Experiences with TA-Bot in CS1

Jack Forden
Marquette University
Milwaukee, WI, USA
jack.forden@marquette.edu

Alexander Gebhard
Marquette University
Milwaukee, WI, USA
alexander.gebhard@marquette.edu

Dennis Brylow
Marquette University
Milwaukee, WI, USA
dennis.brylow@marquette.edu

ABSTRACT

Automated Assessment Tools (AATs) have been used in undergraduate CS education for decades. TA-Bot, a modular AAT, has existed in some form for 25 years serving thousands of students across multiple universities. Class sizes throughout the last decade have continued to grow, while the number of instructors remains stagnant. AATs help instructors mitigate issues without additional resources, while simultaneously providing students with helpful feedback. The research team implemented novel features into the new, web-based TA-Bot such as dynamic rate limiting between submissions, custom code style feedback, and a gamified points system. The experiment discussed in this paper used TA-Bot over the course of three semesters involving 145 students in CS1. During the first semester, student and instructor feedback was collected on how to improve the tool. The second semester was used to rate limit submissions using a new dynamic rate limiting system. Finally, the third semester of TA-Bot was used as a control group with simple submission input/output checking. Instructors found that TA-Bot helped mitigate issues with continual increases in class sizes. When using TA-Bot with a dynamic rate limit, students were more inclined to start their assignment earlier. In addition to this, TA-Bot provides students with the ability to compare their solution against test cases, while simultaneously providing code-style advice using curated novice-friendly examples.

CCS CONCEPTS

• **Social and professional topics → Student assessment**; • **Applied computing → Learning management systems**.

KEYWORDS

automated assessment tools, gamification, unit testing, study behaviors, CS1

ACM Reference Format:
Jack Forden, Alexander Gebhard, and Dennis Brylow. 2023. Experiences with TA-Bot in CS1. In *Proceedings of the ACM Conference on Global Computing Education Vol 1 (CompEd 2023), December 5–9, 2023, Hyderabad, India.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3576882.3617930

1 PROBLEM AND MOTIVATION

For many computer science students taking a CS1 course, it may be their first experience with programming. In some universities (including the authors'), a CS1 course is also often attended by non-CS majors or offered as an elective for similar fields. This mix of students can lead instructors to walk a tightrope of course design. If it is too easy then students with previous programming experience are never challenged. If the pace is too fast it can leave newer students demotivated and may lead to an increase in dropout rates. Any increase in dropout rates should be a sign of concern as universities have historically struggled with retaining students.

According to Bennedsen and Caspersen the failure rate in CS1 courses in the United States is around 28% [5]. A multinational study found the global pass rate of CS1 to be around 75% [22]. The authors for both papers concluded that the pass rate is in line with other rigorous courses such as mathematics or physics, however both papers state that more work can be done to make CS1 courses accessible to a wider range of students. While the rigor of a course can play a part, there are any number of factors, such as stress, anxiety, time mismanagement, or inability to get help outside of class, which can result in students doing poorly. Enrollment is increasing in Computer Science classes, instructors have to manage the increase in students in their courses. This increase in enrollment consequently has decreased the time an instructor or TA might have for helping each individual student.

The goal of the updated version of TA-Bot is to improve the experience for instructors and students. For instructors, TA-Bot was designed to create a platform that helped monitor, track, grade, and most importantly, get useful insights into student coding habits, submission rates, and grammar styles.

Starting assignments earlier has been shown to improve student scores [23]. Individual feedback and attention to each student is also important to continue fostering good learning environments. While there are numerous existing AATs, these are often in-house projects, or platforms with a singular purpose in mind. AATs such as these offer little assistance for institutions that might wish to adopt them. While the goal is to facilitate the process of individual feedback, in the case that a TA or instructor is not readily available, TA-Bot should still be a platform that allows for flexibility, feedback, and advice for each student submission. It is important that students think of TA-Bot as an additional tool that can help them in the interim of instructor feedback, rather than a gatekeeper that increases the complexity of the course.

2 BACKGROUND AND RELATED WORK

Automated Assessment Tools (AATs) are used in courses to provide automated feedback on students' submissions. Students enjoy the feedback AATs provide, which allows them to modify their submissions before the deadline. Teachers also enjoy the freedom AATs provide by allowing them to focus their attention on other parts of instruction rather than the repetitive testing of numerous student submissions [20]. AATs often incorporate other tools, such as static code analysers, to further help students improve their work.

CompEd 2023, December 5–9, 2023, Hyderabad, India
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0048-4/23/12.
https://doi.org/10.1145/3576882.3617930

Using static code analyzers in computer science courses is not new, however recent work has been done to incorporate them into AATs for CS1 courses [8] [2] [17]. PyTA (a wrapper around a Python static code analysis tool called Pylint [3]) was integrated into weekly programming assignments [14]. PyTA gave students an opportunity to opt-in to a curated list of available explanations for specific Python errors. When compared to infrequent PyTA users, students who often used PyTA required fewer submissions to pass an exercise or to rectify the most common errors, and encountered less of the same repeating errors in their resubmissions.

