Using Daily Missions to Promote Incremental Progress on Programming Assignments

Andrew Goldman

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Stephen H. Edwards, Chair
Clifford A. Shaffer
Donald S. McCrickard

May 10, 2019
Blacksburg, Virginia

Keywords: Gamification, Automatic Assessment Tools, Computer Science Education
Copyright 2019, Andrew Goldman
Using Daily Missions to Promote Incremental Progress on Programming Assignments

Andrew Goldman

(ABSTRACT)

Automatic assessment tools are increasingly utilized in undergraduate programming courses to evaluate software solutions, streamlining the grading process for both students and professors. In spite of their benefits of speed and convenience, such online systems for providing instant feedback have the tendency to draw attention to performance-based outcomes while failing to reliably recognize the effort and hard work a student puts into a solution. For the many struggling students who are new to programming, this type of objective feedback can be discouraging and may decrease their motivation to stay engaged towards success. To address this issue, this paper explores strategies for more effectively recognizing student progress on programming assignments and identifying small tasks for students to complete that will steer them in the right direction. Further, this paper will discuss a gamification approach for adding “Daily Missions” to Web-CAT, the Web-based Center for Automated Testing. From an evaluation of results gathered from student experiences with this latest version of Web-CAT at Virginia Tech, this paper will highlight the valuable potential gamification has in boosting student engagement in computer science learning.
Using Daily Missions to Promote Incremental Progress on Programming Assignments

Andrew Goldman

(GENERAL AUDIENCE ABSTRACT)

In college computer science courses today, many students submit their assignments and receive instant feedback through online websites. Although they streamline the grading process for both students and professors, these systems have the tendency to draw attention to performance-based outcomes while failing to reliably recognize the effort and hard work a student puts into a solution. For the many struggling students who are new to programming, this type of objective feedback can be discouraging and may decrease their motivation to stay engaged towards success. To address this issue, this paper explores strategies for more effectively recognizing student progress on programming assignments and identifying small tasks for students to complete that will steer them in the right direction. One strategy in particular is called gamification, which refers to giving game-like attributes to a non-game system. This paper will discuss a gamification approach for adding “Daily Missions” to Web-CAT, the Web-based Center for Automated Testing. From an evaluation of results gathered from student experiences with this latest version of Web-CAT at Virginia Tech, this paper will highlight the valuable potential gamification has in boosting student engagement in computer science learning.
Acknowledgments

This work is supported in part by the National Science Foundation under grant DUE-1625425. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.
Contents

List of Figures viii

1 Introduction 1

1.1 Challenges of Computer Science Learning 1
1.2 Overview of Web-CAT and Objectives 3
1.3 Adopting the Growth Mindset 3
1.4 Research Questions 5
1.5 Outline 6

2 Related Work 7

2.1 Factors Impacting Programming Performance 7

2.1.1 Time Management 8

2.1.2 Sub-Tasking 9

2.1.3 Code Modularity 11

2.1.4 Self-Testing 12

2.1.5 Documentation 13

2.2 Trends Visualized by Student Submission Data 14

2.3 Goals of Automated Assessment 20
## 2.4 Progress Indicators

### 2.5 Gamification Schemes

- **2.5.1 Self-Monitoring**
- **2.5.2 Rising to Challenges**
- **2.5.3 Persistence**
- **2.5.4 Operant Conditioning**

## 3 Daily Missions Design Goals

- **3.1 Formal Definition**
- **3.2 Attributes and Criteria**
- **3.3 Rewards**

## 4 Daily Missions Implementation and Display

- **4.1 Selection Logic**
- **4.2 Visual Design**
- **4.3 Use Case Walkthrough**
  - **4.3.1 Missions Generated on First Submission**
  - **4.3.2 Completing a New Mission**
  - **4.3.3 Completing a Mission With a Score Impact**
  - **4.3.4 Completing Multiple Missions At Once**

## 5 Evaluation
List of Figures

2.1 Line graph showing a progression over the project timeline of students who have made at least one submission ............................................. 16

2.2 Scatterplot that arranges data points by time of first submission and final correctness percentage ................................................... 17

2.3 Scatterplot that arranges data points by final correctness percentage and average code coverage across all submissions ........................ 18

2.4 Snippet of a spreadsheet used to aid the phrasing of comments given by Web-CAT’s virtual teaching assistant ........................................... 24

2.5 Screenshot of a reinforcing comment given by Web-CAT’s virtual teaching assistant [40] ................................................................. 25

2.6 Screenshot of an encouraging comment given by Web-CAT’s virtual teaching assistant [40] ................................................................. 25

2.7 Interest over time chart for the word “gamification” in Google Trends ................................................................. 26

2.8 Gamified display of student progress in an online programming course developed at the University of Turku [24] .............................. 28

4.1 View of the submission energy bar and Daily Missions button ................... 41

4.2 View of the modal window appearing when students click the Daily Missions button ................................................................. 42

4.3 Modal window that lists the new day’s missions to be completed .............. 44
4.4 Modal window that reflects the completion of one new mission . . . . . . . 45

4.5 Modal window that indicates an additional mission just completed as well as
its impact on the student’s score ............................................. 46

4.6 Modal window that reflects the completion of two new missions and a further
increase in score ...................................................................... 47

5.1 Number of students who saw each mission and percentage of students who
completed them ...................................................................... 50

5.2 Line graph showing the progression of students who have made at least one
submission from both the Fall 2017 and Spring 2019 semesters ......... 52

5.3 Survey used to measure student perceptions of the Daily Missions ...... 54

5.4 Visualization of survey responses to statements centered on positive outcomes 55

5.5 Visualization of survey responses to statements centered on negative outcomes 56
Chapter 1

Introduction

Computer Science today can be a challenging major. In the context of national statistics, dropout rates for students attempting computer science courses are exceptionally high—according to research published by the Higher Education Authority, about one-third of computer science students failed to progress beyond their first year between 2012 and 2014. Students embarking on degrees in this field are faced with a brand new set of principles that show little resemblance to the type of course material they are typically used to, giving many the impression of learning a foreign language. Contrary to disciplines that build on prior subject matter and traditional learning models, computer science introduces vast libraries of unfamiliar functions along with complicated syntax and logic rules that require thorough practice to understand. For beginners, this massive new set of tools can be overwhelming.

1.1 Challenges of Computer Science Learning

The most widely used educational model for teaching programming to beginners is the object-first approach. This paradigm places an emphasis on viewing software as a collection of objects that communicate with each other and take on different states, also referred to as object-oriented design. Students learning this structure for the first time immediately dive into the use of public and private data members, encapsulation, inheritance, polymorphism,
constructors, accessors, and modifiers. All of these are complex and abstract concepts that make for a steep learning curve, requiring significant time to get used to.

In addition to the difficulty of acclimating to a new set of mechanics, syntax, and environments, introductory software design courses add a layer of technical application that poses a real challenge for many students. Unlike courses that task students with recalling information on a test or solving a set of problems given initial conditions, the deliverables required of computer science students involve synthesizing many tools to construct a program that invokes some specified behavior. These project-based exercises often entail repeated trial and error before concepts can be fully mastered.

The first major-specific courses for aspiring computer scientists typically involve fundamental programming principles and application in coding projects, where errors and obstacles are likely to surface for those who have no coding experience. One of the biggest sources of frustration among students trying to complete these assignments is not knowing how to fix the problems in their code. Programming exercises are often submitted through electronic grading systems that run tests on student code and report results, meaning that most of the time there is no human assessor to pinpoint what needs to be fixed. This lack of guidance in frequent times of failure has presented an opportunity to enhance automatic grading tools with features that enable students to better understand their errors and realize what they need to do to improve.

From a pedagogical standpoint, assessment tools can be hard to design in a way that accurately measures student effort. This difficulty is due primarily to the all-or-nothing nature of assessing a program’s functional behavior. Evaluating the correctness of a piece of software often translates to testing whether or not it can perform certain tasks, which makes it easy for generated feedback to lack descriptiveness and direction for the student. To address these issues, web applications to which students can submit code and receive
automatic results have been evolving to provide more helpful, qualitative feedback.

1.2 Overview of Web-CAT and Objectives

This paper will motivate and detail new features deployed to Web-CAT, the Web-based Center for Automated Testing. Web-CAT is a system originally developed by Stephen Edwards of Virginia Tech, allowing students to submit many versions of a software project over an allotted time frame and receive a comprehensive summary of results. The users of this system are primarily college underclassmen. The ongoing goals of Web-CAT are to condition students to follow the paradigms of object-oriented programming, provide feedback that helps students identify their issues, and instill in students an optimistic mindset about their ability to learn how to build software. This paper will pinpoint specific areas that tend to be sources of struggle for computer science students and explore principles for incremental progress that can improve these weaknesses. To promote a more frequent practice of these principles among students, this paper will propose a gamification strategy called “Daily Missions”, which are a series of small challenges offered to students every 24 hours that describe manageable software improvements. Finally, this paper will evaluate the effectiveness of these Daily Missions based on comparisons with prior data and analysis of student-provided feedback.

1.3 Adopting the Growth Mindset

The incentive behind crafting automatic assessment tools to make them more useful and beneficial for users lies in the theory that all students are capable of acquiring programming skill if motivated properly. In particular, newer features of Web-CAT seek to encourage
a growth mindset, which establishes that intelligence is not inherited in rigid types and quantities but is rather malleable beyond limit through persistent effort and hard work [12]. Having a growth mindset is important in courses with a steep learning curve such as software design because it promotes the idea that any student can overcome challenges and succeed, as long as they persevere through the learning process and stay engaged.

