Custom "Caring IDE" for Online Offering of CS1

Mohsen Dorodchi
University of North Carolina at Charlotte
Charlotte NC, USA
mdorodch@uncc.edu

Mohammadali Fallahian
University of North Carolina at Charlotte
Charlotte NC, USA
mfallahi@uncc.edu

Erfan Al-Hossami
University of North Carolina at Charlotte
Charlotte NC, USA
ealhossa@uncc.edu

Aileen Benedict
University of North Carolina at Charlotte
Charlotte NC, USA
abenedi3@uncc.edu

Alexandria Benedict
University of North Carolina at Charlotte
Charlotte NC, USA
abenedi4@uncc.edu

ABSTRACT

The role of an integrated IDE capable of several different key features in teaching and learning programming is very clear to everyone. In this work-in-progress paper, we present the new version of our *Caring IDE*, a cloud-based IDE system integrated with a Learning Management System (LMS), an autograder, databases for storage, and dashboard prototypes to (1) deliver a smoother programming learning experience for students and (2) enhance the instructor's ability to informatively perform student success interventions quickly and early. Here, we report and extrapolate on the design and implementation of the *Caring IDE*. We also demonstrate the value of the *Caring IDE* in promoting student self-learning in an online, introductory computer science (CS1) summer course during the COVID-19 pandemic. Finally, we showcase preliminary IDE-based analytics to promote student success in CS courses.

Author Keywords

CS1; Interactive IDE; Learning Analytics; LMS/IDE Integration

CCS Concepts

•**General and reference** → Design; **Experimentation;** •**Applied computing** → **Distance learning; Interactive learning environments;**

INTRODUCTION

Instructors of introductory programming courses often discuss what an ideal Integrated Development Environment (IDE) may be like to best aid students with their learning. Cloud-based IDE's are one related growing area that has capabilities of integrating with systems like LMS (learning management systems) and autograders. They also have potential for features such as blocking and tracking possible plagiarism attempts. In addition, having analytics and visualization tools would provide a more interactive and real-time understanding of the students' progress.

Sixth SPLICE Workshop at L@S 2020 "Building an Infrastructure for Computer Science Education Research and Practice at Scale", SPLICE'20, August 12, 2020, Virtual Event

Learning analytics encompasses the analysis of data about learners in order to gain insights and improve their overall experiences [2, 21]. There has been an increased interest in learning analytics due to the availability as well as enhanced features in collecting data of the students' learning process [22]. To better understand learning and learners in computer science, researchers have augmented learner-centered features into their analytics such as: grades and other academic student-performance metrics [31, 10, 1, 8], student characteristics from surveys and reflections [19, 15, 18, 12], time spent on educational platforms [25, 17], course sequences [23], and novel models of student states [8, 7].

IDE-based learning analytics has been one of the major focal points for learning analytics researchers in which IDEs are used to collect data about learner's programming pattern and deliver learning interventions [21]. IDE-based analytics has many exciting possibilities for gaining better understandings of novice programmers, predicting student performance, and providing enhanced feedback to programmers on their errors. Hundhausen et al. [21] proposed a four-phase process model for IDE-based data analytics consisting of: (1) data collection, (2) data analysis, (3) intervention design, and (4) intervention delivery. Programming data collected from IDEs can include include: (1) editing data (e.g., code snapshots), (2) compilation data (e.g., compilation errors), (3) execution data (e.g., runtime exceptions), and (4) debugging Data (e.g., breakpoints, steps, and inspecting variables). This data comes in varying forms making it difficult for collaborative research in the area. To address this, Price et al. [27] propose a unified format of IDE-collected data to encourage collaboration in IDE-based analytics.

Our IDE, called the *Caring IDE*, integrates with our LMS (Canvas) and an autograder (in our case, Codepost) with the normal IDE functionalities to provide interactive and analytics feedback through the dashboards. Such functionalities would help both instructors and students with the teaching and learning of programming through actively practicing with close-to-realtime feedback. More recently, with the onset of COVID-19 and the fact that many courses have been and may

continue to be taught online, we have also begun to prioritize the IDE's interactivity to help promote self-learning. This interactivity is so that students may get aid more quickly and without necessarily needing the presence of an instructor or teaching assistant (TA). For example, the IDE will collect information about the students' programming habits (such as frequent errors made) so that instant feedback may be provided to help them improve. Another example is the inclusion of reflective prompts given to students after programming assignments to guide them in thinking about where they may be struggling (i.e., string methods, loops, or arrays) and to encourage additional practice in those areas.