More recent work has been done to integrate gamification into AATs in CS1/CS2 courses as well. Deterdign *et al.* define gamification as "*the use of design elements characteristic for games in a non-game context*" [7]. Call *et al.* introduced gamification into an ATT in a CS2 course [6]. Students in the course were placed into teams. The teams were awarded points by completing activities, such as passing test cases, submitting assignments early, and posting/responding to forum posts. A public leaderboard informed teams of their standing in the class. The top teams were awarded bonus points on an exam. Call *et al.* found that these activities motivated students to finish assignments earlier, follow the best version control practices, and help others on the class forums. The new web TA-Bot extends gamification by offering a new incentive enabling students to gain feedback more often on the third day.

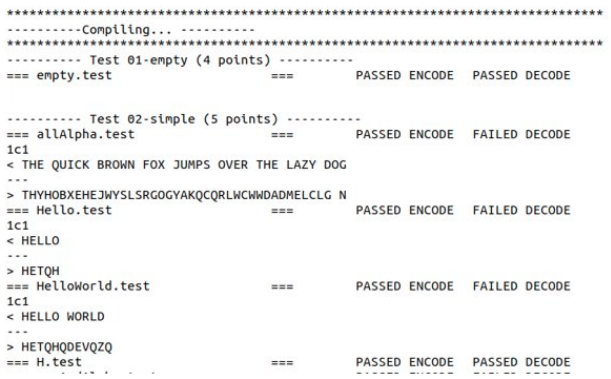
Another avenue of gamification was researched through the Marmoset Project [24], which aimed to improve student programmers experience. Marmoset used the interesting concept of "tokens". For each assignment a student was given 2 or 3 tokens, which when used, would allow the student to see curated feedback on which test cases they were passing or failing. A token would only be given back to a student 24 hours after use. TA-Bot builds on top of the rate limiting system found in the Marmoset Project by dynamically rate limiting based on the days until the assignment is due. This approach has recently been emphasized as an effective method for encouraging students to reflect on their submissions and decrease their dependence on automated grading. [13]

More recently, AAT's have implemented aspects of mobile gaming, such as limiting a player's time and reward systems, to cultivate game-based rewards [11]. The authors implemented a submission energy system for managing student submissions. A full charge consisted of three submissions (energy units). A student submission would consume an energy unit, which would regenerate after an hour. The outcome showed a modest reduction in student procrastination. The authors note that their submission energy limits might have been too lenient and that more work in this area needs to be done. TA-BOT aims to build on the authors' concept of rate-limiting student submissions through gamification with a harsher submission limiting system.

One of the most widely used AATs, Web-CAT [9], is built with the ideals of Test Driven Development (TDD) [4]. This is evident in one of the three main metrics Web-CAT uses. One of the three code analysis points is code correctness, which is solely based on student-written test cases. This score, combined with test completeness and test validity, is combined for a final composite score, which is relayed to students or a professor for further manual grading. Users can manually develop plugins which can significantly change how Web-CAT operates without needing to change the underlying code base. This design decision allows Web-CAT to be more flexible than a static closed system.

3 TA-BOT

TA-Bot originated from a series of shell scripts written at Purdue University. While initially developed to automatically compile and test student submissions in CS1, our institution frequently uses the tool in upper level courses. Command-line TA-Bot has also been adapted by others into a structured system that can run students' code on the Embedded Xinu operating system [15]. This iteration of TA-Bot was purely command-line based. Students would submit their work from a department machine to a centralized server. The student's submission is then stored in a directory awaiting execution. This iteration of TA-Bot batch compiled, tested, and created summary reports. Instructors used other Linux tools such as cron (usually scheduled for very early in the morning) to regularly schedule runs at selected intervals. Students were then emailed once all submissions were finished being reviewed. The resulting email included the contents of the files the student had submitted, whether the student passed or failed each test case, and (in the case that the student failed), the differing output of the expected output to the student's output (See Figure 1). This command-line TA-Bot has been used in testing the submissions of thousands of students at our university as well as others for the past 25 years.