When trying to pinpoint causes for student failure, many believe in a fixed mindset, which holds that intelligence is a predetermined quality that cannot be changed. Among professors around the country, the cohort that adopts this mentality is surprisingly large. In a 2007 study conducted by Clayton Lewis, 77 percent of polled computer science faculty disagreed with the statement “Nearly everyone is capable of succeeding in the computer science curriculum if they work at it” [29]. It is plausible that this percentage has decreased since then due to the evolving outlooks and strategies of computer science educators, but the fact remains that many people have this view. As a consequence, a large portion of students feel discouraged by their failures rather than motivated to keep trying.

The existence of a fixed mindset calls for an ongoing need to promote the opposite philosophy with educational models and tools. As Stanford University psychologist Carol Dweck who coined the term Growth Mindset defines, “In a growth mindset, people believe that their most basic abilities can be developed through dedication and hard work — brains and talent are just the starting point. This view creates a love of learning and a resilience that is essential for great accomplishment.” [11]. The frame of mind illuminated by this description acknowledges the potential for success in any discipline and by any willing individual, and educators have the power to ingrain this mentality in their students. The questions, then, become: How can students be migrated from a fixed mindset to a growth mindset? How effective can this transition be in improving students’ chances of success?

An exploration of recent growth mindset studies helps to answer these questions. In
2018, Katherine Carl Payne, Jeffry Babb, and Amjad Abdullat conducted an experiment in which they observed the effects of revamping the structure of classroom lectures in an effort to foster a higher rate of growth mindsets [38]. Before the restructuring, lectures were primarily given with long powerpoint presentations from which students were expected to absorb information and apply it on their own in programming exercises. Students who struggled to understand the material often did not even attempt the coding exercises after looking at them, indicating that many did not think they were capable. The changes that were made to the lecture style introduced a period of time during which students were given small coding exercises and could ask questions and receive feedback. This problem-driven approach to programming lessons had a positive impact on the atmosphere of the class and the general performance of the students. Those who previously left coding editors blank were now making reasonable attempts at writing solutions, and scores for the exercises increased.

This experiment demonstrates a useful strategy for drawing out the capabilities many students have but are not driven to uncover due to pedagogical deterrents. Especially with learning a brand new skill like programming that beginners find challenging, it is important to have mechanisms that give students easy access to feedback. Later sections of this paper will delve into the specific ways that feedback can be administered to encourage students’ capacity to think and work hard.

1.4 Research Questions

Adopting the growth mindset as a tool accessible to any student leads to an optimistic view of why students struggle. Instead of dismissing the students who don’t achieve highly as being incapable of success, the door is open to investigating weaknesses as ones that students can overcome. With this mentality, poor performance is attributable to reasons
other than simply a lack of raw intelligence. In this paper, the concept of Daily Missions is intended to identify methods of improvement for students that are independent of differences in aptitude. In order to determine the feasibility of this idea, the following questions must be answered:

1. Can a system of gamified Daily Missions cause students to make their initial program submissions earlier?

2. Can Daily Missions condition students to make incremental progress on programming assignments?

3. Can Daily Missions raise feelings of motivation to work on assignments more frequently and help students identify ways of improving their code?

1.5 Outline

To motivate the design of Daily Missions that will answer the research questions listed above, the next chapter explores factors impacting programming performance, strategies for recognizing incremental progress, and theories in gamification. Following this review of related work, design goals for the Daily Missions are laid out in Chapter 3 and the implementation is described in Chapter 4. The final two chapters detail the evaluation mechanisms used to assess the effectiveness of the Daily Missions, an analysis of the results collected from both data comparisons and user surveys, and a discussion of findings and future work.
Chapter 2

Related Work

The set of behaviors that college students tend to demonstrate when tackling difficult new material and skill development is a well researched topic. The motivation of this paper centers on encouraging habits of incremental progress that yield success on assignments and fostering growth in areas that bar students from progress. Thus, the outside findings discussed in this chapter will focus on both obstacles and key tactics that researchers have identified. Supplemented with the correlations found among software design students at Virginia Tech, these patterns will serve as a basis for targeting behaviors to promote and weaknesses to improve on using new designs.

2.1 Factors Impacting Programming Performance

The strategies that will later be proposed operate on the premise that students who have success on programming assignments are more likely to be exercising optimal work habits than those who do not, and these habits can be utilized by anyone regardless of talent. Among these practices, some apply to students working on assignments of any type while others are directly related to software design students. Exercises that involve writing code to achieve a desired task are just a small subset of projects that college students encounter over the course of their education, and so it is useful to address the habits that improve academic achievement in all disciplines such as time management and sub-tasking. For software de-
sign, some of the keys to success supported by research include code modularity, self-testing, and documentation.

### 2.1.1 Time Management

As college underclassmen in particular will experience, there is a huge difference in academic responsibility between secondary and higher education [35]. In secondary education, the structure of classes is more rigid and methodical, giving students less flexibility on how to approach their learning. For many students, this makes it harder to fall behind because they don’t have much of a choice when it comes to attending class and following guidelines. By contrast, college is an environment of newfound freedom and independence regarding educational routes to follow and study habits to employ within courses. With this elevated independence comes a critical need for students to focus on time management practices in order to maximize their potential for success.

Time management can be defined broadly as “the art of arranging, organizing, scheduling and budgeting one’s time for the purpose of generating more effectiveness in both work and productivity” [42]. For first-time college students, the importance of this practice is paramount as it can be easily neglected in an atmosphere of heightened control over lifestyle and work patterns. For students who have especially loaded class schedules or are involved in extracurricular activities, a measure of how effective time is managed is often the biggest variable impacting academic performance.

In a study conducted with freshmen and sophomore students at the University of Georgia, subjects completed a comprehensive questionnaire that collected yes/no responses to indicators of short-range planning, time attitudes, and long-range planning [4]. The gathered data
was then compared with two categories: grade point average and SAT score. The results of these analyses reflected that both short-range planning and time attitudes were strongly correlated with grade point average, while none of the time management measures showed a strong relation to SAT score. In summary, regardless of scholastic aptitude the students who reported better planning habits and time-use strategies tended to have higher GPAs, while those who reported poor planning and time-use habits trended towards lower GPAs. The results of this study help to dispel the idea that natural intelligence is the chief cause of success, instead demonstrating that improved time management routines can translate to better performance.

2.1.2 Sub-Tasking

With any large project that requires several days or weeks of work, the overall objective at first glance has the tendency to seem intimidating to many students. Especially for those who wait too long to get started on the project, students may be discouraged when viewing the assignment as a huge block of work that needs to be turned in. An effective strategy for dealing with this type of stress is breaking up the project into smaller increments [1]. A simple sub-task of the assignment is a lot more mentally manageable than the final product, and completing many miniature tasks that combine in the end to form a large project can ease the overwhelming feeling initially perceived.

From a psychological standpoint, setting smaller goals motivates people to accomplish bigger things due to the natural responses of the human brain. When even a small amount of success is experienced, the brain releases a chemical called dopamine which is responsible for pleasure, learning, and motivation [33]. Not only do such short-term advances satisfy an intrinsic desire for progress, but they trigger a subconscious motivation to repeat the same
class of actions that led to success, which neuroscientists refer to as “self-directed learning” [20]. On the flip side, an extended period of time without any measurable form of success can have the effect of draining the brain of dopamine, which explains why students who focus strictly on long-term goals are often prone to a lack of motivation and difficulty moving forward.

In a study by Teresa Amabile and Steven Kamer whose findings were published in the Harvard Business Review in 2007, members of twenty-six project teams across seven companies responded to end-of-day surveys pertaining to their perceived levels of daily achievement [2]. The survey inquired primarily about participants’ motivation and emotional responses to the challenges, tasks, and outcomes they experienced on the given day. An overarching trend that stood out among the nearly 12,000 entries collected was that days of highest motivation and productivity were most commonly linked to the accomplishment of minor milestones. “Big wins” by contrast were extremely rare, and in fact many of the reports that summarized a bad day at the office expressed frustration in failing to chip away at a dauntingly large task. In general, those who viewed their long-term goals as a collection of miniature targets to hit each day were more likely to make frequent progress than those who seemed to fixate on the grand scheme.

The results of this study demonstrate the power of incremental progress, which is setting subsequent short-term goals that are each more focused and attainable that the larger entity they comprise. Not only can incremental progress increase people’s engagement in their work, but it can also have a major impact on people’s enjoyment of their work [2]. By this token, a key to encouraging students to have a better emotional outlook on completing assignments is to foster an approach that checklists small items rather than heaping all the work together.

Each of the behaviors discussed thus far help to identify means of positively impacting
academic performance regardless of the discipline toward which they are applied. To supplement the targeting of these general tendencies with techniques from which computer science learners in particular will benefit, it is necessary to also explore the practices in software design that tend to yield higher programming success.

2.1.3 Code Modularity

For plenty of students who are just starting to gain experience with programming, there is a natural tendency to write large contiguous chunks of code [9]. Prior to knowing anything about programming, students are accustomed to problem solving approaches that facilitate progress linearly from top to bottom. By contrast, the paradigms of object-oriented programming operate on the communication of many distinct modules that do not have any particular “order”. Although difficult to grasp for many first-time learners, it is a crucial habit for the purpose of localizing functionality and potential sources of error.

