

Iterative Improvement of Practice Exercises By Students and Staff

by

Jenna Himawan

S.B. Computer Science and Engineering
Massachusetts Institute of Technology, 2021

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

© Massachusetts Institute of Technology 2020. All rights reserved.

Signature of Author:.....

Department of Electrical Engineering and Computer Science
May 20, 2021

Certified By:.....

Rob Miller
Distinguished Professor of Computer Science
Thesis Supervisor

Accepted By:.....

Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Iterative Improvement of Practice Exercises By Students and Staff

by

Jenna Himawan

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Practice is an important part of mastering any discipline. As a result, many courses provide students with example problems. However, the processes by which these exercises are created and presented to students can make it difficult to create, review, and edit them. This thesis describes an exercise bank framework that facilitates the authorship of new practice questions and the iterative development of existing ones. The exercise bank uses conceptual models that treat exercises not as seldom-changing pieces of course material but as collections of data that are continually in need of review and revision. Operations on these exercises cause them to transition between different states over the course of their development. This thesis discusses the design and implementation of two exercise bank systems for the course 6.031: Elements of Software Construction. These exercise banks proved useful to both staff and students in generating and understanding course material.

Thesis Supervisor: Rob Miller

Distinguished Professor of Computer Science

Acknowledgements

I owe a debt of gratitude to everyone who, knowingly or otherwise, was crucial to the work contained in this thesis: namely, the present and past instructors, TAs, LAs, and students of 6.031. It is due to the time and effort that they put into the class that I have any basis for my work. I would like to especially thank the instructors, Rob Miller (also my thesis advisor) and Max Goldman, for their support, design advice, technical help, and immense patience. Their guidance made my MEng possible, and I'm so grateful to them for giving me the opportunity to be involved with 6.031, as an LA, TA, and research assistant.

I would also like to thank the entire MIT community, especially the students and staff of Simmons Hall. I never got to properly say goodbye to The Sponge, but I will always cherish my memories of Mystery Hunt in Simmons Dining, board games in 7A lounge, and early mornings and late nights in the Country Kitchen during REX and CPW. I might even miss the fire alarms.

My boyfriend, Michael Tang, has been a wonderful safekeeper of my mental well-being. His dorky sense of humor and passion for puzzles reliably put a smile on my face, but it is his truly kind, considerate nature, and the support he has provided in tough moments, that have allowed me to persevere through some very difficult and stressful times.

I cannot possibly express how much I owe my parents, Jeff and Lisa Himawan. Without the values they taught me, or the time and money they invested in my upbringing and education, I would have had no chance of getting into MIT. I will never be able to pay them back for their exceptional parenting, advice, or emotional support, but I will try in the only ways that I know how: by returning the love that they have showered upon me throughout my entire life, and by writing the thesis that follows, in the hope that I might make them proud.

Contents

1	Introduction	11
2	Related Work	15
2.1	Applications of Learnersourcing	15
2.2	Evaluation of Learnersourcing	17
2.3	The Importance of Moderation	18
2.4	Existing 6.031 Tools	19
3	Design	21
3.1	Goals	21
3.2	Operations	22
3.3	Git as a Version Storage Mechanism	24
3.4	Use of Git in 6.031	25
3.5	User Interface	26
3.5.1	Fill-in-the-Blank Exercises	26
3.5.2	Exercise Bank for Student-Authored MCQs	27
3.5.2.1	Students	27
3.5.2.2	TAs and Instructors	30
4	Implementation	43

4.1	Data Storage	43
4.1.1	Fill-In-The-Blank Exercises	44
4.1.2	Exercise Bank for Student-Authored MCQs	45
4.2	User Interface Implementation	48
4.2.1	Fill-In-The-Blank Exercises	48
4.2.2	Exercise Bank for Student-Authored MCQs	49
5	Evaluation	56
5.1	Fill-in-the-Blank Exercises	56
5.2	Student Practice with Student-Authored MCQs	57
5.3	Facilitation of Exercise Development	58
5.3.1	Number of Contributors	59
5.3.2	Amount of Change per Exercise	60
6	Conclusion	64
6.1	Further Work	65