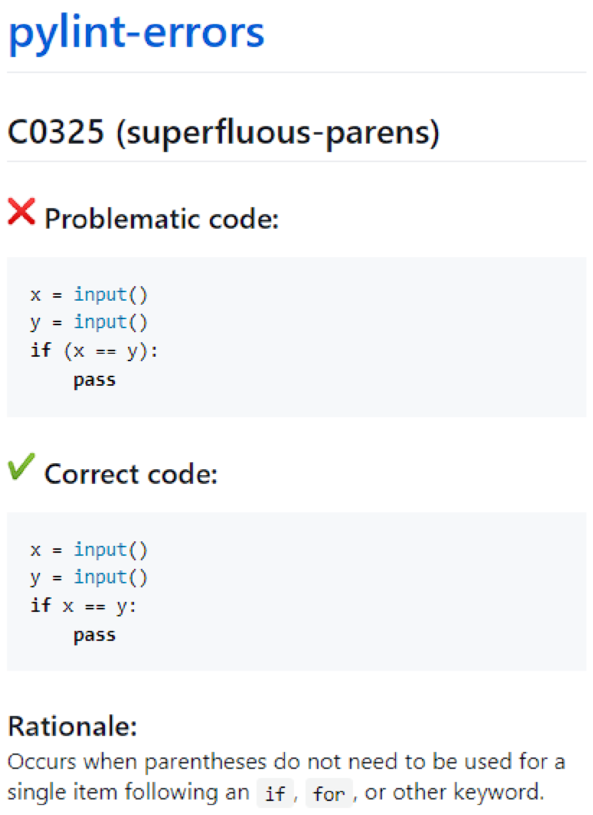
```
********************************************************************************
----------Compiling... ----------
********************************************************************************
---------- Test 01-empty (4 points) ----------
=== empty.test                     ===         PASSED ENCODE   PASSED DECODE


---------- Test 02-simple (5 points) ----------
=== allAlpha.test                  ===         PASSED ENCODE   FAILED DECODE
1c1
< THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
---
> THYHOBXEHEJWYSLSRGOGYAKQCQRLWCWWDADMELCLG N
=== Hello.test                     ===         PASSED ENCODE   FAILED DECODE
1c1
< HELLO
---
> HETQH
=== HelloWorld.test                ===         PASSED ENCODE   FAILED DECODE
1c1
< HELLO WORLD
---
> HETQHQDEVQZQ
=== H.test                         ===         PASSED ENCODE   PASSED DECODE
```

Figure 1: An image of a command-line TA-Bot email output

3.1 Web TA-Bot

The limitations of the legacy command line version of TA-Bot have become steadily more produced over time. As TA-Bot was composed of bash scripts, its automated run time is executed by setting up a time-based automated task. Setting up an extra run that is not scheduled requires a manual intervention from the instructor. Additionally, TA-Bot limits an instructor's ability to provide more detailed automated feedback to students. After the scripts are finished, instructors have the option to manually score and provide feedback before submissions are sent to students. This instructor feedback is limited to plain text. This limitation removes a multitude of possible feedback opportunities.

3.1.1 Time Between Submission (TBS). The goals of the new web-based TA-Bot are twofold: to incentivize students to start assignments earlier and to promote better programming practices in CS1


pylint-errors

C0325 (superfluous-parens)

❌ Problematic code:

```
x = input()
y = input()
if (x == y):
    pass
```

✓ Correct code:

```
x = input()
y = input()
if x == y:
    pass
```

Rationale:

Occurs when parentheses do not need to be used for a single item following an `if`, `for`, or other keyword.

Figure 2: An example of a custom pylint feedback error page

students. Starting good programming practices early saves professors from having to correct bad habits later. In order to decrease the significant learning curve for CS1 students, we chose to design this new iteration of TA-Bot with a user-friendly web interface as opposed to the traditional command-line TA-Bot. This web interface additionally allows the feedback to be significantly robust.

Although the command line TA-Bot typically ran once nightly to serve as a rate limit, the web-based TA-Bot runs with a far more dynamic rate limiting system. Pettit *et al.* demonstrated that implementing some form of rate limiting incentivized students to submit higher quality code [19]. TA-Bot's rate-limiting implementation is designed to serve as additional motivation to encourage earlier submissions. Spacco *et al.* found when students start an assignment earlier, their scores are higher [23]. Implementing a rate limit in an AAT is not a new endeavor. As mentioned previously, the Marmoset Project implemented a token-based rate limit system. Students were given 2-3 tokens that would renew 24 hours after use [24]. Athene [19] is an AAT that used a static rate limit of 15 minutes between submissions.

TA-Bot implements a novel concept called *time between submissions* (TBS) to incentivize earlier submissions. On the day the project is assigned, the TBS is set at five minutes. Once a student uploads and gets instant feedback, they are prevented from resubmitting until the 5 minute TBS cooldown period expires. Every day, as the assignment gets closer to the due date, the TBS is increased up to a maximum of two hours. If the student waits until the due date to get feedback, they will be prevented from submitting again until the 2 hour TBS is up. In the case that an assignment is handed out Monday and due the next Monday, a student who chose to start the assignment the Sunday before the assignment is due would have significantly fewer possible submissions than a student who took advantage of the significantly lower TBS at the start of the week.