The degree of modularity a program has can significantly affect a student’s ability to follow the logic of his or her code and diagnose the issues that are likely to surface in an introductory programming project [23]. In Java, a highly modular program translates to one whose functionality is facilitated by the interaction of several classes whose purposes and attributes are distinct, as well as multiple methods within each class to group actions into reusable functions [27]. Following this model makes it much easier to keep track of the activity that goes on within a program and limit any faulty behavior to a small scope [43]. On the contrary, encapsulating many lines of code within just a few classes and methods tends to make it difficult for students to attain a functionally-sound, error-free program and be able to fix bugs quickly. The latter trend is seen commonly among beginners, which leads many into roadblocks that are hard to overcome.
As with any other new skill, grasping the object-oriented model and understanding its importance takes time and continual application. Encouraging students to write code that adheres to such modular qualities is a staple in developing programming ability, but it is certainly a challenge in introductory environments where students are largely on their own in constructing software solutions. Fortunately, the evolving tools that students use to submit their code offer helpful indications about their degree of modularity. With regard to this quality, identifying aspects of programs that are non-ideal as well as reinforcing positive steps forward has the potential to improve students’ performance significantly.

2.1.4 Self-Testing

Another practice that is correlated with higher achievement on programming exercises is test-driven development. Students are likely to have fewer overall bugs in their code and experience a smoother pattern of incremental progress if they understand how to test their own code and use this strategy frequently over a project’s timeline. Under a test-driven model, students are gravitated towards having a running solution at all times, confidence in the portion of the target solution they have completed, a clear indication of their progress, and the ability to measure the impact of any changes or additions they make to their code [13].

A pitfall that students can suffer from under a contrasting model is writing all the solution code first and saving all testing until the very end, which makes it difficult to identify sources of error. On average, students will experience a greater amount of bugs and spend more time trying to figure out how to fix them if they engage in this “Big Bang” development. The primary benefit of the superior test-driven development lies in the fact that functional soundness is maintained on the fly due to continual testing of every new feature to be added
as well as regression testing of all code previously written. If the new code breaks any of the previous code then test-first coding will reveal this information instantly, allowing for early detection of errors and minimal time spent debugging.

To measure the effects of a test-driven approach compared with a non-test-driven approach, Dr. Stephen Edwards of Virginia Tech compared the performance of students on two identical programming assignments, one from 2001 and one from 2003 [14]. The 2001 class of students used an automatic grading system called the Curator which did not enforce any principles pertaining to test-driven development. The 2003 class used Web-CAT which assigns a test validity score and a test completeness score in addition to a code correctness score, thus rewarding students for writing thorough tests of their own code. The average score of the 2003 students who used test-driven development was 94 percent, while the 2001 students scored an average of 76.8 percent. In addition, the solutions of the 2003 students were found to have 45 percent fewer defects per thousand lines of code than the 2001 students. These results were very significant in reflecting the improved ability of students to minimize errors and achieve high scores if they use the test-driven model.

2.1.5 Documentation

In addition to the mechanisms used to encourage students to iteratively self-test their code, Web-CAT is now equipped with features that reward students for properly documenting their code. This includes writing comments associated with functions to establish what the parameters are and what actions and output are expected, comments associated with classes to describe the data members an object has and what purposes the object serves, and comments that give context and explanation about the algorithmic approach of complex code. The reason these characteristics are factored into Web-CAT’s assessment criteria is
that software documentation is a crucial activity for students. It enables them to follow their own code in an organized way, keep track of the role that each section of the code has, and facilitate habits of strategic planning and decision-making [26]. Further, many software errors and inefficiencies can be traced back to poor documentation, and so following an ideal set of documentation standards can go a long way in reducing these issues.

Unfortunately, a majority of students are neither enthusiastic nor motivated in the area of documentation despite its benefits of organization, efficiency, and easier code reuse and integration [26]. Rather, it is a common tendency to focus on writing only the actual code to achieve the desired program behavior as quickly as possible. This trend gives motivation to adapting program assessment tools so that they incentivize students to comment and organize their code based on a clear structure. Doing so will make students more likely to develop key documentation habits among the other best programming practices that have been discussed.

2.2 Trends Visualized by Student Submission Data

The treatment group exposed to the Daily Missions feature in Web-CAT is a class of students in Virginia Tech’s introductory software design class. To get a feel for the prior behavior of students in this class, data was collected from software design students at Virginia Tech in 2015 pertaining to Web-CAT submissions. In this data table, each row represents one submission and the columns represent various attributes. The attributes used to produce the visualizations in this section are defined as follows:

- **Due Date Offset**: The number of minutes remaining until the deadline when the submission was made
• **Code Coverage**: The percentage of solution code that has been self-tested by the student, computed by one of three metrics chosen by the professor—number of statements tested divided by number of statements, number of methods tested divided by number of methods, or number of conditionals tested divided by number of conditionals

• **Correctness**: The percentage of instructor-provided reference tests passed by the submission divided by the code coverage of the submission

Each of the following charts aggregates data for five different programming assignments based on these attributes. Many attributes are expressed as a percentage rather than a sheer number so that they can visualized using a common scale. As a first example, consider Figure 2.1. Each colored line of this graph reflects data for a different program, particularly the percentage of students in the class who have made at least one submission at the corresponding point in time. The x-axis can be viewed as a timeline with the due date sitting at the rightmost end of the axis.
Figure 2.1: Line graph showing a progression over the project timeline of students who have made at least one submission.

From this graph, it can be observed that across all assignments, a majority of students (over 50%) waited until somewhere between 4% and 15% of the timeline remaining to submit their first attempt. With 25% of the project timeline remaining, about 80% of students on average across all five assignments had not made a submission yet.

The next graph, shown in Figure 2.2, visualizes the relationship between time of first submission and performance, which is measured here by correctness percentage. In this scatterplot, each black dot represents a set of submissions for a particular student for a particular
2.2. Trends Visualized by Student Submission Data

program. The x-position of the dot corresponds to the percentage of the project timeline remaining when the student made their first submission, and the y-position corresponds to the final correctness percentage attained by the student for that assignment.

![Scatterplot](image)

**Figure 2.2:** Scatterplot that arranges data points by time of first submission and final correctness percentage

A noteworthy observation from this graph is that a vast majority of final correctness scores below 75% came from students who made their first submissions in the last quarter of the project timeline. Most of the students who made their first submission with earlier than a quarter of the timeline remaining scored in the 85-100% range. The cohort of students waiting until the last 10% of the timeline to make their first submission scored an average of 79% correctness, which was significantly lower than the rest of the class.
The next graph, shown in Figure 2.3, also takes the form of a scatterplot and reflects the average code coverage students had across all their submissions versus their final correctness percentage. As in the previous graph, each black dot represents a set of submissions for a particular student for a particular program. The x-position of the dot corresponds to the final correctness attained by the student for that assignment, and the y-position corresponds to the average code coverage attained by the student across all their submissions for that assignment. The latter value is computed by dividing the sum of all code coverage percentages of each submission in the set by the number of submissions in the set.

![Figure 2.3: Scatterplot that arranges data points by final correctness percentage and average code coverage across all submissions](image)

The overall trend that be seen in this graph is that as final correctness scores approach...
100%, average code coverage generally becomes increasingly clustered toward the top of the y-axis. In particular, the highest density of students who attained a final correctness percentage between 95 and 100 percent were those who also had an average code coverage between 95 and 100 percent. Further, a vast majority of students scoring higher than 80% correctness in the end had an average code coverage higher than 85%.

By observing the trends at work in each of these visualizations, a set of takeaways can be summarized as follows:

1. Most students do not start making submission attempts until well over a majority of the project timeline has passed.

2. Students who make their initial submission earlier receive, on average, a higher final correctness score than those who submit later.

3. Students who more consistently and thoroughly self-test their code across all of their submissions tend to receive higher final correctness scores than those who have lower rates of self-testing.

The evidence provided by these charts coupled with the findings of outside research about the effects of various practices on programming performance offers a reasonable set of conclusions regarding factors that help and hurt students, which can be used to craft a strategy for improving students’ success rate. The next sections will discuss approaches for overcoming the constraints of automatic assessment tools in measuring and recognizing student effort rather than just program correctness. The first of these approaches was formulated by Stephen Edwards, Zhiyi Li, and Mukund Babu Manniam Rajagopal of Virginia Tech who developed methods to give qualitative feedback to students based on a set of progress indicators [12, 40], which are used in the logic of the Daily Missions designs later
proposed. The other approaches from which inspiration is drawn pertain to gamification tactics used in digital learning platforms.

2.3 Goals of Automated Assessment

Automated assessment systems evaluate the quality and correctness of software projects by allowing students to upload their code to a website and view results and feedback instantly. Many institutions have adopted these systems for computing because they serve as a quick and easy mechanism for assigning grades to students’ programs. As assessment tools continue to develop, focus has been increasingly placed on going beyond a simple grading system and creating a platform that guides student learning while offering direction and support to those who struggle [22]. Strategies geared toward this objective include identifying sub-tasks for students to complete, implementing features that offer emotional support to students to raise motivation, and providing incentives for students to engage in practices that optimize programming success. Most assessment tools lack these characteristics and are typically more score-oriented in nature, which is why the chief goal of evolving systems is to foster progress in work habits and programming ability.

