspects. One measure of such struggle, we introduced in an earlier paper [2], is the number of attempts on a test, which is the number of times the test was executed before it gave its final score. For each topic, we calculated the average of the number of attempts on a test related to that topic, which we refer to as *topic test-attempts*.

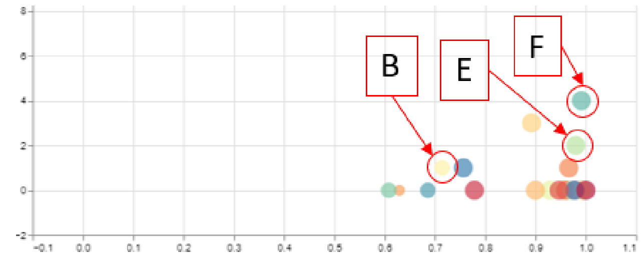


(a) Other



(b) Threads


(c) Synchronization



(d) Coordination

Fig. 5. Concurrency-Centered Plots for Office Hour Visits (Y-axis increment = 2)

Fig. 7 mirrors Fig. 3-6 except the Y-axes now show average topic test-attempts. As we see here, there is high variance also in these numbers. Some instructors like to limit the number of times instructor-provided tests are run, though there is no agreement on what this number should be. Our tracked data tell us how many attempts would be made when no bounds are placed. There were 46 tests overall (on all topics) and on average a student attempted an individual test about 3.5 times giving an average number of total test attempts of 162, a number much higher than the bounds we expect instructors place. If this is indeed the case, then these data show that limits are unnatural, in that, left to themselves, students would like to have more attempts. What impact the number of attempts has on learning is a matter for future research. Assuming the average of actual attempts divided by the number of desirable attempts is constant, the differences in the topic attempts distributions indicate that the limits should depend on the topic.

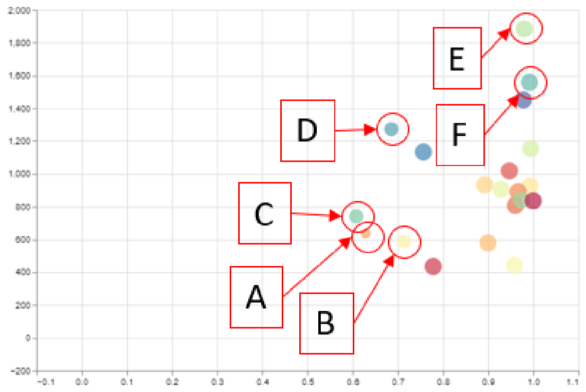


Fig. 6. Active Time Awareness (Y-axis increment is 200 minutes)

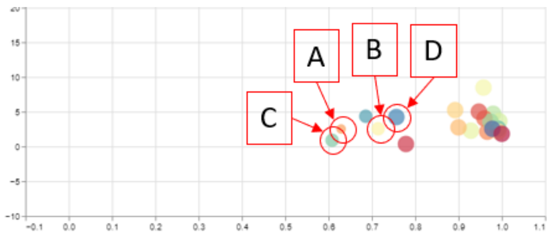
E. *Topic Test-Time Awareness*

The number of attempts on a test does not indicate how much time it took to fix the problems uncovered by it. Our test-attempts algorithm gives the time a test was first attempted and the time it reached its final score. Our task completion algorithm gives work-time between any two times, given the timestamped commands logged during this period. Using these two algorithms we computed the time a student spent on a test, and from these data, the average time spent on a test on a topic, which we refer to as the *topic test-time.* Fig. 8 mirrors previous concurrency-based plots with the Y-axes now showing average topic test-times in minutes.

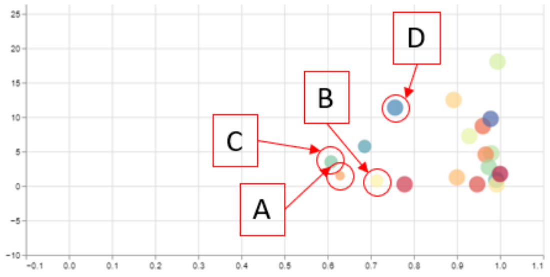
Continuing with the low-scoring students, we find that though they had average work-times, they had lower than average attempts and times on concurrency topics, which seems to imply that they did not make enough progress to have large numbers in these metrics. E and F have average attempts and times, which seems to indicate that their high work-time was caused by other topics.