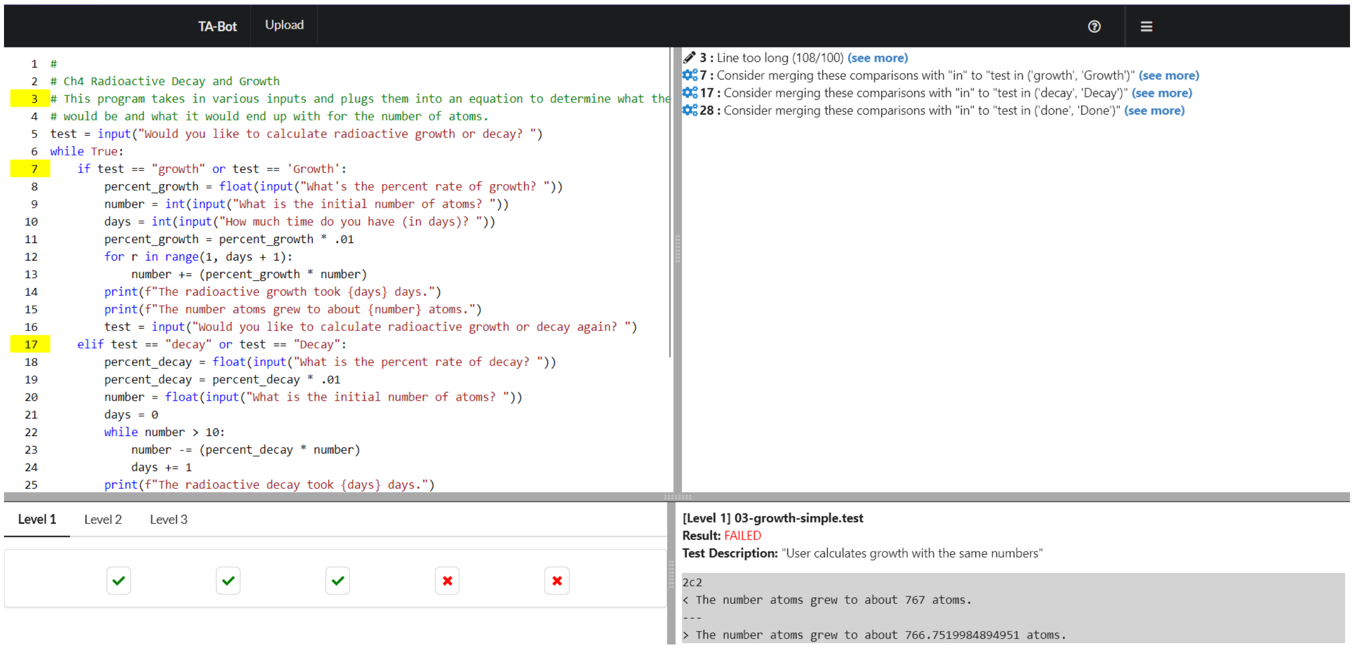
3.1.2 *Pylint and Gamification.*

In addition, TA-Bot implements static code analysis via Pylint [3]. While Pylint helps identify stylistic errors in student code, it also adds another level of difficulty for CS1 students. Pylint errors can be complex and confusing at times. With this in mind, we extended a popular repository by Vladyslav Krylasov that contained explanations and examples of common Pylint errors [12]. We modified our own version to be more accommodating for CS1 students, with detailed examples of common errors and their respective solutions. In the results page, the line that causes the Pylint error is highlighted (see Figure 3), and both the error itself and a link to an explanation page is provided. The page includes text-based rationale to explain the suggestion and why Pylint flagged it (see Figure 2). The errors displayed to students are a curated subset of the total possible Pylint suggestions. The justification for creating this subset was to avoid possible suggestions that are outside of the scope of a CS1 course.

The number of test cases that the student passes, as well as the number of Pylint suggestions present, are combined to calculate an overall score. This score is out of a maximum of 100 points. 60 points were from the test cases, with each test case weighted evenly. 40 points came from Pylint suggestions. If a student had 5 or fewer Pylint suggestions, they received the full 40 points dedicated to Pylint. The number of points the student receives for Pylint decreases as Pylint offers more suggestions. If a student had more than 10 Pylint errors, they could only receive 10 points for the Pylint portion of the score. When the Pylint score and test case score combined is greater than 75 points on their final submission from the previous assignment, then students have the opportunity to freeze the TBS on the third day of a week-long assignment. Instead of the TBS being 45 minutes on the third day, students will get a TBS of 5 minutes. Our hypothesis is that this motivates students to minimize the number of Pylint errors in their code and maximize the number of test cases passed. It is important to note that the resulting TA-Bot score are not automatically converted to grades; human graders still follow a more general rubric for assigning points.