One of the biggest challenges in making assessment tools more useful for students is being able to generate feedback that not only measures performance but that also measures effort. Assessment approaches in computing are often characterized by points-driven scoring that is easy to be taken in a negative tone due to its focus on mistakes that were made and criteria that were failed to be met [12]. Automatic grading in particular tends to demonstrate this effect because programatically it is much easier to determine the raw correctness of a submission than it is to recognize the time and hard work put in.
2.4 Progress Indicators

To address the issues highlighted above and to make strides towards a system that better reflects student effort in auto-generated reports, Edwards and Li developed a series of progress indicators to be used in Web-CAT’s feedback algorithms. The goal of these indicators is to recognize many types of advancement in addition to just performance outcomes and offer constructive criticism in areas of weakness.

The utility of progress indicators relies on the ability of students to make many submissions over the project time frame to continually gauge how close they are to a complete solution. In total there are fifteen indicators, and each one compares a particular attribute of the current submission with previous submissions. If there is a measured difference between successive states of students’ code that satisfies the criterion for a particular indicator, then that indicator is considered to be “triggered” or “hit”. Multiple indicators can be hit at once or none can be hit at all depending on the code that was added, removed, or changed in the latest submission compared to those prior. Triggering an indicator is intended to be a reflection that the student improved their solution in some specific way other than just achieving more of the functional behavior required by the project specification. To name a few, these additional actions that are tested for include adding new solution methods, reducing cyclomatic complexity, increasing comments density, increasing solution classes, adding new test methods, and increasing conditional coverage.

In addition to using attribute comparisons between successive submissions to detect if a student is making progress in a particular category, keeping track of indicators that are repeatedly failing to be triggered can reflect if a student needs to improve a particular category. As it stands, Web-CAT has the ability to take a given submission and identify which indicators are hit or missed based on any collection of previous submissions. The
question, then, becomes how to determine what feedback to give and when to give it based on the circumstances at hand. Recognizing every single time a student triggers or misses an indicator would be excessive and unlikely to capture students' attention. Rather, a scheme must be devised for strategically interspersing feedback reports with comments that incur a positive emotional reaction from students and motivate them to keep working.

The mechanics of such a feedback scheme were recently explored by Rajagopal in his thesis project [40]. Rajagopal’s work centered primarily on integrating a virtual teaching assistant into Web-CAT to provide easily accessible and human-like support for students. The functionality of this pedagogical agent, referred to as Maria, is equipped with several capabilities. One of Maria’s roles is to provide help via a live chat box that students can interact with by typing in questions about general programming terms and receive instant responses as well as links to external sources of information. In addition, a select few feedback reports throughout a student’s submission set will contain a speech bubble from Maria whose content is associated with one of the progress indicators discussed above. This is where a scheme must come into play for deciding when to display indicator comments, which of the indicators to address, and whether to target indicators that were triggered or ones that were missed.

With regard to how often to administer feedback pertaining to the indicators, it is important to display comments frequently enough that students will actually notice them but not so often that students begin to disregard them. To achieve this balance, Rajagopal describes the use of a variable interval reinforcement technique [40]. This term refers to a method of scheduling repeated events such that the amount of time (or other unit) between any two pairs of successive events is different than the average time between events across the entire set [21]. For example, if five pop quizzes are to be given in February with an average time separation of one week between each quiz, then a variable interval reinforcement scheme
2.4. Progress Indicators

might administer the quizzes on the 1st, 10th, 14th, 19th, and 28th of the month respectively. The same principle can be applied in displaying indicator feedback to students based on an average number of submissions desired to separate each occurrence. Variable interval reinforcement has been studied to be one of the best scheduling approaches for developing habitual behaviors [10].

The other issue the scheme must address is how to handle which progress indicator to focus on in the event that multiple are triggered at once or that multiple have been missed over a series of consecutive submissions. Regardless of the selection process in these cases, a list of potential comments is required for each scenario. Comments are grouped into two categories, one that reinforces actions that trigger the indicators and one that provides encouragement to hit the indicators that were missed. To supply the verbal content expressed by Maria in each set of cases, a spreadsheet of variably-phrased comments for each indicator was assembled. A portion of this spreadsheet is shown in Figure 2.4. The language used in these comments focuses on steps students can take to improve their code rather than draw attention to the weaknesses that called for action, and when even a small amount of progress is made the comments provide recognition.
<table>
<thead>
<tr>
<th>Indicator</th>
<th>Reinforcing Comments</th>
<th>Encouraging Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reducing Cyclomatic Complexity</td>
<td>Great job on refactoring your solution by reducing the number of if-statements and loops. This is a great way to ensure that your methods work as expected.</td>
<td>Testing becomes easier if you have fewer paths to test. We recommend that you simplify your program to reduce the number of if-statements and loops used.</td>
</tr>
<tr>
<td></td>
<td>Removing excessive if-statements and loops is key in reducing the number of branches your program can take. Good job on simplifying your program.</td>
<td>It appears that the logic of your solution can be implemented using fewer conditional branches and loops. Try to identify and condense these statements to make your code less complex.</td>
</tr>
<tr>
<td></td>
<td>We are pleased to see that your solution is cleaner and less complex than before. Avoiding unnecessary branches and loop iterations is a great way to make your code more understandable.</td>
<td>To minimize any unwanted behavior that your program may have, we recommend reducing the number of conditional branches and loops in your code.</td>
</tr>
<tr>
<td>Reducing Average Method Size</td>
<td>Good job on reducing the average number of statements per method in your solution. This will make it easier to test your code.</td>
<td>You can create a simpler solution than the one you have right now. Try to look at ways to reduce the number of lines per method while achieving the same functionality.</td>
</tr>
<tr>
<td></td>
<td>We see that your have made your code more readable by reducing the average number of statements per method in your solution. Well done!</td>
<td>It appears that some of your methods contain many lines of code. Making these methods smaller will make it easier to identify potential bugs.</td>
</tr>
<tr>
<td></td>
<td>We are pleased to see that your methods generally contain fewer statements than before. This is a good practice to exercise because it prevents any one method from having too much capability.</td>
<td>Some of your methods have a large number of statements. Don’t hesitate to create new methods to disperse the functionality covered by your existing methods.</td>
</tr>
<tr>
<td>Increasing Comments Density</td>
<td>Well done adding explanatory details to your code. This helps with comprehensibility of your program.</td>
<td>We would recommend that you add more comments to your program to make it easier to understand your logic.</td>
</tr>
<tr>
<td></td>
<td>We are glad to see that your submission contains more comments than before. Adding detailed comments helps people to better understand your program and its intended behavior.</td>
<td>There appear to be large blocks of code in your program that are not supplemented by any comments. We recommend adding comments to these sections to make your code easier to follow.</td>
</tr>
<tr>
<td></td>
<td>Great job on supplementing your code with helpful comments. A well-documented solution is key in making your program easy to follow.</td>
<td>To make your program clearer to an outside reader, identify and explain blocks of code throughout your solution that are missing comments.</td>
</tr>
</tbody>
</table>

Figure 2.4: Snippet of a spreadsheet used to aid the phrasing of comments given by Web-CAT’s virtual teaching assistant

The comments compiled in this spreadsheet were used to populate the intermittent speech bubbles appearing on students’ screens when significant indications of progress or lack of progress are present. To give a sense of how these messages are mapped visually to the elements of Web-CAT’s user interface, Rajagopal provides web mockups for each type of feedback. Figure 2.5 gives an example of how a reinforcing comment would be displayed and
2.4. Progress Indicators

Figure 2.6 gives a similar example with an encouraging comment.

![Figure 2.5: Screenshot of a reinforcing comment given by Web-CAT’s virtual teaching assistant](image)

From this examination of the progress-driven feedback techniques that Web-CAT is continuing to integrate, a number of useful principles can be highlighted to motivate the design of new features. The most relevant tactic observed is the crafting of feedback to highlight specific and manageable tasks students can complete to improve their solution, as well as recognizing when a student has made an advancement. The underlying theory is that giving students a sense of what actions they can take to make progress in addition to reflecting the points they’ve already received is more effective than simply identifying which portions of the code work and which do not. In addition, using a system in which feedback is given periodically can raise the likelihood of students making a habit out of making frequent small improvements to their code.
2.5 Gamification Schemes

In parallel with automated feedback, a technique that has been increasingly used in online education tools to strengthen their effectiveness is gamification, which refers to giving game-like attributes to a non-game entity [25]. Gamified systems add a new layer to an existing process that intrigues users beyond the instrumental activity of the task [28]. This practice has gained significant popularity in the last few years, with applications spanning from business and marketing strategies to online textbooks, problem-solving modules, and digital education tools. To visualize this recent growth, a Google Trends chart of “gamification” is shown in Figure 2.7.

![Figure 2.7: Interest over time chart for the word “gamification” in Google Trends](image)

As evidenced by this chart, there has been a steadily high interest in gamification since 2012 indicating its recent widespread exploration as a motivational approach. This trend raises the question: what makes gamification work? One answer centers on the fact that many students are already familiar with gameplay elements from their years of using video games as teenagers. In fact, today the average young person will have spent 10,000 hours playing video games before the age of 21, which is nearly equivalent to the total time they
spend in a classroom during all of middle school and high school [31]. The enjoyment that young people derive from video games makes a strong case for the theory that gamifying educational tools will make learning more fun and engaging. The other answer lies primarily in psychology. User experience designs are increasingly placing emphasis on elements that target human emotions [41], and gamification in particular appeals to the natural human desires of competition, status, and achievement [30]. In the context of tools that facilitate virtual education and assessment, gamifying certain features aims to incentivize progress by stimulating these intrinsic desires.