List of Figures

1.1	Current states, data flows, and operations for fill-in-the-blank coding exercise questions	13
1.2	Current states, data flows, and operations for Questionable Makeup submissions	13
3.1	Exercise state transitions	23
3.2	States, data flows, and operations for fill-in-the-blank exercise questions with the exercise bank system	33
3.3	States, data flows, and operations for Questionable Makeup submissions with the exercise bank system	33
3.4	The VSCode-based interface for playtesting a fill-in-the-blank exercise .	34
3.5	An example email about an error in fill-in-the-blank exercises	35
3.6	The home page of the exercise bank	36
3.7	The practice page of the exercise bank	37
3.8	The box for student feedback	38
3.9	After the student leaves feedback	38
3.10	The modal box for suggesting edits	39
3.11	Error message seen if student does not attempt a question	40
3.12	Error message seen if student does not complete the formatting checklist	40
3.13	The add exercises page	40

3.14	A page for an individual exercise which is being considered for addition to the exercise bank	41
3.15	The review exercises page	41
3.16	The page for reviewing a single exercise	42
4.1	if::else-if concept	51
4.2	Basic TypeScript concept group	51
4.3	if, else-if, and else tutorial in TypeScript	51
4.4	An example of metadata at the top of a fill-in-the-blank exercise file . .	52
4.5	The GitHub web UI for editing and committing a file	53
4.6	Names of GitHub directories in the exercise bank repository	54
4.7	Filenames of exercises about the topic of ADTs	54
4.8	The webhook information page on GitHub	55
5.1	Number of attempts per day	58
5.2	Number of users per day	59
5.3	An example of formatting, wording, and conceptual changes	61
5.4	Changes made by TAs	62
5.5	Changes made by students	63

Chapter 1

Introduction

Providing students with opportunities to practice the concepts that they have seen in class is vital for their learning. Collections of questions and exercises explicitly designated for practice are especially helpful. However, creating and maintaining these collections is quite difficult. Generating and proofreading new questions to use is time-consuming, as is editing existing problems. Encouraging the creation of and iteration on course exercises can be done by increasing the number of people involved in these processes and decreasing the amount of time and effort necessary to participate.

Coming up with an exercise takes time. One way to ensure that a greater amount of time is devoted to exercise generation is for existing authors to put in more effort. However, this is not easily scalable, and these people often have other, higher-priority tasks, especially lecturers and section instructors. A natural alternative is to increase the number of contributors. This has the added benefit of bringing new perspectives to the task, which is especially helpful for creative undertakings like exercise creation.

Course infrastructure occasionally imposes an additional barrier to adding and modifying exercises. For instance, someone may need to learn where and how data is stored and edited before even starting on the process of exercise revision. This can be unneces-

sarily troublesome, especially if data is in multiple places or uses unfamiliar frameworks or file formats. Furthermore, reviewing interactive exercises for correctness may require running a local development server. Decreasing the number of things that a potential exercise author has to learn and the number of steps required to bring an exercise from its concept stage to its review stage would make the exercise-writing process more user-friendly and less laborious.

To achieve these goals, this thesis proposes an exercise bank system that allows more users to contribute to practice problems and makes it easier for any individual user to do so. The exercise bank is designed and tested for MIT’s 6.031: Software Construction class, consisting of approximately 250 students per semester. This thesis describes two exercise bank implementations for two groups of exercises in this class: *fill-in-the-blank exercises* and *student-authored multiple-choice questions*.

In order to ensure that students practice the syntax and features of a possibly unfamiliar language, 6.031 requires that they complete a series of fill-in-the-blank coding exercises, using a system based on Java Tutor [14]. Adding one of these exercises requires uploading a source file and specifying metadata. This metadata includes hints, links to tutorials, names of concepts that are being utilized, and regular expressions that match possible solutions. Prior to the implementation of the exercise bank framework, testing a new exercise in order to verify that it appears correctly to students required that the tester had a clone of the Git repository of exercises as well as a local development server. This workflow is shown in Figure 1.1. The exercise bank streamlines the process of adding new exercises, testing them, and modifying them.