We observed a much more significant variance in students' time and attempts for Threads than all other topics. For example, the standard deviation of average attempts for the thread topic is 7.04, while the standard deviation for other topics is only 1.75. Similarly, the standard deviation for average test times on threads was 51.77 minutes, while other topics were 9.38 minutes. Threads was also the topic on which every student received a high score, Thus, these data may indicate that with enough time, this is a topic that can be mastered, unlike the more advanced topics. Another cause might be that thread tests run student animations, and there were no requirements for the nature of the animations or how much time they took. Animations that finished much before the test timeouts (which were several minutes) resulted in much smaller test times than those that ran until the timeouts.

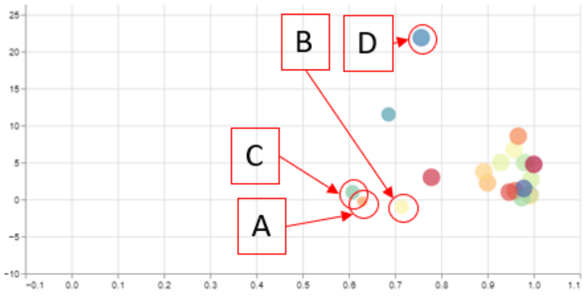
It seems intuitive to assume that the number of test attempts and test time would be correlated. We saw no correlation between the total number of attempts per test and the time spent on tests (Pearson Correlation = 0.07). Likewise, there was only a significant correlation between attempts and time on tests for the synchronization topic (Pearson Correlation = 0.90). With a bigger class, we might have seen some correlation.



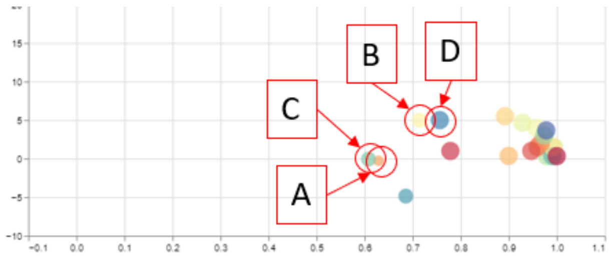
(a) Other



(b) Threads



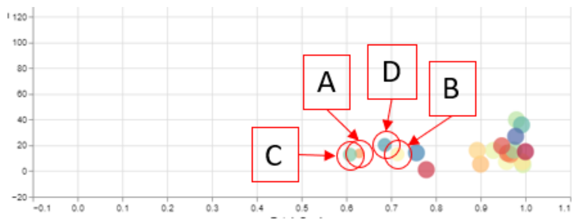
(c) Synchronization



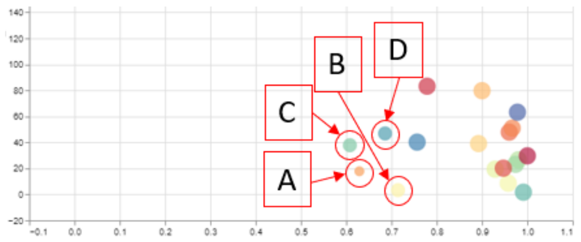
(d) Coordination

Fig. 7. Concurrency-Centered Plots for Test Attempts (Y-axis increment is 5)

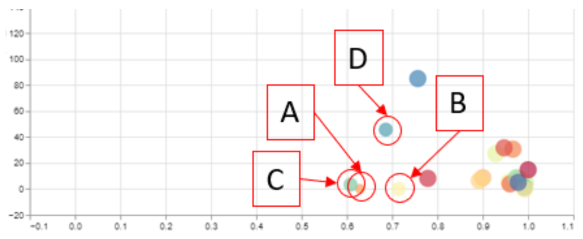
It is generally thought that some programmers are inherently faster than others. This could imply that a student who spent less than average time on tests on one topic would also do so for other topics. We found a strong correlation between time on other and total time (Pearson Correlation = 0.84), but found no correlation between times spent on other pairs of topics. This result can perhaps be explained by the fact that other had a very high fraction of tests and this contributed heavily to the grade – 36 out of 46.



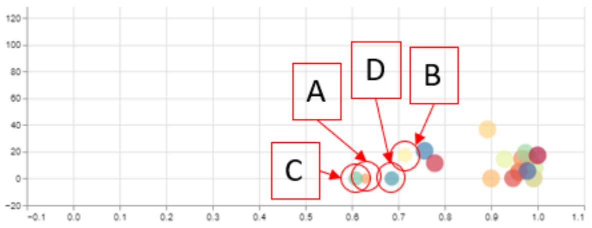
(a) Other



(b) Threads



(c) Synchronization



(d) Coordination

Fig. 8. Concurrency-Centered Plots for Test Time (Y-Axis increment = 20 minutes)