To measure the effectiveness of gamification techniques on student motivation and engagement in learning, Chanut Poondej and Thanita Lerdpornkulrat of Srinakharinwirot University in Thailand conducted an experiment with undergraduate students in an information literacy skills course [39]. They compared a non-gamified (control) group with a gamified (treatment) group, the latter of which used an online learning management tool called CourseSites that implemented a system of points, levels, achievement badges, and leaderboards. Each CourseSites activity was set up as a mission that students could complete to earn experience points and increase their rank, and along the way students could see the achievement points they’ve earned so far and what is required to receive additional rewards. To estimate the unbiased treatment effect, Poondej and Lerdpornkulrat used a propensity score matching approach, which quantifies comparisons based on controlled background covariates such as motivational goal orientation, computer self-efficacy, and perception of meaningfulness in classroom environments. Results indicated that students in the treatment group had a markedly higher engagement in learning than students in the control group. These findings align with existing literature in suggesting that a gamified learning activity generates higher levels of engagement and motivation.

A similar study was conducted by Erkki Kaila, Mikko-Jussi Laakso, Teemu Rajala and
Einari Kurvinen at the University of Turku where game-like features were implemented in a university-level online programming course [24]. The designs proposed in this study give a sense for how systems used to complete assignments may be gamified, one of which is displayed in Figure 2.8. This snapshot of a scenario students would typically see combines several gamification techniques:

- The progress bar gives the student a sense of how far he or she has advanced through the assignment and how much is left to cover.

- Virtual trophies along the progress bar give the student a sense of rank and communicate a feeling of achievement for the tasks that have been completed.

- The list of items the student is tasked with breaks a large assignment into smaller pieces making it more manageable.

- Check marks and quantitative scores make the student feel accomplished for the requirements that have been satisfied, while the red dot next to the unmet requirement pinpoints the area that needs improvement.

![Gamified display of student progress in an online programming course developed at the University of Turku](image)

Figure 2.8: Gamified display of student progress in an online programming course developed at the University of Turku [24]
The tactics visualized above highlight some of the key attributes that make gamification a successful approach. The core idea is to make users feel like they are on a journey with many steps along the way, and that they are in control of their own destiny [5]. Building on this model, an effective way to encourage steady progress is the concept of milestones, which break up the journey into perceivably more manageable targets. In this structure, the sense of achievement that so strongly motivates people can be acquired in recurring instances rather than just once at the end, resulting in a repeating pattern of positive user responses. Reinforcing favorable behavior has been proven to increase the likelihood that people will perform desired actions repeatedly, and this holds especially true in systems that frequently reward small achievement. This phenomenon is rooted in the theory of operant conditioning developed by B.F. Skinner in the 1940s. In addition to milestoning and operant conditioning, the biggest benefits demonstrated by the gamified visual above include allowing users to monitor their own progress and also challenging users to accomplish new tasks. These principles are important to address in detail as they give particular motivation to the Web-CAT design features later presented.

2.5.1 Self-Monitoring

Gamified educational systems facilitate a valuable learning style that puts perceived control over progress in the user’s hands. Tracking progress is a common utility provided by gamified tools, enabling students to recognize their own growth and build confidence when they see how far they’ve come [17]. This model of promoting learner accountability and establishing clear expectations for work that lies ahead is often referred to as self-regulated learning [18]. This process is characterized by a cycle in which students monitor the effectiveness of their work methods and react to feedback accordingly [44]. From this
pattern of continually gauging achievement and adapting to the demands of new tasks, students attain a sense of self-actualization that motivates their willingness to learn. This theory is emphasized particularly in phenomenology, which points to intrinsic values as the primary drivers of progress rather than external rewards.

2.5.2 Rising to Challenges

Another useful property of games that translates powerfully into gamified systems is their focus on challenging players to achieve goals and outcomes [25]. The main objective of challenges in gamified educational tools is to drive participation among students and stimulate their problem-solving capacities. Challenges are dictated by a set of rules and can have varying difficulty based on the scope of user actions that comply with these rules. There is often a delicate balance to be attained between challenges that are too hard and challenges that are too simple in order to most effectively motivate students to stay engaged in the learning activity. With this ideal level of difficulty, challenges combined with gameplay can encourage students to practice skills more frequently and view obstacles as opportunities to improve rather than deterrents to their self-esteem. As described by teacher Amanda Moore who gamified her classroom with a site called Classcraft, “rather than feeling defeated when failing at a task in our game, my students have returned to the task with renewed determination, rising to the challenge with a positive attitude” [34].

2.5.3 Persistence

A major cognitive benefit of gamification is its ability to forge a sense of perseverance in learners in the face of difficulty. In contrast to traditional learning environments where students can lose motivation easily, gameplay elements reduce the level of discouragement
2.5. Gamification Schemes

a user may feel as a result of failing to complete a task. Instead, they draw attention to the user’s progress toward reaching the next level or mission based on tasks that have been completed [8]. By this nature, mistakes are offloaded to the process of play in a way that “minimizes personal association with failure and encourages students to strive for mastery” [18]. Just as video games make players want to keep trying because they’ve come far enough to believe they are capable, game-like features in learning activities engage students to continually work towards their next unmet challenge. In addition, the frequent feedback provided by gamified systems yields a recurring feeling of accomplishment, and this intrinsic reward motivates users to keep chipping away at more difficult goals.

2.5.4 Operant Conditioning

The underlying tactic that drives most of the motivational benefits of gamification is the concept of reinforcement. Skinner defines reinforcers as responses that increase a behavior’s chance of being repeated, forming the basis for the term operant conditioning which means “changing of behavior by the use of reinforcement which is given after the desired response” [32]. In the context of video games, the desired response is for players to return to the game over and over, making actions that urge them to keep playing a focal point for reinforcement. Gameplay reinforcers often entail some kind of reward mechanism that “provides a sense of fun by fostering an intrinsically rewarding experience” [19]. The word intrinsic is used rather than extrinsic here because for most gamers, rewards in the video game do not translate to any tangible real-life gain but rather a positive emotional response. In particular, feelings of accomplishment, satisfaction, and elevated status are commonly taken advantage of to incentivize play.

The gamification theories detailed in this section help to form the basis of the software
contributions to Web-CAT on which this paper is focused. The feature introduced is a system of “Daily Missions” that students can complete over the course of their progress on any programming assignment. These Daily Missions offer frequent opportunities for incremental achievement and are designed, as their name implies, to encourage small amounts of progress each day.
Chapter 3

Daily Missions Design Goals

The concept of Daily Missions started with the idea of offering successive milestones for students to achieve along their path to completing a program. These milestones were intended to help students target specific tasks to improve their solution and encourage students to work at a gradual pace over a longer time frame. To align more closely with the paradigms of gamification, this idea transformed into a design of more frequent, shorter-term “missions” that students are able to attempt on any given day. At a glance, this system operates by randomly selecting a subset of programming-related missions from a predefined list and offering a virtual reward to students for completing them. The objectives of these Daily Missions are motivated by the literature and research discussed thus far pertaining to student performance factors, progress indicators, and gamification techniques, as well as qualities surfaced by reworked initial approaches. This chapter lays out the proposed requirements of the Daily Missions based on these observed findings.

3.1 Formal Definition

Before delving into the mechanics and characteristics of the Daily Missions, it is first essential to define what a Daily Mission is. Inspired by challenge in video games that have a fixed period of availability, a Daily Mission in Web-CAT is a task offered to students open for 24 hours requiring a distinct code improvement in order to be completed.
3.2 Attributes and Criteria

On the first submission of each day of the project timeline, students are presented with a fresh set of Daily Missions that their current program state has the capability of achieving with future improvements. This section provides a complete list of characteristics these missions aim to have, along with brief explanations detailing why each attribute will have a positive impact on the student’s program.

1. **Missions should target tasks that help students improve the overall quality of their software.**
   
   Gamified challenges are most effective when they induce a intrinsically rewarding feeling upon completion. Students want their software to improve so they can reach a higher score, and so when students recognize the positive impact of completing a mission they will be motivated to attempt more missions.

2. **Missions should map directly to the criteria of the progress indicators.**
   
   In accordance with the objective of identifying tasks that recognize many types of software improvement rather than just correctness, crafting the missions to be based on the progress indicators achieves this goal.

3. **Students should have multiple missions to choose from each day.**
   
   Another objective discussed was to prevent students from falling into “stuck” situations while attempting the missions. By offering several alternatives in cases where the student cannot get past a certain mission, students are more likely to keep working to improve their solution. This property mirrors game design features that allow players to choose from multiple paths in order to reach the next level or reward.

4. **Missions should each focus on a specific software element.**
A key property of the Daily Missions is that they are high in numbers and low in individual size rather than consisting of a few missions with multiple criteria per mission. This is advantageous to the student because it breaks up the large assignment into small manageable tasks that are each reinforced by a rewarding experience. By this method, students are recognized for their efforts more frequently and thus more likely to repeat the behavior, according to the reinforcement theories in gamification.

5. **Missions should be diverse with regard to the software elements they aim to strengthen.**

As explored in Chapter 2, there are many factors that impact student performance on programming assignments and principles students should follow for the purpose of optimizing their coding experience. To draw attention to each of these identified categories, the missions should each pertain to a different attribute and should collectively cover the entire breadth of indicators.