In 6.031, students are able to do a bit of additional work in order to recover some lost points from in-class exercises. Students use the Questionable Makeup System [11] to answer and provide feedback on three multiple-choice questions (MCQs) similar to ones that might appear on exams, and then finally write one multiple-choice question

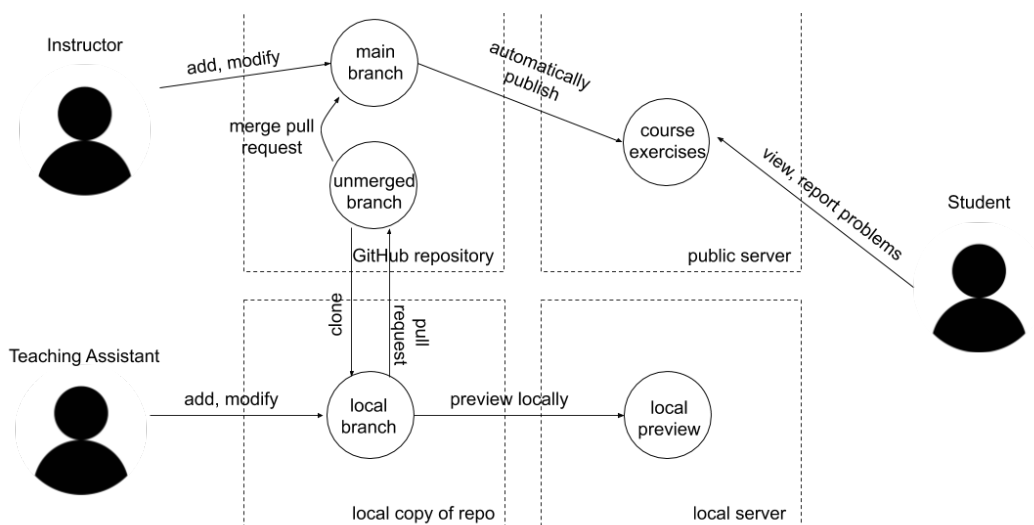


Figure 1.1: The current states, data flows, and operations for fill-in-the-blank coding exercise questions.

themselves. Prior to the creation of the exercise bank, these student-authored MCQs were graded by staff and stored in a database, but not subsequently used. This workflow is shown in Figure 1.2. The introduction of our framework allows the student-authored MCQs to be reviewed, revised, and made public for other students to practice.

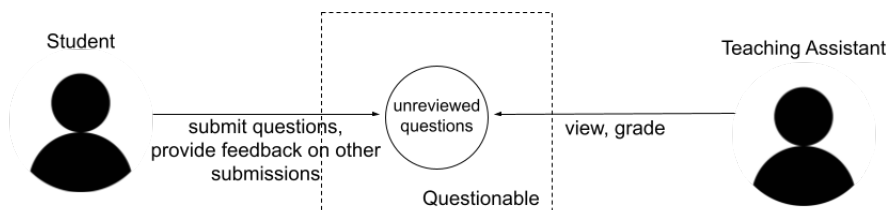


Figure 1.2: The current states, data flows, and operations for submissions to the Questionable Makeup system.

The two exercise bank implementations model exercises as objects whose associated operations cause them to transition between different stages of revision. The current contents and history for each exercise are stored in a Git repository, which supports version histories very naturally in addition to facilitating collaboration. This Git repository and MongoDB together store the exercises and metadata the system

needs. Staff members add and edit fill-in-the-blank exercises through the Git repository, which they can then preview using an IDE extension. The exercise bank of student-authored MCQs is managed through a web app. Staff members review, revise, and publish student-authored MCQs, and students answer the published ones.

The exercise bank for fill-in-the-blank exercises allowed collaboration from many members of the 6.031 staff. It successfully reduced the workload on the instructors, and was intuitive enough that the non-instructors who contributed were not overly burdened by a learning curve. The improvements this thesis contributed to the user interface were also well-received by the staff.

Students demonstrated significant interest in practicing with the exercise bank of student-authored MCQs. Even though interacting with this exercise bank was optional, almost half of the class attempted at least one exercise. Usage was relatively steady throughout the semester, with a very noticeable peak right before the quiz held in the middle of the term. In addition to being useful for students, the exercise bank successfully allowed both students and staff to iterate on the student-authored MCQs. The majority of questions were edited in some way from the original, and approximately 37% of these revisions were more significant than small edits to formatting and wording.

Chapter 2

Related Work

The exercise bank is designed to provide students with high-quality practice questions. However, students do not have to simply be viewed as a passive audience that consumes questions. Kim demonstrates that learnersourcing [9], which refers to using a large group of learner input in order to improve course material and establishing a feedback loop between learners and the system, is incredibly valuable. Not only can learner interactions with a system be observed in order to determine how to make improvements, learners can participate actively as content creators.