In this work, we demonstrate the value of our custom cloud-based IDE system in promoting student self-learning in an online CS1 summer course during the COVID-19 pandemic. We also showcase preliminary IDE-based analytics to promote student success in CS courses. The rest of this paper is structured as follows. In the next section, we review literature regarding IDE-based analytics in computer science education. Next, we overview our Caring IDE system's architecture followed by a discussion of its features and our experiences using it in an online CS1 course. Finally, we present a preliminary analysis using synthetic student data and discuss our future directions.

LITERATURE REVIEW

Several works in IDE-based Analytics try to predict student performance [31, 10, 1, 8]. For example, Watson et al. [31] predict student performance in a computer science introductory course by using programming features (e.g., code snapshots upon compilation, compilation successes or failures, error messages, and timestamps) logged from the BlueJ IDE. The study reported that making less repeated errors and less time to resolve a compilation error is associated with stronger course performance [31]. Some works also took the next step and extracted further features from code snapshots. For example, Wang et al. [30] compare three different popular feature extraction techniques on programming code blocks: bag-of-words, and two abstract syntax tree features (n-grams and pq-grams) for classifying the code behavior of novice programming projects. Abstract syntax trees (ASTs) are code elements represented as tree nodes. Code errors are often discouraging to have; however, errors can be informative in understanding what students struggle with. For example, Ahadi et al. [1] analyzed SQL syntax errors and were able to identify concepts that students struggle in. Some errors are not as informative, perhaps to novice programmers, compiler errors may seem cryptic. Becker et al. [3] conduct an intervention study and report that enhanced Java compiler error messages reduced the number of overall errors, errors per student, and several repeated error metrics. With that in mind, several works address syntax errors by automatically fixing syntax errors [4] and generating hints to programmers [9, 28].

IDE-based analytics has also been used to study student behavior during programming exercises. Spacco et al. [29] use CloudCoder [20], an online IDE used to practice programming, to study the relationship between student exam performance with several IDE-based features (compilation success rates, time spent, exercises completed, etc.). Furthermore, IDE-based analytics has enabled researchers to create novel student state categorizations. For example, Carter et al [8] proposes the Programming State Model (PSM), which "categorizes students' programming within a two-dimensional space that captures both a student's current activity (e.g., editing, debugging) and the correctness of the student's most recently compiled programming solution". Several works use IDE-based features to categorize student programming experiences and student states [7, 6].

ARCHITECTURAL OVERVIEW

In this work, we present a revised and improved system architecture from our previous work [14]. The new architecture is illustrated in Figure 1. As shown, the new system displays a full-stack architecture which heavily utilizes cloud-based API's and microservices. The system includes several major blocks, including the main IDE interface, dashboards, microservices-based back-end and databases, learning analytics engine, and the integration submodule for the LMS and autograder. In the figure, green components symbolize what is already completed, yellow shows in-progress sections, and red components are still in the design phase. These system blocks would provide a spectrum of features and capabilities to the students and the teaching staff, which are explained in more detail in section 4.

The dashboards are designed to provide better access to available information inside the Caring IDE for both instructors and students. The student dashboard is more specifically designed to help students view information about their progress, weaknesses, and potential areas of improvement. Inspired by [9, 28] the future work hint-generator submodule provides hints to students automatically based on their inferred coding behavior. In our work-in-progress system, instructors would get class-level descriptive statistics of IDE-based analytics, including patterns of course-work submissions, coding attempts, and compilation success and failure rates broken down by assignment. The future work Machine Learning component would detect students who are at-risk of not meeting course requirements. Those students would be flagged for the instructor on their dashboard.

The Caring IDE integrates with the autograder and LMS API's for grading, assessment, due-date updates, and integrity check purposes. Currently, our system integrates with the autograder Codepost, and Canvas, the LMS that our CS1 course uses. With this integration, students experience a more uniform and understandable process of accessing and submitting their works directly inside the Caring IDE while grades will be stored in the LMS. For the instructor, the assignments only need to be created within the autograder, and then the IDE pulls that information using the autograder's API's to display to students.

These features were especially critical for our first time offering asynchronous and remote learning of the first programming course. Students have less direct instruction and interactions compared to traditional scaffolded active learning course models with additional lab sessions [16] and [17], which is why uniformity is essential to prevent students frustration in finding