6. **Missions should be descriptive enough to provide a clear understanding of the task but not so wordy that students don’t read them.**

Web-CAT already contains plenty of information that captures users’ attention, and so it is important to phrase the missions in a way that motivates users to read them beyond the other elements on the screen. Missions can be thought of as concise instructions that reveal just enough information for users to gain a clear idea of their objective. Too many words risks an information overload that students are unlikely to read, and missions that appear excessively long might give students the impression of missions being too challenging.

7. **Missions should clearly indicate what the student must achieve and include the benefit of completing the task.**
The sentence structure of the missions has two parts—a description of the action a student must take to complete the mission, and a note in parentheses describing the positive impact of the action. To ensure that everyone interprets the missions in the same way, the action part of the missions indicates specifically the code change that is required, and the impact part provides additional incentive for students to attempt them.

8. **Missions should incentivize students to spread their progress among frequent and spaced out work sessions.**

   One of the trends observed in Chapter 2 was that most students wait until the deadline is near before they start making submissions to Web-CAT. To encourage students to distribute their progress more evenly over the project timeline, a gamified strategy has the potential to be effective. Just as challenges in video games make users want to keep playing until they accomplish the task, the Daily Missions are intended to motivate students to keep adding to their solution and re-submitting until they achieve the missions.

9. **Missions should be associated with a game-like rewards system.**

   In addition to the reinforcement students feel when they complete a mission and recognize its benefit on the score and quality of their solution, students can be incentivized by a virtual reward according to the theories discussed in Chapter 3 pertaining to operant conditioning. This virtual reward is the concept of submission energy, which is described in more detail in the next section.

10. **Rewards should not be extrinsic in nature.**

    This property was determined after examining the viability of other types of rewards that provide external gain such as extra credit. The use of extra credit as the sole
incentive was unfavorable due its potential for ineffectiveness. Many professors have found that extra credit tends to primarily benefit the students who already achieve highly in the class [6] and may provide a misleading basis for judging ability and performance [36]. Further, gameplay rewards usually do not translate to any tangible real-life gain and are effective on the basis of their intrinsic benefits.

11. The display should indicate which missions have been completed and which missions have yet to be achieved.

   Completed missions should stay visible to students in order to recognize their accomplishments made so far, but the display should also differentiate between missions completed and missions not completed so that students do not get confused.

12. The display should recognize when a student completes a new mission in a way that is distinguishable from older missions completed.

   When a new mission is completed, it is important that students be made aware of the measured progress they just made and attain a feeling of accomplishment. Therefore, newly completed missions should be given visual attributes that are different from those completed in previous submissions.

3.3 Rewards

As discussed in the previous chapter, the success of many gamification strategies relies on their use of operant conditioning. The behavior that the Daily Missions aims to induce repeatedly among students is the act of viewing and attempting missions until they are completed, where the incentives for doing so include indications of code improvement and virtual rewards. Although it was established that offering a purely extrinsic reward such
as extra credit would not be the best choice of reinforcement, there are other types of points-independent rewards that can achieve the desired effect. In particular, Edwards has recently experimented with the use of submission energy inspired by commercially successful game design techniques that condition frequent gameplay. [15]. These techniques operate on players having a limited but regenerative resource required for play which they are responsible for managing. Doing so to encourages “small, periodic play sessions spread over a longer period of time” [15]. To achieve the same goal in Web-CAT, each student is given an energy bar that holds a maximum of three units of submission energy. One unit of energy is used each time a student makes a submission, and a student is only able to submit if they have at least one unit of energy. A partially filled energy bar regenerates at the rate of 1 unit per hour until the bar is full. While the effectiveness of this approach is still being evaluated, the intended consequence is a general improvement in time management habits among students.

The Daily Missions feature adds another layer of incentive to this submission energy concept. In addition to the passage of time acting as a mechanism for regeneration, submission energy is awarded to students for achieving a task associated with their current missions. In particular, for each newly completed mission, one unit of submission energy is earned. Further details on how these rewards are visually recognized will follow later in the Daily Missions use case walkthrough.
Chapter 4

Daily Missions Implementation and Display

This chapter describes how the design goals for the Daily Missions translate to their implementation and visual feel. The criteria established for the Daily Missions form the basis of the algorithm used to determine which missions are given to students each day. To give a sense for the various visual states the Daily Missions may have, a summary of the user-interface elements as well as a use case walkthrough is provided.

4.1 Selection Logic

On the first submission of each day, students are presented with five Daily Missions selected from a predefined list of 14 missions. Each of these missions is available for 24 hours, after which point a new set of missions is generated. The algorithm for determining which missions are offered on a given day can be summarized as follows:

1. From the list of all 14 possible missions, a subset of “eligible missions” is extracted. In order for a mission to be eligible, it must be possible for the student to complete based on the student’s latest submission.

2. From the list of eligible missions, all missions that were offered on the previous day
are removed unless doing so makes the size of the list drop below five missions.

3. From the remaining list, four missions are randomly selected plus the correctness mission as long as the student has less than 100% correctness.

The list of all 14 missions from which each day’s missions is selected is given below. Each mission corresponds to a different progress indicator:

1. Add more methods to your solution, to make your methods shorter and easier to understand (to increase readability and improve method design)

2. Fix coding style and documentation errors (to increase your score)

3. Simplify the logic in your solution by removing or consolidating if statements or loops (to increase readability and testability, and reduce potential for bugs)

4. Reduce the length of your methods by breaking them into more manageable pieces (to increase readability and improve method design)

5. Add comments to your solution (to increase readability)

6. Add another class to your solution (for better self-checking)

7. Pass one or more additional reference tests by improving your program’s behavioral correctness (to improve your score)

8. Add new test methods to increase your self-checking (for better self-checking)

9. Add to your software tests to increase your self-checking (for better self-checking)

10. Add to your software tests to exercise more of the statements in your solution (for better self-checking and to improve your score)
11. Add to your software tests to exercise more of the methods in your solution (for better self-checking and to improve your score)

12. Add to your software tests to exercise more of the logic conditions in your solution (for better self-checking and to improve your score)

13. Add more assertions to your software tests so they check expected behaviors more comprehensively (for better self-checking)

14. Add another test class (for better self-checking)

4.2 Visual Design

The section details the visual elements used to represent the Daily Missions and why these choices were made over others. Since the Web-CAT feedback page already contains plenty of static information on the screen, a desired property of the Daily Missions display was for the missions to generally only be viewable in response to the user clicking a button. A view of this button is shown in Figure 4.1. An important detail to note in this visual is that the button is located directly underneath the submission energy bar along with a “Need more submission energy?” message to make it clear that completing the missions will cause the student’s submission energy to increase.

![Figure 4.1: View of the submission energy bar and Daily Missions button](image-url)
In contrast to initial approaches that proposed a slide-out panel to present the content of the Daily Missions, the interface that was ultimately decided on was a modal window that overlays the screen when the button is clicked. The reason for this decision was that the amount of information displayed is not as high as the level suitable for slide-out panels, so instead a widget was used that better focuses attention on precise content [16]. A sample view of the Daily Missions modal window is shown in Figure 4.2.

![Figure 4.2: View of the modal window appearing when students click the Daily Missions button](image)

With regard to the visual process for notifying students of their progress towards completing missions, the goal is to display information noticeably enough to draw attention to recent accomplishments while avoiding recurring content that acts as annoyance or visual noise. In accordance with this property, when interacting with the Web-CAT submission page students generally only see the popup Daily Mission modal in response to clicking the Daily Missions button. However, there are a few cases where this modal window will pop up automatically. One such case is if the student is viewing the website as the current missions expire (triggering a new set of missions to generate), or if the student is viewing the site for the first time of the day. The reason for this logic is to notify the student that the current Daily Missions contains a new set of tasks than the last set of missions attempted. Another case in which the Daily Missions modal window will pop up automatically is if the student
makes a submission that satisfies the criteria of a new mission that has not been previously completed that day. The intent of this event-driven behavior is to make students clearly aware of the progress they’ve just made and communicate a sense of accomplishment that will motivate them to keep moving forward.

4.3 Use Case Walkthrough

With several distinct items students are tasked with by the missions and the differences in time at which they may be completed, the content of the Daily Missions modal window can have various potential states. To demonstrate actions that would trigger these different states, a series of snapshots is provided that indicate the visual changes a student might see over the course of completing a particular set of missions.