2.1 Applications of Learnersourcing

In Kim’s paper on learnersourcing [9], he details two examples of learnersourcing in the context of educational videos. The first system he presents involves passive learner-sourcing, in which inputs obtained from learners as they use the system—such as when they tend to pause, rewind, and skip—is used in order to detect areas of particular interest and allow for easier navigation to and within such areas. The second type of learnersourcing is active—users were asked to annotate step-by-step how-to videos by

providing timestamps, textual descriptions, and before and after images, and those results were combined to generate user interfaces that provided an overview of the steps and facilitated jumping to particular subtasks.

Learnersourcing has also been used in a variety of university courses. One possible usage is creation of materials for the course. Tarasowa et al. created SlideWiki [17], which allows students to collaboratively work on lecture slide decks by creating new slides, editing the text on a particular slide, and reordering slides. In a framework used by an introductory Information Systems course at the University of Cologne, students write problems and solutions, which are then reviewed and improved by other students before being assembled into exams [10].

Other systems have used student input to supply personalized hints and feedback. Heffernan et al.’s ASSISTments system [8] enables learners to solve practice problems, request hints when they are stuck, and leave comments. Course staff can view the most common wrong answers, then add hints and feedback that are shown specifically when future students demonstrate the same misconception. An example of active learner-sourcing with a similar goal is Glassman et al.’s Dear Beta system [5], in which students who recently eliminated a bug, as detected by a change in their code’s performance on teacher-designed test suites, are prompted to create hints about how to fix that problem. Students who encounter similar issues are shown these hints automatically, and can upvote helpful ones. Additionally, a mandatory self-reflection activity requires students to write hints that help others optimize their code. These hints are shown upon request, rather than automatically.

Of particular relevance to this thesis is using learnersourcing to generate student-written practice problems. For example, Denny, Cukierman, and Bhaskar [3] established a program that enabled students to devise short-form coding problems, example solutions, and test suites as part of preparation for an exam. All students were then

required to solve some of the problems created by their peers. In Mitros' pilot study [13], high-achieving students in an introductory electronics class were able to produce high-quality complex design questions after attending a short course about how to write good autograded assessments and participating in a discussion forum where other students and staff members could comment and collaborate on draft questions. Lastly, PeerWise, a platform initially created by Denny, Hamer, and Luxton-Reilly [4] but since used in many other university courses [7], enables students to create multiple-choice questions and explanations, and answer, discuss, and rate others' submissions.

2.2 Evaluation of Learnersourcing

The majority of experiments conducted on learnersourcing aimed to discover its effect on grade performance and/or exam performance, and found that they generally improved with increased engagement in learnersourcing activities [3, 4, 7, 17]. For example, a review of PeerWise usage in five science courses at Edinburgh, Glasgow and Nottingham found positive correlations between the numbers of questions authored, answers submitted, comments posted, and days of activity with exam scores [7]. Furthermore, Denny, Cukierman, and Bhaskar conducted a randomized experiment which divided an approximately 200-student introductory programming course into two groups, both of which answered student-generated practice questions but only one of which actually authored said problems, and found that the average score of students who invented exercises was nearly 14% higher than the other group's on the next exam [3]. Thus, they demonstrated that the learning materials and feedback that a learnersourcing framework produces are not the only means of benefiting the students. The act of generating input for a learnersourcing framework is helpful in itself.

Many of these studies also attempted to gauge how the students felt about the

learnersourcing opportunities. One way of assessing student sentiment was exit surveys. These questionnaires usually indicated that students enjoyed the learnersourcing activities and appreciated their outputs. The most common benefits of learnersourcing that the students mentioned in their surveys are the wide variety and large quantity of practice opportunities [3, 17], the hints and prompt feedback [3, 5], and the viewing of their peers' approaches and solutions to problems [3]. A second way of analyzing student opinions is the amount of voluntary participation—the level of engagement with the platform beyond what contributes to their grades. Learnersourcing activities usually elicited more participation than was required for class, implying that students found the activities fun and/or rewarding [3, 4, 7, 17].

2.3 The Importance of Moderation

In experiments involving active learnersourcing, researchers were often curious about the quality of learner output. Many of them went through the output in order to assess its quality. Some approaches include rating it on a 0-5 scale [3, 7], comparing it to outputs produced by experts [9], or qualitatively evaluating it [13]. All of these approaches found that learner output, on average, is of moderate quality. In cases where students produced practice questions for their peers, most problems were found to be of medium difficulty or above.