3.1.3 *Leveling System.*

Another avenue for implementing gamification into TA-Bot arose when implementing our leveling feature. TA-Bot takes a novel approach to gamification with a unique leveling system that does not appear to correspond to features in any other AATs attested in the research literature. Test cases for each weekly project were separated into different levels, with each level featuring progressively more sophisticated test scenarios. All students start on Level 1. Students are not shown the test cases for the next level until they pass at least half of the test cases on their current level. Starting large, week-long assignments can be daunting for CS1 students. TA-Bot encourages students to start with the basic test cases and gradually work their way up to the more advanced test cases in a TDD fashion.



```
 1 #
 2 # Ch4 Radioactive Decay and Growth
 3 # This program takes in various inputs and plugs them into an equation to determine what the
 4 # would be and what it would end up with for the number of atoms.
 5 test = input("Would you like to calculate radioactive growth or decay? ")
 6 while True:
 7     if test == "growth" or test == 'Growth':
 8         percent_growth = float(input("What's the percent rate of growth? "))
 9         number = int(input("What is the initial number of atoms? "))
10         days = int(input("How much time do you have (in days)? "))
11         percent_growth = percent_growth * .01
12         for r in range(1, days + 1):
13             number += (percent_growth * number)
14         print(f"The radioactive growth took {days} days.")
15         print(f"The number atoms grew to about {number} atoms.")
16         test = input("Would you like to calculate radioactive growth or decay again? ")
17     elif test == "decay" or test == "Decay":
18         percent_decay = float(input("What is the percent rate of decay? "))
19         percent_decay = percent_decay * .01
20         number = float(input("What is the initial number of atoms? "))
21         days = 0
22         while number > 10:
23             number -= (percent_decay * number)
24             days += 1
25         print(f"The radioactive decay took {days} days.")
```

3 : Line too long (108/100) **(see more)**
7 : Consider merging these comparisons with "in" to "test in ('growth', 'Growth')" **(see more)**
17 : Consider merging these comparisons with "in" to "test in ('decay', 'Decay')" **(see more)**
28 : Consider merging these comparisons with "in" to "test in ('done', 'Done')" **(see more)**

[Level 1] 03-growth-simple.test
Result: FAILED
Test Description: "User calculates growth with the same numbers"

```
2c2
< The number atoms grew to about 767 atoms.
---
> The number atoms grew to about 766.7519984894951 atoms.
```

Figure 3: An image of web-based TA-Bot showing an example submission

3.1.4 Design. The design of the new, web-based TA-Bot page (Figure 3) emphasizes simplicity. We have chosen to display the submitted code at the top left of the page, with line numbers that can be highlighted when Pylint detects a suggestion. These Pylint suggestions are displayed in greater detail at the top right, with their respective line numbers in bold text. Each Pylint suggestion has an accompanying “See more” hyperlink which directs a student to a simplified code example relevant to the suggestion encountered. Student test case results were displayed in the bottom left. Each green check or red X is clickable, allowing students the ability to get more information on the test case. If the student failed a test case, a standard Linux diff [10] shows the student the difference between their output and the expected output. To navigate from one level to another, a student would simply click on “Level #”.

3.1.5 Behind the Scenes: Web TA-Bot. Web-based TA-Bot uses a back-end API implemented in Python using the Flask library [18]. Due to its popularity and student familiarity, Python was chosen as the new programming language for TA-Bot. (Command-line TA-Bot relies on several scripting languages, including Bash and Expect.) The front-end UI is written in Typescript with UI components coming from Sementic UI [16]. The front-end UI communicates with the back-end API via a series of JavaScript web requests.

Once a student uploads code, it is stored in a staging directory awaiting execution. Every TA-Bot project has a series of expected outputs for a given input (i.e. if this input is entered in student code, then this output should be printed). These are stored in separate text files on the TA-Bot server. A separate execution is performed for each input. The student code is run in a sandbox to minimize the security and performance impact on the TA-Bot server via an application called Piston [21]. Piston is a Docker-based code execution engine that runs code inside containers. Piston reports the output of the code, which is then used to compare against the expected output. If the output from the student’s code matches the expected output, then the student passes the test case and receives appropriate credit. If the outputs do not match or an error occurs when Python tries to execute the file, then the student fails the test case. A standard Linux diff is used to show the student the difference between their output and the expected output.

TA-Bot, as with most other AATs mentioned in this paper, uses this expected output vs. student output to determine if a student passes the test case. TA-Bot does support the ability to run other types of tests, such as indeterministic tests, where the system cannot automatically determine if the output is correct. With indeterministic tests, instructors need to manually review if the output is correct. This type of test is common in programs where multiple threads are running, since the order in which the threads complete cannot be determined. Wilcox provides an overview of possible test types that AATs support [25]. TA-Bot supports output analysis by comparing the student’s output to an instructor’s expected output. TA-Bot doesn’t support reflection, object instantiating, and instrumentation. Test case results as well as user account information are stored in a MySQL server. In addition, teachers have the ability to submit the most recent submission from all students for a given project to MOSS [1], with the results emailed once they are completed. TA-Bot is an open-source and readily accessible tool for any university considering its adoption. To facilitate ease of adoption, TA-Bot has been constructed within Docker containers, streamlining the installation process for all required packages. To set up your own TA-Bot instance, you can follow the simple steps provided in the documentation available at: https://github.com/JForden

4 EXPERIMENT DESIGN

Our new incarnation of TA-Bot was introduced in a summer 2021 CS1 course for new, first-generation college students. This course is designed to motivate and enable low-income and first-generation students to enter and succeed in higher education. During the course, we gathered informal feedback from students to improve the tool before its full release. TA-Bot made its full roll out in the Fall 2021 and Spring 2022 semesters in our CS1 course. The Fall 2021 semester had 2 primary instructors with a total of 100 students enrolled across both instructors. The Spring 2022 semester had 44 students enrolled with one primary instructor.