4.3.1 Missions Generated on First Submission

To begin the scenario, suppose that for a particular assignment a student has just submitted to Web-CAT for the first time. When the feedback page loads, the Daily Missions modal window would pop up automatically and display a list of missions in a checkbox format, as shown in Figure 4.3. Since each of these missions were freshly generated, none of them have been satisfied and thus the checkboxes are all initially empty.
4.3.2 Completing a New Mission

Now, suppose the student looks at these missions and decides they want to fix their coding style and documentation errors in order to make progress. After the student makes these changes and submits to Web-CAT a second time, the content of the Daily Missions modal window would appear as shown in Figure 4.4 if the student was successful. From this screen capture, it can be observed that the box next to the “Fix coding style and documentation errors” mission has been filled with a green check mark and the text has been highlighted in green. Also, a green box has appeared near the bottom of the window containing a comment that notifies the student that they just completed one new mission and earned a full recharge of submission energy.
4.3.3 Completing a Mission With a Score Impact

At this stage of the scenario, the student has completed one Daily Mission but they have yet not passed any reference tests meaning their score is still zero. In order for fixing coding style and documentation errors to have an impact on the student’s score, their correctness percentage must be greater than zero since the score is determined by multiplying several metrics together including correctness. Recognizing this fact, suppose the student next makes changes to their code in an effort to pass one or more additional reference tests, which is another one of the current missions of the day. After making these changes and submitting to Web-CAT again, the student would be shown the modal window displayed in Figure 4.5 if successful. Visual details to note in this screen capture include that the older mission completed is still marked with a green check mark but the text is no longer highlighted in green, while the new mission that was just completed is highlighted in green to make the distinction between old and new. In addition to the green box containing a comment about completing one new mission, another green message box has appeared that reflects the number of project points the students has gained today from completing the missions.
4.3.4 Completing Multiple Missions At Once

To wrap up the use case walkthrough, suppose next that the student attempts to complete the remaining missions, which all happen to relate to self-testing. If the student makes changes to their code that exercise both more of the statements in their solution and more of the methods in their solution and then submits to Web-CAT again, the resulting Daily Missions modal window the student would see is shown in Figure 4.6. As indicated by the visual, two new missions have been completed and thus two missions are highlighted in green with check marks next to them. In addition, the value by which the missions have increased the student’s score has been updated to reflect the missions that were just completed.
4.3. Use Case Walkthrough

Figure 4.6: Modal window that reflects the completion of two new missions and a further increase in score
Chapter 5

Evaluation

This section discusses the mechanisms used to measure the appearance rate and completion rate of the Daily Missions as well as the impact of the Daily Missions on student progress and perceptions. Quantitative data from submission sets were compared between two classes of software design students at Virginia Tech who completed the exact same assignment, one from the Fall 2017 semester which was not exposed to Daily Missions and one from the Spring 2019 semester which was offered Daily Missions. In addition, feedback pertaining to user experiences with the Daily Missions was collected from student surveys given at the end of the Spring 2019 semester.

5.1 Evaluation Criteria

With regard to the data gathered on the Daily Missions, the metrics that were used for evaluation include number of students who saw each mission, percentage of students who completed each mission, and distribution of students who saw only one set of missions versus those who saw multiple. Attributes compared between the 2017 dataset and the 2019 dataset include average number of hours remaining at the time of first submission and average number of progress indicators triggered per submission. In addition to these quantitative data, a survey was given to students asking them to rate their level of agreement with statements pertaining to the Daily Missions’ usefulness, difficulty, motivational impact,
and emotional impact.

5.2 Quantitative Results

Based on the data relevant to the Daily Missions themselves, several quantitative attributes were analyzed. Figure 5.1 visualizes the number of students who saw each mission at least once and the percentage of students who completed each mission among the pool who saw them. On the y-axis of the chart, the names of the progress indicators targeted by each mission are displayed for missions on which data was able to be collected. On the x-axis, the numerical value colored in blue represents the number of students who saw the corresponding mission, and the length of the green bar maps to the percentage of those students who completed the mission.
Another useful subset of the quantitative data collected is the number of distinct sets of missions students saw during their experience with Web-CAT. Since one new set of missions is offered each day, a measure of how many mission sets a student saw is also a measure of how many days early a student made their first submission. Among all students who submitted the assignment, 67.4% only saw one set of missions, 23.3% saw two sets of missions, 5.1% saw three sets of missions, and 4% saw four sets of missions. The average number of mission
sets a student saw was 1.46.

In addition to data comparing distinct Daily Missions to each other, a comparison of submission attributes was conducted between the 2017 class of students and the 2019 class of students to determine the potential impact of having Daily Missions offered versus not using them. A particular metric of interest was how early on students made their first submission, which is visualized in Figure 5.2. In this chart, the x-axis represents the number of hours remaining until the deadline and the y-axis represents the percentage of students who have made a first submission. Data for the 2017 students is denoted by the blue line, while data for the 2019 students is denoted by the green line. On average, students in the 2019 class made their first submission 5.87 hours earlier than the 2017 class. However, based on an analysis of variance (ANOVA) between the two datasets, the P-value is 0.147 indicating that the difference in mean value is not statistically significant.
Figure 5.2: Line graph showing the progression of students who have made at least one submission from both the Fall 2017 and Spring 2019 semesters.

In addition to a comparison of average first submission times, the average percentage of progress indicators triggered by submissions in 2017 was compared to the average percentage of progress indicators triggered by submissions in 2019. In order to more accurately measure the potential impact of Daily Missions on rate of indicators triggered, the 2019 data was filtered by students who saw at least 2 sets of daily missions and the 2017 data was filtered by students who made their first submission prior to 24 hours remaining in the project timeline and last submission after 24 hours remaining (thus indicating that they would have seen multiple mission sets if Daily Missions were offered). The result of this filtered data comparison was that 2017 students meeting this criterion triggered a mean of 3.74 indicators per submission while the 2019 students meeting this criterion triggered a mean of 3.51 indicators per submission. An ANOVA between these two datasets reveals a P-value of
0.113, indicating that the difference is not statistically significant.

5.3 User Survey Results

This section provides the contents of the survey given to the 2019 students at the end of the semester as well as a visualization of the results. For each of the statements listed on the survey, students are able to respond with one of seven choices: strongly agree, agree, somewhat agree, neither agree nor disagree, somewhat disagree, disagree, and strongly disagree. To avoid as much as possible any built-in bias incurred by the phrasing of the statements, some statements were centered on positive outcomes pertaining to the Daily Missions and others were centered on negative outcomes. A full view of the survey that was given to students is shown in Figure 5.3.
Survey on Web-CAT Feedback

This voluntary survey includes questions regarding your opinions on the automated feedback you received on your programming assignments. We will use this information to understand better how you view that feedback, and whether it is helpful to you.

The results from this survey will be used for research purposes and for improving the feedback you receive as you work on assignments. Please complete all items, even if you feel that some are redundant. This may require 5 minutes of your time. Usually it is best to respond with your first impression, without giving much thought. Your answers will remain confidential, and will not affect your grade in any way.

Your participation is voluntary. If you do not wish to participate, simply do not fill out the survey. You must be 18 or older to take part in this research.

Thank you for your participation.

Complete the following items. Select the option that best describes how strongly you agree or disagree.
5.3. User Survey Results

The next two charts summarize the aggregation of responses given by students who participated in the survey. For each chart, the survey statements are displayed on the lefthand side and a number line from -3 to 3 is displayed on the righthand side. Each integer on the number line corresponds to a different answer choice on the survey, with -3 representing “strongly disagree” and 3 representing “strongly agree”. After each response was mapped to an integer in this fashion, the average of all responses was computed and mapped on the number line. This mean value is denoted by the x-position of a circle, whose color is either green if it falls on the positive end of the number line and red if it falls on the negative end. Additionally, the number of students who responded with each level of agreement is displayed in blue above the tick marks. Figure 5.4 visualizes the responses to statements centered on positive outcomes and Figure 5.5 visualizes the statements centered on negative outcomes.

Figure 5.4: Visualization of survey responses to statements centered on positive outcomes
5.4 Answers to Research Questions

The data comparisons observed in the previous section provides the information necessary to answer the research questions declared in Chapter 2. Each of these questions along with the findings and analysis associated with their objectives are detailed below.

1. Can a system of gamified Daily Missions cause students to make their initial program submissions earlier?

   On the basis of pure attribute comparison, the 2019 class of students made their first submissions on average 5.87 hours earlier than the 2017 students. However, as mentioned, the P-value resulting from the ANOVA performed on the two datasets is not low enough to suggest that the presence of Daily Missions had a statistically significant impact on the mean value. By this measure, it is inconclusive whether Daily
Missions cause students to initially submit to Web-CAT earlier.

2. **Can Daily Missions condition students to make incremental progress on programming assignments?**

   As Chapter 2 and 3 discuss, incremental progress on programming assignments is measured in this project by the rate of progress indicators triggered by submissions on average. Based on this metric, the 2019 class of students (filtered by those who saw at least two sets of Daily Missions) triggered an average of 3.51 indicators per submission, while the 2017 class of students (filtered by those who would have seen at least two sets of Daily Missions if they were offered) triggered an average of 3.74 indicators per submission. Since the P-value resulting from the ANOVA performed on the two datasets is also not low enough to suggest that the presence of Daily Missions had a statistically significant impact on the mean value, it is inconclusive whether Daily Missions condition students to make incremental progress.

3. **Can Daily Missions raise feelings of motivation to work on assignments more frequently and help students identify ways of improving their code?**

   This research question can be answered by the results of the user survey. In general, responses indicate that students had an overall positive experience with the Daily Missions – for the statements centered on positive outcomes, all sets of responses were calculated to have an average value closer to the “Strongly agree” end of the number line than the “Strongly disagree” end. In particular, the average responses falling closest to the “Strongly agree” end of the spectrum were to statements pertaining to motivation to working on the assignment more frequently and giving a better idea of ways to improve code. By this measure, although not to the desired degree, the Daily Missions did indeed raise feelings of motivation to work more frequently and identify methods of improvement.
Chapter 6

Conclusion

The effectiveness of the Daily Missions strategy described in this paper was difficult to evaluate due to a variety of factors. Contrary to the ideal experiment that was originally intended, the Daily Missions feature was not able to be deployed to Web-CAT until students had only one assignment remaining in the Spring 2019 semester. Because of this, the only form of conditioning to the Daily Missions students had was an advertisement of their features and benefits at the beginning of the final project. If the Daily Missions were deployed to Web-CAT early enough so that students could experience them in multiple assignments rather than just one, it is plausible that students’ repeated exposure to the Daily Missions would have driven them more to the practices outlined in the objectives and research questions. However, the reality was that nearly 68 percent of students in the treatment group only saw one set of Daily Missions, which significantly limited their potential impact. Additionally, for the assignment to which the Daily Missions were offered, the majority of students made their first submission between one and two days before the assignment was due, meaning most students saw only one or two sets of Daily Missions. If the missions were offered in an assignment that spanned a longer interval of time in which students generally made their first submissions several days in advance of the deadline, exposure to the missions would be significantly higher and more likely to increase their impact. Unfortunately, the circumstances at hand were limited in this sense and so evaluations were based on data that were not reliably driven by the Daily Missions. That being said, the responses gathered from
the user survey suggest a net-positive effect of the Daily Missions on student experiences with Web-CAT.