Although learnersourced output is, on the whole, decently good, it is possible to increase average quality through moderation, which may result in the elimination of bad submissions and/or improvements to each submission. The amount and kind of moderation used by each system varies. The most common approach was to allow students to upvote and downvote each others' submissions [5, 8] or rate submissions on a point scale [4, 7, 10, 17]. Some experiments allowed students or instructors to

comment on the output [4, 7, 13]. Other processes allowed students to iterate on each others' work [9, 13, 17]. While some authors believe that instructor moderation is time-consuming, thus somewhat neutralizing one of the main benefits of learnersourcing [3], others allow instructors to drop content from the system [8] and some require instructor vetting of all exercises [10]. How best to assure quality of crowdsourced output is a more general question which has been studied extensively outside of educational settings. Some ideas include limiting the pool of users who can contribute by using reputation scores or other credentials, or checking the results with expert review or majority consensus [1].

2.4 Existing 6.031 Tools

The work presented in this paper largely builds on two frameworks that were previously made for, and are currently used in, the MIT course 6.031: Software Construction. The first is Java Tutor [14], which prompts students to complete a concept map by solving short, fill-in-the-blank coding exercises. If a student gets a problem wrong, the student attempt is compared to an instructor-provided regex-to-hint mapping, and an appropriate hint is shown along with some tutorials. Once the student solves the question, they are given an explanation to reinforce the concept they just learned. However, the Java Tutor framework was not built to easily support changes to the set of exercises. Without the exercise bank, making any modifications involved working with an extremely large data file, and previewing the output required the setup and running of a local development server. As a result, there was a lot of room for potential improvement by making it easier to iterate on the exercise content.

The second tool that this work directly builds on is Questionable [11], a website which allows students to improve their grades by answering and providing feedback on

several multiple-choice questions—most of which were written by students as part of participation in Questionable—and then authoring and submitting one multiple-choice question of their own. The benefit of Questionable largely comes from the testing effect, the fact that answering questions about material improves learning more than studying it for the same amount of time [15]; the advantages of peer assessment, which has been shown to improve students’ critical thinking, communication and collaborative skills [2, 16]; and the advantages of creating practice problems [3]. However, this large collection of student-authored multiple-choice questions was not made available for practice.

Chapter 3

Design

The exercise bank system needs to facilitate authorship of and iteration on class exercises in a wide variety of situations. Thus, this section presents a conceptual model which emphasizes the development of a problem over time. A single exercise is not represented as a static object, but rather a collection of versions in an assortment of states, each of which corresponds to a different level of modification and/or quality. The operations that can be performed on an exercise cause different transitions between these states.

3.1 Goals

An exercise bank implementation must store information about an exercise and its development. It must then present this data to the users and provide them with ways to interact with and modify it.

Our representation of exercise content should reflect not just its current text and state, but also its history. Being able to see the past versions of an exercise is important for understanding its trajectory and enabling further improvements. By looking at the

history of an exercise, a user can see the differences between one revision and the next. This allows them to determine the issues that each edit was attempting to address and thus pinpoint possible areas of continued improvement, as well as problems that may have arisen with the most recent update. Furthermore, the ability to reference past attempts at writing the question allows users to more easily avoid the issues that affected previous iterations.

The user interface of the exercise bank is critical to its success. In order to reduce the amount of time that is required for a new author to begin contributing, or the amount of effort needed to produce a new iteration on a previous exercise, the tool must be easy to understand and use. It should be clear to each type of user which operations they are allowed to perform, and how to accomplish each one. Furthermore, the platform should facilitate iteration by calling attention to exercises that may be in need of fixing or topics that are currently lacking in coverage.

3.2 Operations

The evolution of an exercise can be thought of as transitions between several states, from an initial concept to a published exercise. However, even a published exercise can and should continue to be modified. These states do not form a linear pipeline that eventually produce a finished product, but rather form miniature cycles of revision and improvement that will be traversed many times over the lifetime of an exercise. The transitions between states are supported through several operations:

- **suggest-add**: creates a suggestion for a new exercise, which must be approved through **add**.
- **add**: puts an exercise into the exercise bank.