With the implementation of TA-Bot, we created ten weekly assignments (both semesters used the same assignments) based on chapters from the class. Students were given an assignment based on class content the week after they covered the content in class. Assignments were released on Monday morning, and students had until the following Monday to submit their work. TA-Bot did not accept submissions after the deadline.

To determine the effectiveness of TA-Bot, TA-Bot stored the following data every time a student made a submission: date & time, submitted code, Pylint output, and which test cases the student passed or failed. In the Fall 2021 semester, the leveling system, time between submission (TBS), and the ability to unlock a lower TBS on the third day of the assignment were all enabled. We used Spring 2022 as our control semester. During this semester, students could submit as often and as frequently as they wanted. We did have a five minute cooldown between a student's submissions in order to not overload the server. The leveling system was also disabled, so students could see all test cases from their first submission. We gathered and analyzed student feedback, instructor feedback, and submission data from both semesters.

5 RESULTS AND DISCUSSION

5.1 Pylint

While instructors found the Pylint feature useful as an additional indicator of student progress, students were often confused by the meaning of errors and how to resolve them. Although all possible Pylint errors were reviewed before the semester started, some Pylint errors still caused confusion amongst students. These often resulted from Pylint suggestions that were outside the scope of what students had learned up to that point in the semester. This demonstrates a downside in the errors we chose to document. Unlike PyTA, which used a curated list of Pylint errors, we used a more complete repository containing hundreds of Pylint errors. Before the semester started, we disabled a handful of Pylint errors that weren't relevant for the class. Errors related to topics not covered in class or errors that did not get at the goal of making students better programmers (such as spell checking) were disabled. Nonetheless, during the Fall 2021 semester, we had to disable errors that were generating too many false positives or were overly confusing to students. One such error we disabled during the Fall 2021 semester was Pylint error C0103, triggered whenever a constant is not in all uppercase letters. This error generated too many false positives and decreased student gamification scores. Students also often got multiple suggestions from Pylint on each submission, leading to students feeling overwhelmed and thus ignoring the suggestions completely. Going forward, we look to further refine the list of errors enabled to minimize student confusion.

5.2 Time Between Submission

After interviewing students and instructors, a common theme that emerged was negative student impressions of TBS. This was expected, as TBS encouraged students to work on the assignments earlier in the week, and could be perceived as penalizing those who waited until later. Due to many students being on a cooldown period when the submission deadline passed, instructors were sometimes asked to upload final submissions on behalf of the students. An example of this is a student uploading their submission at 7 a.m. with a final deadline of 8 a.m. A student would realize they still failed test cases, yet were unable to resubmit their assignment due to the TBS. Students made the argument that it was unfair to prevent further submissions as the assignment was technically due at 8 a.m. While this is a deliberate design decision, a clear flaw in this feature is its reliance on instructors being willing to enforce hard deadlines. Allowing a final, manual submission undermined the TBS system and often resulted in a final submission that had been lightly modified and untested. A common complaint from students was that other commitments prevented them from working on assignments when the TBS period was low. Student opinion was that a higher TBS later in the week was a punishment rather than an incentive to start earlier. When TBS was not enabled during the Spring 2022 semester, instructor and student feedback was more positive about uploading at their convenience.

The heatmaps show the number of students who have made their first TA-Bot submission for the weekly assignment on a given day for the Fall 2021 (see Figure 4) and Spring 2022 (see Figure 5) semesters. Each column represents one week-length assignment. Columns with all zeros represent a week where no homework was assigned. Students who made a submission early Monday morning (before the assignment was due at 8 a.m.) were included in the Sunday count to not interfere with the next week's data. In the Fall 2021 semester, there is noticeably more students starting the first three days when TBS was enabled. When TBS was disabled in the Spring 2022 semester, students tended to wait until after lab on Day 4 to start. This demonstrates that TBS did push some students to start earlier in the week when they could submit with a smaller cooldown. Even though students viewed TBS as a punishment for starting later, it did appreciably alter student submission habits.