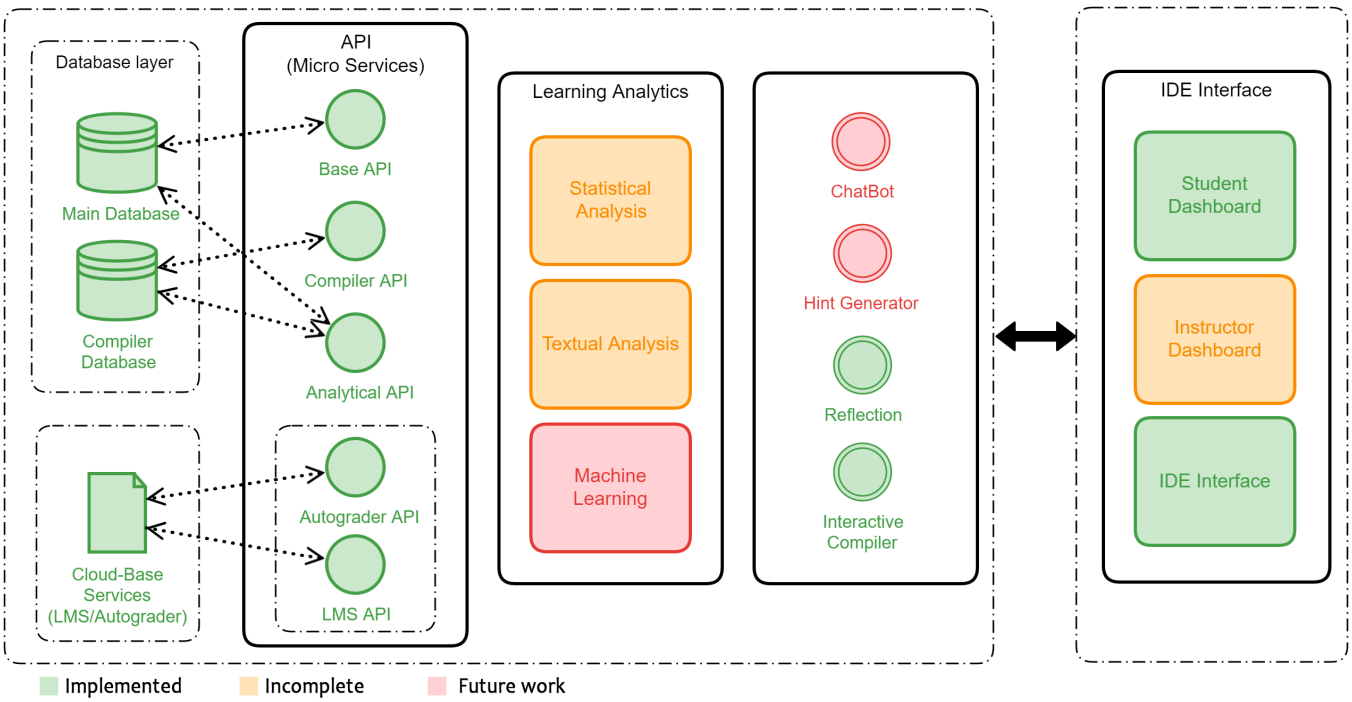

Figure 1. Architecture Diagram of the Integrated Caring IDE illustrating the database layer, API services, Learning Analytics, and Interface. Green components are completed, yellow are in-progress , and red are future work.

tasks. The students' reflections are also collected. Students are asked after completing each course exercise about their experience programming.

CARING IDE FEATURES

Our introductory programming course, ITSC 1212, has been heavily team-based and followed the active learning pedagogy in class as presented in [17, 13]. However, with recent events, the course was made to switch to an online, asynchronous structure for the Summer 2020 term. This created many obstacles in how the course content would be presented to students without deducting from their overall learning experience.

In an asynchronous class, there is no planned meeting time for lectures or labs. The course content consisted of the same material as previous semesters; however, it needed to be completely restructured in a way for students to still be able to follow along without the guidance that may come with an in-person class. Our class model includes closed lab to benefit student learning through hands-on coding practice. Students interact with each other throughout the labs to learn materials better. Being able to keep lab functionalities in an online setting was crucial. Our solution to this problem was the Caring IDE. Here, we will discuss its various features, the reasoning behind those features, and their educational benefits. The Caring IDE is a web-based application where students can login using account information provided by the instructor to then practice and complete their programming work. One concern of having an online, asynchronous course was the potential complications which can result from requiring more advanced IDE applications to be set up at the start of the semester. With this web-based IDE, students would no longer have to go through that initial set up process. Instead, they could then immediately focus on settling into the course itself, adjusting to the online environment, and learning the course material.

The Caring IDE is divided into three sections, as described below:

- **Assignment**: Students will go to this area to work on their large programming assignments. The instructions, due dates, and other information about the assignment is pulled from the autograder API. Student progress is saved as they work, and once they are complete, they can submit directly to the autograder tool from the IDE.

- **General IDE**: Here, students are provided with a blank area for coding. This can be utilized for practice or for any other lecture or lab homework activities which are not submitted through the autograder API. Students are able to export the programming file once complete and submit through the separate LMS.