- **suggest-modify**: creates a suggestion for changes to an existing exercise, which must be approved through **modify**.
- **modify**: makes changes to an existing exercise.
- **publish**: makes an exercise that is currently present in the bank available for public viewing.
- **attempt**: records a user's attempt at answering an exercise.
- **unpublish**: temporarily prevents users from answering an exercise until it is revised.
- **delete**: eliminates an exercise from the exercise bank. Unlike with **unpublish**, it is not expected that the exercise will be revised and made available again for answering.

The exercise bank system allows any number of groups of users to be created and permits each group to be given any set of the above permissions.

To clarify the use of the operations in the exercise bank, consider the following example:

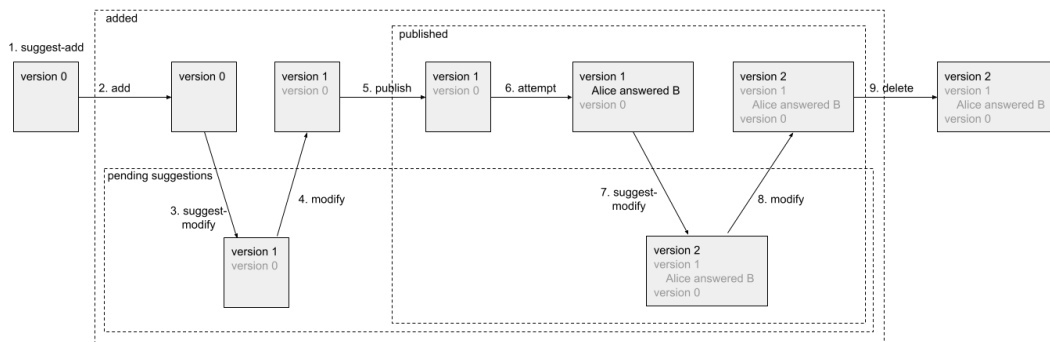


Figure 3.1: State transitions, the operations causing them, and the contents of an exercise over time.

1. Someone creates a new exercise and suggests that it be added to the exercise bank. The contents of this exercise are currently portrayed in Figure 3.1 by **version 0**.
2. The exercise is added to the bank.
3. A suggestion for an edit to the exercise is made. The updated contents are represented by **version 1**, and the history of the exercise is grayed out below.
4. The suggestion is accepted.
5. The exercise is published.
6. Alice attempts the exercise, and information about her attempt is stored. This information is associated with **version 1** of the exercise, since that was the version Alice saw.
7. Someone suggests an edit to the exercise. Since this is only a suggestion for the moment, anyone attempting the question will continue to see **version 1**.
8. The suggested edit is accepted. Now, attempters will see **version 2**.
9. The exercise is deleted from the bank.

3.3 Git as a Version Storage Mechanism

One natural way to track and interact with all of the iterations of an exercise over time is Git. Git is designed for making incremental improvements to software, and it therefore has many useful capabilities, such as the ability to store version history and create multiple branches. Hosting Git repositories on GitHub provides a web interface that allows contributors to create commits and pull requests. It also provides APIs and

permits users to set up webhooks to respond to certain events and create organizations containing roles with different permission levels. Additionally, Git and GitHub should already be familiar to most members of a computer science course staff, and possibly some of the students, which reduces the learning curve for new authors.

3.4 Use of Git in 6.031

It is easy to use Git as database that easily supports collaborative and iterative development, and it is easy to draw parallels between parts of a Git repository, such as the main branch or an unmerged branch, and the possible states for an exercise, as detailed in Section 3.2. To illustrate this, this section describes the operations which each group of participants in 6.031 have permissions for, as well as the Git data flows in both of the implemented exercise banks.

For fill-in-the-blank exercises, instructors have permissions for **publish**, **add**, and **modify**; teaching assistants (TAs) have **add** and **modify**; and students have **suggest-modify** and **attempt**. This means that the standard workflow is as follows. To add a new exercise, either an instructor or TA composes the exercise and commits it to any branch. To publish the exercise, an instructor merges it to the main branch, and it is automatically published. Students can then view the exercise and communicate suggestions about modifications to the staff. A TA can then create a new branch with some changes, and an instructor can merge it (thus updating the published version). This new workflow is shown in Figure 3.2.