### 6.1 Contributions

In terms of the measured results that were produced from the evaluation mechanisms of this project, the strongest findings from the Daily Missions were that the missions generally caused students to have a better experience with Web-CAT, albeit only slightly. This claim is based on the positive value of the survey response aggregates pertaining to statements about Daily Missions’ motivational impact and identification of ways for students to improve their code. Responses also suggest on average that 24 hours was an appropriate amount of time to complete the missions and that the missions were not overly difficult or overly easy. As with any new feature deployed to a system in its first iteration, the Daily Missions strategy in Web-CAT was based little on prior data and more so on observed literature making it difficult to form reliable predictions about the effect various attributes would have. However, from this experiment, a number of future improvements were highlighted to fully uncover the potential of Daily Missions in Web-CAT.

### 6.2 Future Work

Based on the theories and designs explored in this project, a smaller and more focused paper will be submitted the 2019 HICCS Conference (Hawaii International Conference on System Sciences) within the gamification in education track. With regard to the future direction of the project, one of the biggest problems illuminated was the environment to which the Daily Missions were deployed—students were not exposed to the missions frequently
enough to be significantly impacted by them. A future study this project would benefit from would be observing the effects of the Daily Missions on an assignment spanning a longer-term time frame such as one month, where students would typically make their first submission weeks in advance of the deadline instead of one day. Daily Mission sets refresh every 24 hours, and so theoretically the more days a student sees a new set of missions the more likely it is that students will adapt their habits to fit the missions’ demands. Another approach to this problem would be to shorten the time period that missions are available for, thus making them no longer “Daily”.

In addition to studying the effects of the Daily Missions’ availability period and exposure, other interesting variables for experimentation include number of missions offered to students per day, difficulty of the missions, and rewards for completing the missions. This project offered five Daily Missions to students each day, but it would be useful to also measure the impact of having fewer, more challenging missions versus many easier missions. With regard to rewards, one of the issues brought to attention was that the reward for completing one mission was the same as the reward for completing multiple at once. To more appropriately reflect the level of accomplishment incurred by a submission, a future improvement would be to update the reward mechanism so that completing two missions earns twice the amount of submission energy as completing one mission. This may entail allowing the energy bar to be “overcharged”, giving the student more energy units than three in situations that apply.

An idea that was discussed in the design phase of the project but not implemented was the concept of “Starter Missions” consisting of basic tasks providing direction on how to initialize the program. These missions upon completion would form the skeleton and structure of the program to be later filled in with functionality. These missions would ideally give students a greater sense of early accomplishment and better guide students through the beginning phases of programming assignments. A future version of Daily Missions in which this type
of initialization missions are offered would be helpful to explore.

The final focal point of future work to be discussed pertains to an important problem with the current Daily Missions, which is the ability for students to “game” the system to easily earn the desired reward whenever they need it. For example, if a student figures out that the “Adding New Solution Methods” mission can be completed simply by adding one empty method that has no relevance to the rest of the code, the student can intentionally keep this mission incomplete until they need a recharge of submission energy. The current set of submission attributes collected for data analysis does not include any metrics about this potential gaming, and so a future task would be to figure out ways of measuring how often students engage in such activity. Developing these tools would help Web-CAT to combat system gaming and tailor designs to prevent students from exploiting these loopholes.
Bibliography


Appendices
Appendix A

Text of Survey
Survey on Web-CAT Feedback

This voluntary survey includes questions regarding your opinions on the automated feedback you received on your programming assignments. We will use this information to understand better how you view that feedback, and whether it is helpful to you.

The results from this survey will be used for research purposes and for improving the feedback you receive as you work on assignments. Please complete all items, even if you feel that some are redundant. This may require 5 minutes of your time. Usually it is best to respond with your first impression, without giving a question much thought. Your answers will remain confidential, and will not affect your grade in any way.

Your participation is voluntary. If you do not wish to participate, simply do not fill out the survey. You must be 18 or older to take part in this research.

Thank you for your participation.

Complete the following items. Select the option that best describes how strongly you agree or disagree.
In some of your programming assignments, Web-CAT used a **new feedback presentation** to present the issues in your code in a different way:

Please indicate your agreement/disagreement with the following statements related perceived **ease-of-use** or **usefulness** of this feedback.

<table>
<thead>
<tr>
<th>This new style of feedback was easier to understand than before</th>
<th>Strongly agree</th>
<th>Agree</th>
<th>Somewhat agree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat disagree</th>
<th>Disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>It helped me decide on my goal for what to work on next</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The code snippets in the feedback helped me find problems more easily</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I was able to resolve issues with my assignment faster with this form of feedback</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the old style of feedback easier to understand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The new style of feedback made me more discouraged with my work</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The new feedback helped me improve my score faster</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The new feedback is too long and complicated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It was easier to understand what I needed to work on with the new feedback</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I prefer the old style of feedback</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How can we further improve the feedback generated by Web-CAT?
In some of your programming assignments, **Maria the Virtual TA** was available on your feedback page. Maria may have provided feedback about your work. Please indicate your agreement/disagreement with the following statements related to **Maria the Virtual TA**.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly agree</th>
<th>Agree</th>
<th>Somewhat agree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat disagree</th>
<th>Disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maria’s presence in my Web-CAT feedback is distracting</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Maria comments made it feel like I was making progress</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Whenever a comment from Maria appeared on the submission report, I noticed it right away</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>I took the time to read the comment from Maria whenever one appeared</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The comments made me feel more optimistic about my ability to complete the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Maria was helpful in making progress on the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>I found Maria’s comments to be encouraging</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Maria’s comments were relevant to my situation and the current state of my work on the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>I want a virtual TA who is more capable than Maria</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>I did not see Maria</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>I ignored most of the comments from Maria</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Maria’s comments reflected my hard work on the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

**How can the virtual TA be improved to help you better?**


In some of your programming assignments, you may have seen **Daily Missions** that provide small challenges that correspond to tasks you normally work on as you develop your solution. Please indicate your agreement/disagreement with the following statements about **Daily Missions**.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly agree</th>
<th>Agree</th>
<th>Somewhat agree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat disagree</th>
<th>Disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Daily Missions motivated me to start on the assignment early</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions motivated me to work on the assignment more frequently</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions conditioned me to be a better programmer</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions made it seem like I had extra work to do for the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions reinforced the things I did well on the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions gave me a better idea of ways to improve my code</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions were difficult to understand</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions made me more confident in my ability to complete the assignment</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions did not help improve my assignment score</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions made the assignment seem more manageable</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions were too difficult or frustrating to achieve</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>The Daily Missions were too easy to achieve</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Accomplishing the Daily Missions was a satisfying feeling</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>One day was not an adequate amount of time to complete a Daily Mission</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

**How can the Daily Missions be improved to help you better?**
Appendix B

IRB Approval
MEMORANDUM

DATE: May 1, 2019

TO: Stephen H Edwards, Catherine Amelink, Bob Edmison, Michael Scott Irwin, Zhiyi Li, Andrew Goldman, Dylan Finch Finch

FROM: Virginia Tech Institutional Review Board (FWA00000572, expires January 29, 2021)

PROTOCOL TITLE: Collaborative Research: Using Automated Feedback to Promote a Growth Mindset in Programming Assignments (NSF 1625425)

IRB NUMBER: 16-014

Effective May 1, 2019, the Virginia Tech Institution Review Board (IRB) approved the Continuing Review request for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report within 5 business days to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at: https://secure.research.vt.edu/external/irb/responsibilities.htm

(Please review responsibilities before beginning your research.)

PROTOCOL INFORMATION:

Approved As: Expedited, under 45 CFR 46.110 category(ies) 5,7
Protocol Approval Date: May 13, 2019
Protocol Expiration Date: May 12, 2020
Continuing Review Due Date*: April 28, 2020

*Date a Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

ASSOCIATED FUNDING:

The table on the following page indicates whether grant proposals are related to this protocol, and which of the listed proposals, if any, have been compared to this protocol, if required.
<table>
<thead>
<tr>
<th>Date*</th>
<th>OSP Number</th>
<th>Sponsor</th>
<th>Grant Comparison Conducted?</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/01/2016</td>
<td>P534CP3I</td>
<td>National Science Foundation (Title: Collaborative Research: Using Automated Feedback to Promote a Growth Mindset in Programming Assignments)</td>
<td>Compared on 05/04/2016</td>
</tr>
</tbody>
</table>

* Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this protocol is to cover any other grant proposals, please contact the HRPP office (irb@vt.edu) immediately.