- **Lab Activities**: This is an area to practice lab activities. Lab activities are created by an IDE administrator. In our case, we inputted our own lab activities from the course so that students could work on them from here. Once a student is done, they can submit their work to view the answer key and compare their code. Furthermore, students are prompted to reflect on their work after submission, which will be described further.

The Lab Activities section of the Caring IDE is designed in a way to compromise for the lack of in-person interactions (e.g., pair programming and in-class guidance). By providing

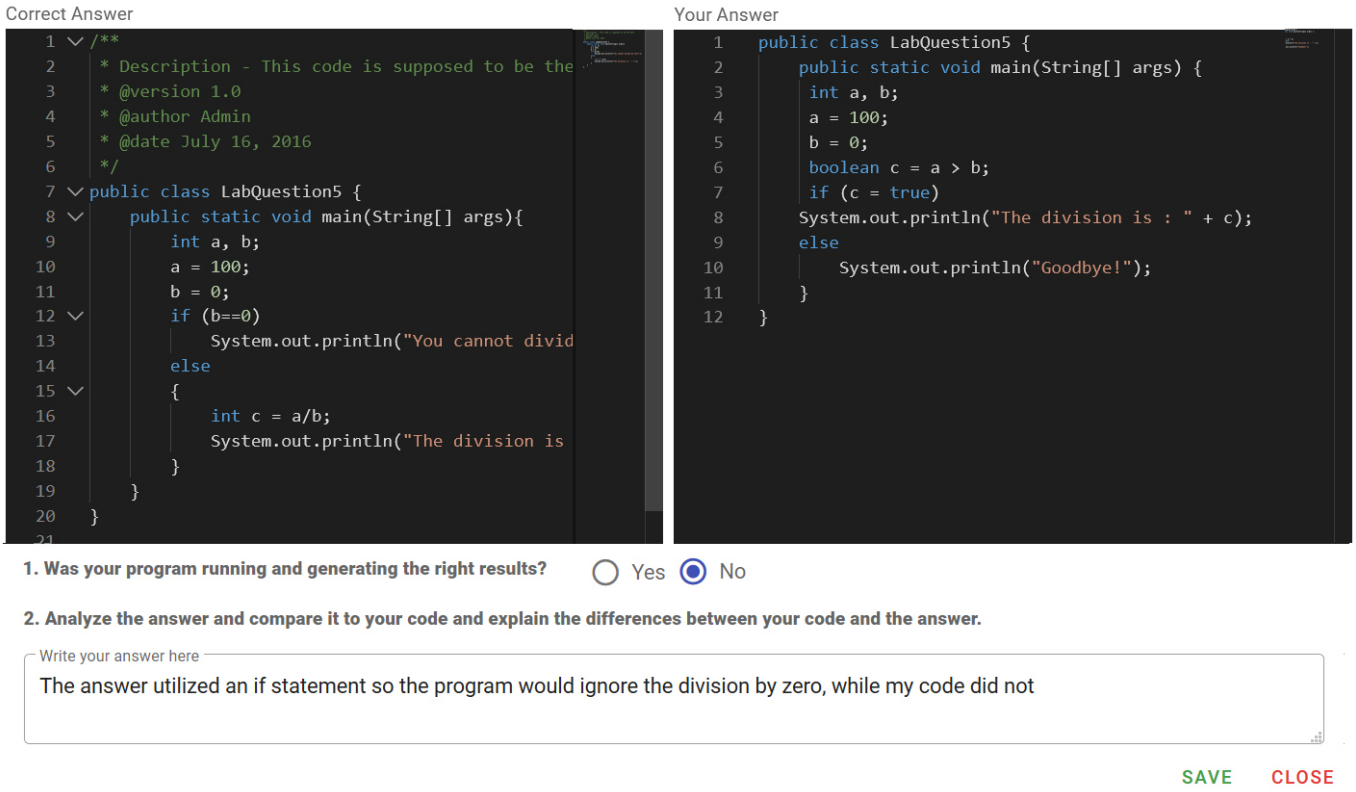

Figure 2. The top section shows a side by side comparison of a student's coded solution versus a solution provided. On the bottom, the student is asked to reflect on their work.

students with a side-by-side comparison of their submitted code to a solution provided by the instructor, students are able to get instant feedback from the IDE. It was often the case for an instructor or TA to provide a live demo during the lecture or lab as a way to help further explain concepts. It is also beneficial for students to see various examples of programs and to become aware that it is possible to program a solution in more than one way. However, this can be difficult to do in an online, asynchronous environment, and so this feature was one way to try to fill that gap.