For the exercise bank of student-authored MCQs, course instructors and TAs have **publish**, **add**, and **modify**; students have **suggest-add**, **suggest-modify**, and **attempt**. Thus, the workflow is as follows. New exercises can either be committed on the main branch and automatically published by TAs and instructors, or they can start out as

student-authored commits on non-main branches and then be merged into main. Students can also suggest edits in a similar fashion, by committing to a non-main branch which may later be approved, thus updating the published version of the exercise. This new workflow is shown in Figure 3.3.

3.5 User Interface

In order to streamline the processes of exercise authorship and iteration as much as possible, it is important that students and staff members easily understand and intuitively interact with the exercise bank. This section details the user interfaces for both systems as well as the workflows for students and staff members.

3.5.1 Fill-in-the-Blank Exercises

Prior to the implementation of the exercise bank for fill-in-the-blank exercises, the workflow for TAs was as follows:

1. In a local copy of the repository, edit concept map and exercise files
2. Run an upload script which informs the user about any errors, and if none occurred, uploads the changes to a local database
3. Preview changes on a local development server
4. Commit to a non-main branch in GitHub and make a pull request

(See figure 1.1 for a pictorial representation of this workflow.)

Staff members were able to discover which problems were in need of attention through student feedback. If a student clicked on a "report a problem" through the interface they used to complete the exercises, an email containing the written feedback,

as well as automatically-produced data about their interactions with the exercise, would be sent to one of the instructors. The instructor would then relay this information to the TAs and request that they re-examine the potentially problematic question.

In order to reduce the workload and learning curve for both TAs and instructors, this thesis implemented automatic processing and serving of draft versions of exercises and improved the email alert system. After a TA pushes a new version of an exercise to the repository, if no errors occurred, they can then preview it using the same interface that students use to complete exercises. This interface is shown in 3.4. However, if an error does occur, the TA and an instructor receive an email alert describing the problem. An example error alert is shown in 3.5.

3.5.2 Exercise Bank for Student-Authored MCQs

This exercise bank, unlike the one for fill-in-the-blank exercises, did not have an existing user interface. Thus, the exercise bank needed to provide students with an easy way to perform exercise bank operations. In addition, students should be able to conveniently perform activities that do not directly correspond to the operations outlined in Section 3.2 but are helpful in reviewing and enhancing their learning. One example of such an activity is revisiting problems that they had attempted previously. The exercise bank should also facilitate interaction with staff members, rather than having them interact directly with Git, so it provides web pages for staff member operations as well. To provide user interfaces for both student and staff, the exercise bank uses a web app based on Questionable [11].

3.5.2.1 Students

Students have `suggest-add` permissions on the exercise bank. A suggestions for a new exercise is created whenever a student submits an MCQ through Questionable.

Staff members later use the exercise bank interface to review and potentially approve these student-authored MCQs. Additionally, students use the exercise bank interface to request questions for practice, revisit questions that they have attempted before, and suggest edits to or leave feedback comments on existing questions.

The home page of the exercise bank, as shown in Figure 3.6, consists of two parts. The first is a dropdown menu that allows students to select a topic that they wish to review. Each topic is prefixed with a number, which is the lecture number that covered that topic during the current semester. The mapping of lecture numbers to topics is not constant between semesters, and is generated based on database information populated prior to the beginning of the semester. The second is a list of questions that the student has attempted, in most-recent to least-recent order. Each item contains the topic; the filename of the exercise, which contains some keywords from the exercise, as described in Section 4.1.2; and the time of the most recent viewing of the question.

Selecting any item from the dropdown menu and clicking "next" brings the student to a separate page, shown in Figure 3.7. This page has three exercises about the selected topic. The exercise bank system selects questions by finding all questions about that topic, then sorting them in ascending order of number of attempts by the querying student and selecting the first three. As a result, the student is shown all questions at least once before any questions repeat. Furthermore, once they have seen all questions, they do not see any given question at a disproportionate frequency. Beneath each question are some options to leave feedback on the question or an edit to the question.

Clicking on the "leave feedback" option creates a text box, along with associated "cancel" and "submit" buttons. Clicking on "cancel" gets rid of the text box and buttons, while typing a message and clicking on "submit" will disable the text box, hide the buttons, and show a confirmation message to the student. The text box, both

before and after feedback is submitted, is shown in Figures 3.8 and 3.9.