V. Conclusions

Our awareness mechanisms provided us with several insights about unseen student work on our assignment: Scores on concurrency topics, especially the two more advanced ones, created a much wider separation of students than non-concurrency scores. The low concurrency scores were not associated with lack of attempts or time spent on the associated tests, but were associated with low help requests in terms of office-hour visits and Piazza posts. Students who were not separated by scores were separated by other metrics such as requests for help, total work time, number of test attempts, and test time. The distributions created by these metrics were mostly unrelated. Threads created a wider distribution of attempts and test time than other topics. The problems solved by the tests were more than those created by the requirements they imposed. Screen sharing was used in the vast majority of office hour visits, even those that apparently required not debugging. The ones that were successful in debugging were less than 20 minutes. The more advanced concurrency topics also had more posts and office-hour visits. Some students spent an undue amount of time on the non-concurrency aspects of the assignment while asking for little help.

The findings we made from the awareness mechanisms presented here are, of course, of interest to us and anyone who wishes to offer a similar concurrency-based assignment to a similar set of students. Future work is required to determine if they apply to other settings.

Our more general contribution is the set of new awareness mechanisms themselves, which apply to other concurrency courses. Moreover, while our concurrency-centered plots are specific to concurrency assignments, the underlying information captured by the awareness mechanisms applies to any course.

Our mechanisms were developed after the course was completed. If these mechanisms indeed can identify struggling students to offer help to them, then infrastructures and user-interfaces are needed to provide their information to instructors as the assignment is being implemented.

Such mechanisms may not be usable in large, under-resourced classes, as they impose on instructors the burden of processing additional information. At least two approaches can be used to combat this problem. One is to provide this information, anonymized, to students, who can then learn from and help other students. The other is to use the information from these mechanisms to augment existing algorithms for automatically identifying difficulty [10] and recommending solutions [11]. The fact that these mechanisms created wide uncorrelated distributions indicates that each of them provides information that is independently useful.

Our classification of office hour visits and Piazza posts reveals information about the kind of problems students face in concurrency assignments. However, it is relatively coarse-grained and does not tell us, for example, if coordination problems were caused by people forgetting to call a *notify()* to unblock a corresponding *wait()*. A more detailed analysis of the recordings could reveal this information. Such an analysis is possible only because the Covid pandemic required us to provide online, recordable office hours, and thus the pandemic, interestingly, in this respect, was useful rather than detrimental. Motivated by this experience, the first author continues to support recorded online office-hour visits in the in-person class he is currently teaching. Our work also highlights the need for more NLP and transcription research to make transcripts more understandable. This work provides the basis for investigating these future directions.

Acknowledgment

This work was funded in part by NSF awards OAC 1829752 and 1924059.

REFERENCES

[1] Grossman, M., M. Aziz, H. Chi, A. Tibrewal, S. Imam, and V. Sarkar, *Pedagogy and tools for teaching parallel* computing at the sophomore undergraduate level. J. Parallel Distrib. Comput., 2017. 105(C): p. 18-30.

[2] Dewan, P., S. George, A. Wortas, and J. Do, Techniques and tools for visually introducing freshmen to object-based thread abstractions. Journal of Parallel and Distributed Computing, 2021. 157.

[3] Yoon, Y. and B.A. Myers. Capturing and analyzing low-level events from the code editor. in Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools. 2011. New York.

[4] Gutwin, C. and S. Greenberg, A Descriptive Framework of Workspace Awareness for Real-Time Groupware. Comput. Supported Coop. Work, 2002. 11(3): p. 411-446.

[5] Guo, P.J. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. in ACM Symposium on User Interface Software and Technology (UIST). 2015.

[6] Stotts, D. “Digital Show-How”: Extreme Active Learning for Introductory Programming. in 2019 14th International Conference on Computer Science & Education (ICCSE). 2019. IEEE.

[7] Hegde, R. and P. Dewan. Connecting Programming Environments to Support Ad-Hoc Collaboration. in Proc 23rd IEEE/ACM Conference on Automated Software Engineering. 2008. L'Aquila Italy: IEEE/ACM.

[8] Lönnberg, J. Defects in concurrent programming assignments. in Proceedings of the Ninth Koli Calling International Conference on Computing Education Research (Koli Calling 2009). 2010.

[9] Dewan, P., S. George, A. Wortas, and J. Do, Techniques and Tools for Visually Introducing Freshmen to Object-Based Thread Abstractions Journal of Parallel and Distributed Computing..

[10] Carter, J. and P. Dewan. Mining Programming Activity to Promote Help. in Proc. ECSCW. 2015. Oslo: Springer.

[11] Price, T.W., Y. Dong, and D. Lipovac. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. . in ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). 2017. ACM.