After providing this side-by-side comparison, the IDE also prompts students to reflect on their work. This is shown in Figure 2. By prompting students to reflect on their work, there are benefits for both the students and the instructor. For students, additional learning occurs in the reflective process. For example, by reviewing their mistakes and intentionally thinking about where improvements could be made, more information is retained. Furthermore, students are able to see different ways of problem solving so that they can explore new methods and concepts they may be unfamiliar with. For instructors, information collected from student reflections can be used for various purposes such as tracking student progress, or knowing which topics students may need additional support in. For example, student reflections have been used previously to predict early on which students are at risk of falling behind so that interventions can be made [15].

On top of its many other benefits, the Caring IDE's web-based functionality can be white-listed within the LMS test-taking browser (in our case, we used Lockdown Browser), which is used to prevent plagiarism. This solves one of the complications of transforming the course from in-person to asynchronous. By being browser-based, students are able to take lab tests within the test-taking browser while being unable to diverge from the coding environment to search for outside sources. To also help prevent possible plagiarism during lab tests or assignments, students are unable to copy or paste within the coding environment. This is a feature only IDE administrators can toggle on or off, and can also prove to be beneficial to students as it promotes self-learning and encourages them to not rely on shortcuts that may take away from their learning progress.

ANALYTICAL SAMPLES

In this section, we showcase various IDE-based analytical figures from our instructor dashboard. The data used for demo purposes has been synthesized based on real students activities in the IDE, using the *DataSynthesizer* [26] method as we proposed previously in [11]. Figure 3 illustrates the relationship between student on-time submissions of course work and student final grades. We find that students who have done well in the course have tended to submit on-time more frequently (around 60% of the time) compared to those who have not satisfied course requirements.

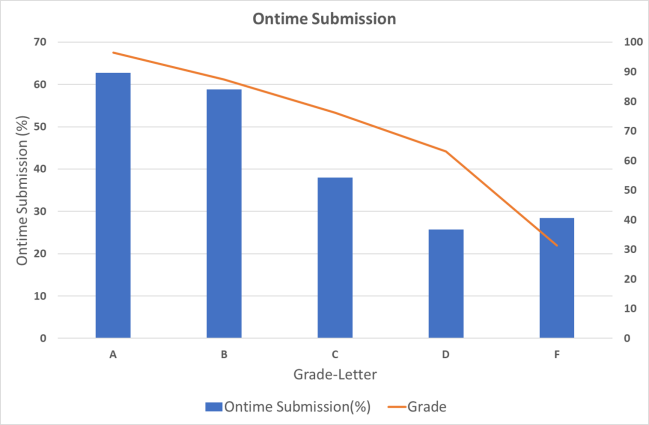

Figure 3. Comparison of on-time submission percentages based on the final points and grades.

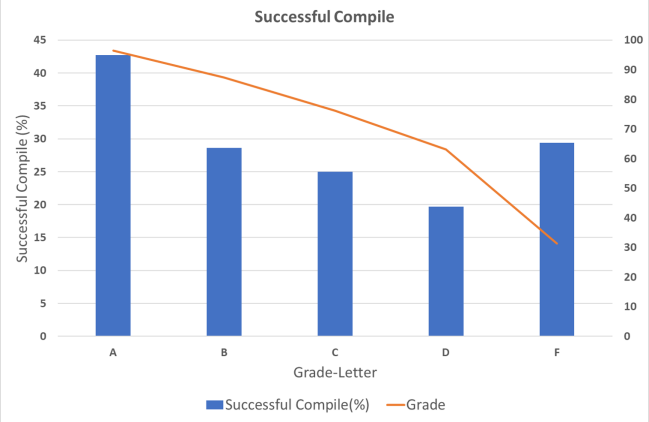

Figure 4. Comparison of successful compile percentages based on the final points and grades.

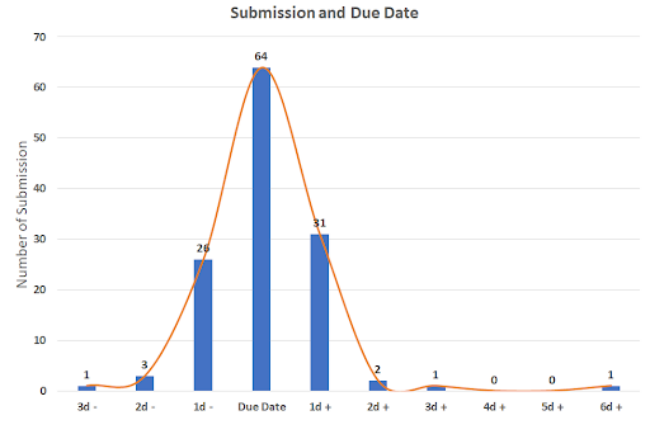

Figure 5. Distribution of submission dates considering the due date.