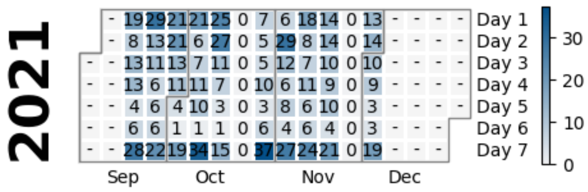


Figure 4: Day of students' first submission with TBS enabled

5.3 Gamification

At the start of the fall semester, the ability to unlock a submission was only possible on the third day. Due to instructor and student

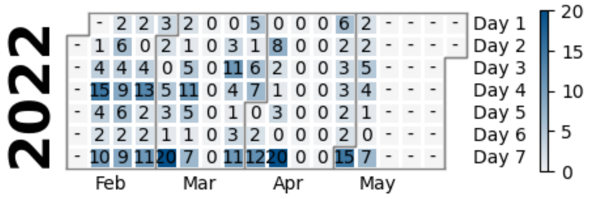


Figure 5: Day of students' first submission with TBS disabled

feedback, we changed this design to allow students to pre-unlock the shortened TBS (still redeemed on the third day) from the moment the weekly assignment was activated. After looking into the data, the ability to "unlock" an extra day of shortened TBS was redeemed 43 total times by 26 unique students. After talking to both students and instructors, it became clear that there was general confusion about the feature. TAs began deferring questions to instructors, who in turn passed them to the research team. Another issue arose from the course structure. As this course was co-taught, some labs were on Tuesday, while others were on Thursday. Some students in the Thursday section felt that the week day chosen was too early in the week, as students often waited to start until after their weekly lab. In the future, rewards for a high score could be modified to include multiple reward options.

5.4 Leveling System

What we learned from the deployment of our leveling system was the importance of deliberately designing coursework that plays to the strengths of the Web TA-Bot system. The weekly assignments early in the semester were too simplistic, resulting in bimodal results with students failing most test cases or immediately passing all test cases on each level. While the goal was to promote breaking a large weekly assignment into smaller chunks, students often attempted to finish the whole assignment before making a submission. Even as more complex assignments used base and edge cases, students felt the level system was an additional hurdle rather than a road map that encouraged good coding practices. The leveling system might be better suited for CS2/CS3 courses that have assignments with multiple parts. In addition, students in CS2/CS3 courses will be more familiar with the development process, so the leveling might not be as overwhelming.

5.5 Instructor Feedback

At the end of the semester, instructors were interviewed, and they were asked a series of questions using a Likert scale for their responses. The responses to these questions were a numeric value between 1 and 5, which correlated to: *strongly disagree*, *disagree*, *neither agree nor disagree*, *agree*, or *strongly agree*. Questionnaire results indicated that instructors felt TA-Bot significantly helped reduce administrative tasks. Instructors reported that reviewing students code was less time-consuming than in previous semesters, with an average response of *strongly agree*. The respondents noted that the administrator view saved significant time by allowing instructors to view students code without needed to manually download and run the script from the course management tool. Similarly, instructors responded that they *strongly agree* that TA-Bot helped identify issues in students code more quickly. Instructors *strongly agree* that TA-Bot helped them manage a larger number of students than in prior semesters without TA-Bot.

5.6 Student Feedback

Common themes in feedback gathered from students centered on TA-Bot output formatting and TBS. Students struggled to understand issues in their output when looking at the standard Linux diff. Detail-oriented issues such as spelling mistakes in output statements occasionally required TA intervention before they were resolved. Instructors also noted that they received a significant number of complaints about the TBS version of TA-Bot. Some students expressed their view that the increase in TBS punished students who did not start earlier. Conversely, students enjoyed the instant feedback TA-Bot provided without TBS enabled.

6 CONCLUSION

With Computer Science class enrollments in our institution quadrupling over the past decade, and the national aggregate CS1 dropout rate hovering around 30%, tools such as TA-Bot are needed to improve student and instructor experiences. This paper presents novel features added to an existing, well-tested AAT system affectionately codenamed "TA-Bot" by local students; new aspects included a testcase complexity leveling system, a dynamic Time Between Submissions (TBS) timer, and a gamified implementation of Pylint code style analysis. This improved model of TA-Bot can provide students with immediate feedback on their project code correctness and style without an instructor present, while also incentivizing students to start their week-length programming projects earlier. For instructors, TA-Bot reduced the time and effort of testing student code, freeing them to concentrate on other pedagogical issues not as amenable to automation.

7 FUTURE WORK

The new, web-based TA-Bot involved several different pedagogical changes, each of which warrants further research. While TA-Bot helped reduce administrative tasks, it occasionally caused confusion when students did not understand aspects of the system, such as the Linux diff output format. We plan to draw from UI/UX research to implement a diff using more visual elements such as text, color, or images. We hypothesize that by simplifying the output, it might help students understand their errors more quickly. Additionally, we plan on experimenting with different types of rate limiting to further research the TBS timer system. This rework could involve the TBS timer being tied to the number of submissions rather than the date. By basing the cooldown on the number of submissions, we hope to see students make higher quality submissions. Finally, we plan on changing the methodology for how Pylint results are scored. Rather than taking away points whenever Pylint errors occur, students could be rewarded for fixing Pylint errors after their original submission.

REFERENCES

[1] Alex Aiken. 1994. *Moss (for a Measure Of Software Similarity).* http://theory.stanford.edu/~aiken/moss/

[2] Kirsti Ala-Mutka, Toni Uimonen, and Hannu-Matti Järvinen. 2004. Supporting Students in C++ Programming Courses with Automatic Program Style Assessment. *JITE* 3 (01 2004), 245–262. https://doi.org/10.28945/300

[3] Python Code Quality Authority. 2022. *Piston.* Retrieved August 19, 2022 from https://github.com/PyCQA/pylint

[4] Kent Beck. 1999. *Extreme Programming Explained.* Addison-Wesley.