Figure 4 highlights the relationship between successful compiles throughout the course and final grade of students. We observe a negative relationship (with the exception students receiving F's), in which students who tend to perform well in the course tend to compile code successfully more frequently. Figure 5 showcases the distribution of student submissions and the time-distance from due dates. We observe that most submissions happen on the same day of the due date. On the student reflections, we perform several text mining methods including log odds ratio [24], topic modeling (i.e. Latent Dirichlet Allocation (LDA)) [5], and n-grams. Unfortunately we did not collect sufficient data to provide preliminary results.

FUTURE WORK

We envision an IDE that collects substantial and informative data points on students' programming experiences to enable close-to-real-time diagnostics and early interventions to enhance student coding experiences and learning. This vision was discussed in the original model [14] and in this work, we showcase further progress and details as illustrated in Figure 1. In the future, we plan to incorporate a standardized data format and protocols such as the one proposed by Price et al. [27] for our IDE logs to enable collaboration. We also have plans for a more complete integration along with many new features. For example, grades are currently entered manually into Canvas LMS by the teaching assistants after it is graded in the autograder. In the future, the system will further utilize the Canvas API's so that grades can be automatically be transferred from the system into Canvas. Furthermore, the hint-generator and chatbot development is currently in the design phase and will be our next major step. Lastly, we plan to add a Python compiler so that we can expand its usage for other courses, and will then complete a thorough user study with students in our upcoming semester.

REFERENCES

[1] A. Ahadi, V. Behbood, A Vihavainen, J. Prior, and R. Lister. 2016. Students' Syntactic Mistakes in Writing Seven Different Types of SQL Queries and its Application to Predicting Students' Success. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. 401–406.

[2] R. S. Baker and P. S. Inventado. 2014. Educational data mining and learning analytics. In *Learning analytics*. Springer, 61–75.

[3] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (July 2016), 148–175.

[4] S. Bhatia and R. Singh. 2016. Automated correction for syntax errors in programming assignments using recurrent neural networks. *arXiv preprint arXiv:1603.06129* (2016).

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[6] P. Blikstein. 2011. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge - LAK '11*. ACM Press, Banff, Alberta, Canada, 110.

[7] A. S. Carter and C. D. Hundhausen. 2017. Using Programming Process Data to Detect Differences in

Students' Patterns of Programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, Seattle Washington USA, 105–110.

[8] A. S. Carter, C. D. Hundhausen, and O. Adesope. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. 141–150.

[9] S. Chow, K. Yacef, I. Koprinska, and J. Curran. 2017. Automated data-driven hints for computer programming students. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*. 5–10.

[10] N. Diana, M. Eagle, and J. Stamper. 2018. Measuring Transfer of Data-Driven Code Features Across Tasks in Alice. In *Proceedings of SPLICE 2018 workshop Computing Science Education Infrastructure*. 5.

[11] M. Dorodchi, E. Al-Hossami, A. Benedict, and E. Demeter. 2019a. Using Synthetic Data Generators to Promote Open Science in Higher Education Learning Analytics. In *2019 IEEE International Conference on Big Data (Big Data)*. 4672–4675.

[12] M. Dorodchi, E. Al-Hossami, M. Nagahisarchoghaei, R. S. Diwadkar, and A. Benedict. 2019b. Teaching an Undergraduate Software Engineering Course using Active Learning and Open Source Projects. In *2019 IEEE Frontiers in Education Conference (FIE)*. 1–5. ISSN: 2377-634X.

[13] M. Dorodchi, A. Benedict, and E. Al-Hossami. 2019a. CS1 Scaffolded Activities: The Rise of Students' Engagement. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. 299.

[14] M. Dorodchi, A. Benedict, and E. Al-Hossami. 2019b. Integrated "Caring" IDE: A CS1 Tool. *Proceedings of SPLICE 2019 workshop Computing Science Education Infrastructure* (2019).

[15] M. Dorodchi, A. Benedict, D. Desai, M. J. Mahzoon, S. MacNeil, and N. Dehbozorgi. 2018. Design and Implementation of an Activity-Based Introductory Computer Science Course (CS1) with Periodic Reflections Validated by Learning Analytics. In *2018 IEEE Frontiers in Education Conference (FIE)*. 1–8.

[16] M. Dorodchi and N. Dehbozorgi. 2017. Addressing the Paradox of Fun and Rigor in Learning Programming. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. 370.

[17] M. Dorodchi, N. Dehbozorgi, A. Benedict, E. Al-Hossami, and A. Benedict. 2020. Scaffolding a Team-based Active Learning Course to Engage Students: A Multidimensional Approach. In *2020 ASEE Virtual Annual Conference Content Access*. ASEE Conferences, Virtual On line.

[18] R. Duran, J. Rybicki, J. Sorva, and A. Hellas. 2019. Exploring the Value of Student Self-Evaluation in Introductory Programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. ACM, Toronto ON Canada, 121–130.

[19] J. Fraser, J. McCuaig, and D. Gillis. 2019. IFS: An educational platform for analyzing course outcomes and student experience. In *Proceedings of SPLICE 2019 workshop Computing Science Education Infrastructure*. 7.

[20] D. Hovemeyer, M. Hertz, P. Denny, J. Spacco, A. Papancea, J. Stamper, and K. Rivers. 2013. CloudCoder: building a community for creating, assigning, evaluating and sharing programming exercises. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 742–742.

[21] C. D. Hundhausen, D. M. Olivares, and A. S. Carter. 2017a. IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda. *ACM Transactions on Computing Education* 17, 3 (Aug. 2017), 1–26.

[22] C. D. Hundhausen, D. M. Olivares, and A. S. Carter. 2017b. IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda. *ACM Trans. Comput. Educ.* 17, 3, Article 11 (Aug. 2017), 26 pages.

[23] S. M. MacNeil, M. Dorodchi, E. Al-Hossami, A. Benedict, D. Desai, and M. J. Mahzoon. 2020. Curri: A Curriculum Visualization System that Unifies Curricular Dependencies with Temporal Student Data. In *2020 ASEE Virtual Annual Conference Content Access*. ASEE Conferences, Virtual On line.

[24] B. L. Monroe, M. P. Colaresi, and K. M. Quinn. 2008. Fightin'words: Lexical feature selection and evaluation for identifying the content of political conflict. *Political Analysis* 16, 4 (2008), 372–403.

[25] Y. V. Paredes and D. Azcona. 2018. Predictive Modelling of Student Reviewing Behaviors in an Introductory Programming Course. In *Proceedings of SPLICE 2018 workshop Computing Science Education Infrastructure*. 5.

[26] H. Ping, J. Stoyanovich, and B. Howe. 2017. Datasynthesizer: Privacy-preserving synthetic datasets. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. 1–5.

[27] T. W. Price, D. Hovemeyer, K. Rivers, G. Gao, A. C. Bart, A. M. Kazerouni, B. A. Becker, A. Petersen, L. Gusukuma, S. H. Edwards, and D. Babcock. 2020. ProgSnap2: A Flexible Format for Programming Process Data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Trondheim Norway, 356–362.

[28] K. Rivers and K. R. Koedinger. 2017. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 37–64.

[29] J. Spacco, P. Denny, B. Richards, D. Babcock, D. Hovemeyer, J. Moscola, and R. Duvall. 2015. Analyzing Student Work Patterns Using Programming Exercise Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*. ACM Press, Kansas City, Missouri, USA, 18–23.

[30] W. Wang, Y. Rao, Y. Shi, A. Milliken, C. Martens, T. Barnes, and T. W. Price. 2020. Comparing Feature Engineering Approaches to Predict Complex Programming Behaviors. (2020), 8.

[31] C. Watson, F. W.B. Li, and J. L. Godwin. 2013. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*. 319–323. ISSN: 2161-377X.

APPENDIX

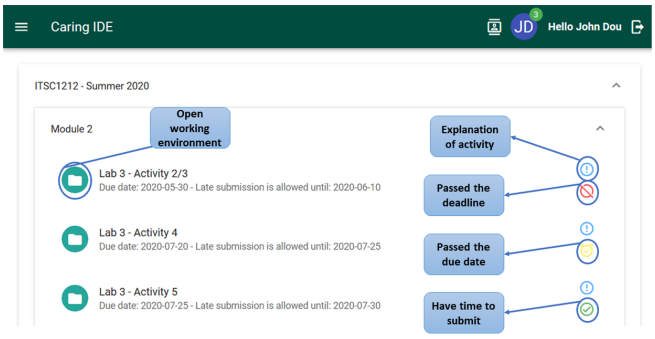

Figure 6. In lab activity section, students can find list of activities which are posted in LMS.

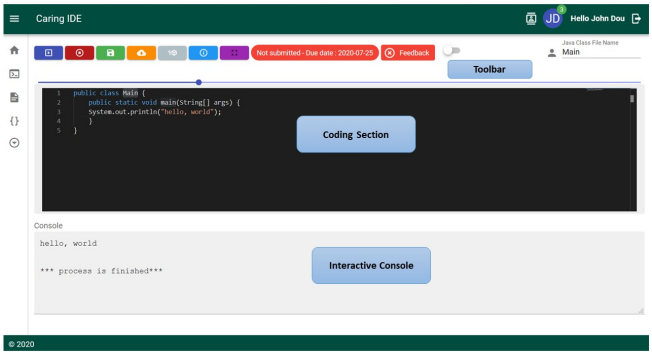

Figure 7. Explanation for each section in the developed IDE.