[5] Jens Bennedsen and Michael E. Caspersen. 2019. Failure Rates in Introductory Programming: 12 Years Later. *ACM Inroads* 10, 2 (apr 2019), 30–36. https://doi.org/10.1145/3324888

[6] Tristan Call, Erik Fox, and Gina Sprint. 2021. Gamifying Software Engineering Tools to Motivate Computer Science Students to Start and Finish Programming Assignments Earlier. *IEEE Transactions on Education* (2021), 1–9. https://doi.org/10.1109/TE.2021.3069945

[7] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. 2011. From Game Design Elements to Gamefulness: Defining "Gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments* (Tampere, Finland) *(MindTrek '11)*. Association for Computing Machinery, New York, NY, USA, 9–15. https://doi.org/10.1145/2181037.2181040

[8] Stephen H. Edwards, Nischel Kandru, and Mukund B.M. Rajagopal. 2017. Investigating Static Analysis Errors in Student Java Programs. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 65–73. https://doi.org/10.1145/3105726.3106182

[9] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 328. https://doi.org/10.1145/1384271.1384371

[10] Free Software Foundation. 2022. *GNU Operating System.* Retrieved August 19, 2022 from https://www.gnu.org/software/diffutils/

[11] Michael S. Irwin and Stephen H. Edwards. 2019. Can Mobile Gaming Psychology Be Used to Improve Time Management on Programming Assignments?. In *Proceedings of the ACM Conference on Global Computing Education* (Chengdu,Sichuan, China) *(CompEd '19)*. Association for Computing Machinery, New York, NY, USA, 208–214. https://doi.org/10.1145/3300115.3309517

[12] Vladyslav Krylasov. 2022. *Pylint Errors.* Retrieved August 19, 2022 from https://github.com/vald-phoenix/pylint-errors

[13] Juho Leinonen, Paul Denny, and Jacqueline Whalley. 2022. A Comparison of Immediate and Scheduled Feedback in Introductory Programming Projects. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 885–891. https://doi.org/10.1145/3478431.3499372

[14] David Liu and Andrew Petersen. 2019. Static Analyses in Python Programming Courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 666–671. https://doi.org/10.1145/3287324.3287503

[15] Matthew H. Netkow and Dennis Brylow. 2010. Xest: An Automated Framework for Regression Testing of Embedded Software. In *Proceedings of the 2010 Workshop on Embedded Systems Education* (Scottsdale, Arizona) *(WESE '10)*. Association for Computing Machinery, New York, NY, USA, Article 7, 8 pages. https://doi.org/10.1145/1930277.1930284

[16] Semantic Organization. 2022. *Semantic UI.* Retrieved August 19, 2022 from https://github.com/Semantic-Org/Semantic-UI

[17] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 22, 3, Article 34 (jun 2022), 40 pages. https://doi.org/10.1145/3513140

[18] Pallets. 2022. *Flask.* Retrieved August 19, 2022 from https://github.com/pallets/flask

[19] Raymond Pettit, John Homer, Roger Gee, Susan Mengel, and Adam Starbuck. 2015. An Empirical Study of Iterative Improvement in Programming Assignments. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 410–415. https://doi.org/10.1145/2676723.2677279

[20] Raymond Pettit and James Prather. 2017. Automated Assessment Tools: Too Many Cooks, Not Enough Collaboration. *J. Comput. Sci. Coll.* 32, 4 (April 2017), 113–121.

[21] Brian Seymour. 2022. *Piston.* Retrieved August 19, 2022 from https://github.com/engineer-man/piston

[22] Simon, Andrew Luxton-Reilly, Vangel V. Ajanovski, Eric Fouh, Christabel Gonsalvez, Juho Leinonen, Jack Parkinson, Matthew Poole, and Neena Thota. 2019. Pass Rates in Introductory Programming and in Other STEM Disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19)*. Association for Computing Machinery, New York, NY, USA, 53–71. https://doi.org/10.1145/3344429.3372502

[23] Jaime Spacco, Davide Fossati, John Stamper, and Kelly Rivers. 2013. Towards Improving Programming Habits to Create Better Computer Science Course Outcomes. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (Canterbury, England, UK) *(ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 243–248. https://doi.org/10.1145/2462476.2465594

[24] Jaime Spacco, William Pugh, Nat Ayewah, and David Hovemeyer. 2006. The Marmoset Project: An Automated Snapshot, Submission, and Testing System. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA) *(OOPSLA '06)*. Association for Computing Machinery, New York, NY, USA, 669–670. https://doi.org/10.1145/1176617.1176665

[25] Chris Wilcox. 2016. Testing Strategies for the Automated Grading of Student Programs. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 437–442. https://doi.org/10.1145/2839509.2844616