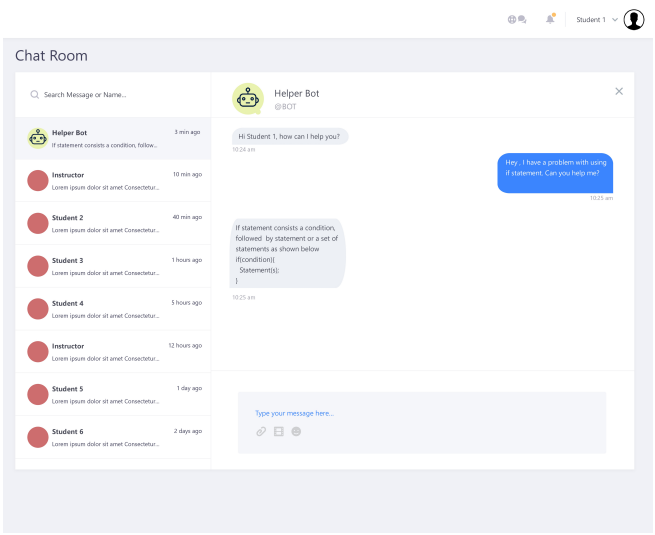

Figure 8. Student Calendar with assignment due dates retrieved from the LMS and Caring IDE.

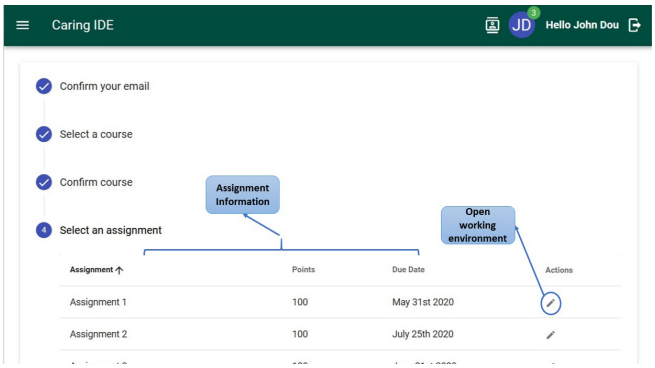

Figure 9. In the Assignment section, students can find list of assignments from auto-grader API.

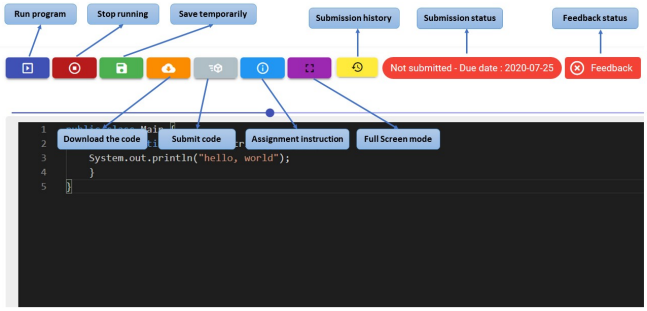

Figure 10. Explanation for the functionality of each action button in the developed IDE.

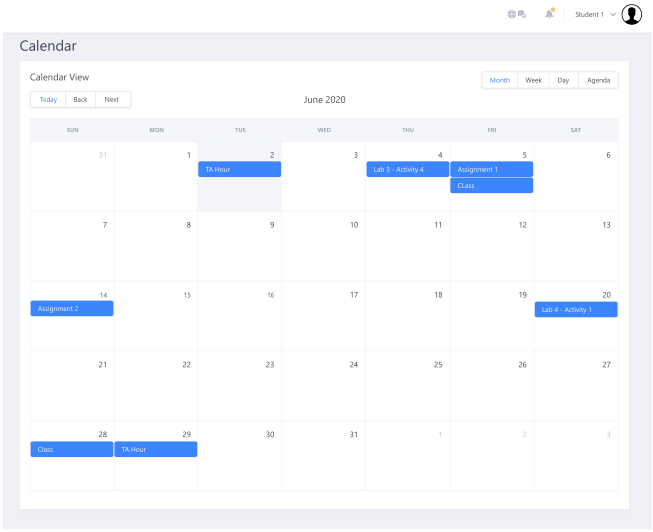

Figure 11. Student Calendar with assignment due dates retrieved from the LMS and Caring IDE.