Clicking on the "suggest an edit" option pops up a modal that allows the student to make and preview edits to the question, shown in Figure 3.10. At the top of this modal is a text box, populated with the handx source of the question. Underneath that are two previews—one that displays the question without its explanation, and the other that shows the explanation as well. At the very bottom is a checklist which asks students to verify that their submission is formatted correctly, and buttons to cancel or submit. This interface is very similar to the one that allows students to submit MCQs in Questionable, and is thus familiar to at least some of them. Furthermore, including the two previews and encouraging students to double-check their formatting should help them avoid formatting errors.

From the home page, if a student chooses to revisit an exercise, they are brought to a page containing a single exercise. Below it are options to add feedback and suggest edits. Clicking on either of these results in behavior very similar to that described above. Once the student is done looking at that exercise, they can go back to the home page or click on navigation arrows in the header to go to another one of the questions they have tried before.

Students are forbidden from submitting feedback on or suggesting edits to exercises for which they have not read the explanation. This is because it is possible for a student to think the question is wrong when in fact they have a misconception that the explanation would have corrected. The user interface also prevents students from submitting a suggested edit without checking all of the items in the formatting checklist. The alert boxes which appear in these scenarios are shown in 3.11 and 3.12.

3.5.2.2 TAs and Instructors

TAs and instructors have the same permissions, so they are collectively referred to as staff members for brevity. Staff members have access to several pages devoted to the administration of the exercise bank. Most staff members start out at a simple landing page with links to and descriptions of the other pages. A link to this page is located at the top right of all of the administration pages, making navigation back to this page and from there to other pages easy and intuitive.

Staff members must be able to add new exercises to the bank. To do this, they can visit an add exercises page, which is depicted in Figure 3.13. At the top is a box that allows a staff member to specify which semester they are interested in taking submissions from. The default, which is populated upon page load, is last semester. Below that is a table where each row contains a topic and a status, which is one of accepted, rejected, or blank. The last of these is used for unreviewed exercises. The columns each have sorting and filtering capabilities.

Clicking on a row brings the staff member to another page, which allows them to evaluate an individual exercise. This page is shown in Figure 3.14. Arrows at the top and bottom of the page allow navigation to the next and previous exercises in a chronologically sorted order. In the left column is a preview of the exercise, which updates automatically based on the text in the box below. Much as students can type and preview their changes as they write, as described in Section 3.5.2.1, staff members can make modifications to the student submission before adding it to the bank. On the right is information extracted from graders and other students. Any comments that were provided by the staff member who graded the submission, either to the student or to other staff members, are shown at the top and labeled as a staff comment. Below that is a series of cells, each representing a student interaction with the question. Within each cell are columns of check boxes, where each check corresponds

to a correct answer from an attempt to answer the multiple-choice question. To the right of the check boxes is a comment written by the student about the question. At the very bottom of the page is a comment box for the reviewer to write their thoughts about the suitability or unsuitability of the exercise, and buttons to accept or reject it from the bank.

Once an exercise has been added to the bank, it needs to be reviewed and iterated on. The review exercises page, seen in Figure 3.15, allows staff members to identify exercises that are most in need of improvement. It consists of a table where each row corresponds to an exercise. In this row are the question topic, the timestamp of the most recent review, a list of links to pull requests created by students when they suggest edits, the number of distinct users that have answered the question, the percentage of those users that answered it correctly on the first attempt, and the average number of attempts taken before a correct answer. The last three statistics only deal with the most recent revision of the exercise. This is because a problematic question, with low success rates, may be improved substantially after a single revision, and the new version of question should not receive undue attention.

From the review exercises page, clicking on one of the rows navigates to a page which allows the user to evaluate a single exercise, pictured in Figure 3.16. On the left of this page is a preview of the question, which automatically renders the handx source code in the box below it. The next column contains links to the GitHub repository's pull request review pages, and the last column contains comments and checkboxes representing student attempts to answer the question. At the bottom of the page are buttons to update the handx of the question, mark the exercise as "reviewed", or remove the question from the exercise bank entirely (if this happens, the question will be deleted from the file tree of the Git repository, but its version history will still be visible).

Lastly, a user statistics page presents information about the various users who have attempted practice questions through the exercise bank. For each user, it lists their full name, their username, which semesters (if any) when the user was a staff member of 6.031, the number of readings practiced, total number of questions answered, and percentage that the user got correct. This page does not facilitate any of the exercise bank operations, but rather exists so that the maintainers of the exercise bank can get an overview of how many students are using it and what their patterns of usage are.

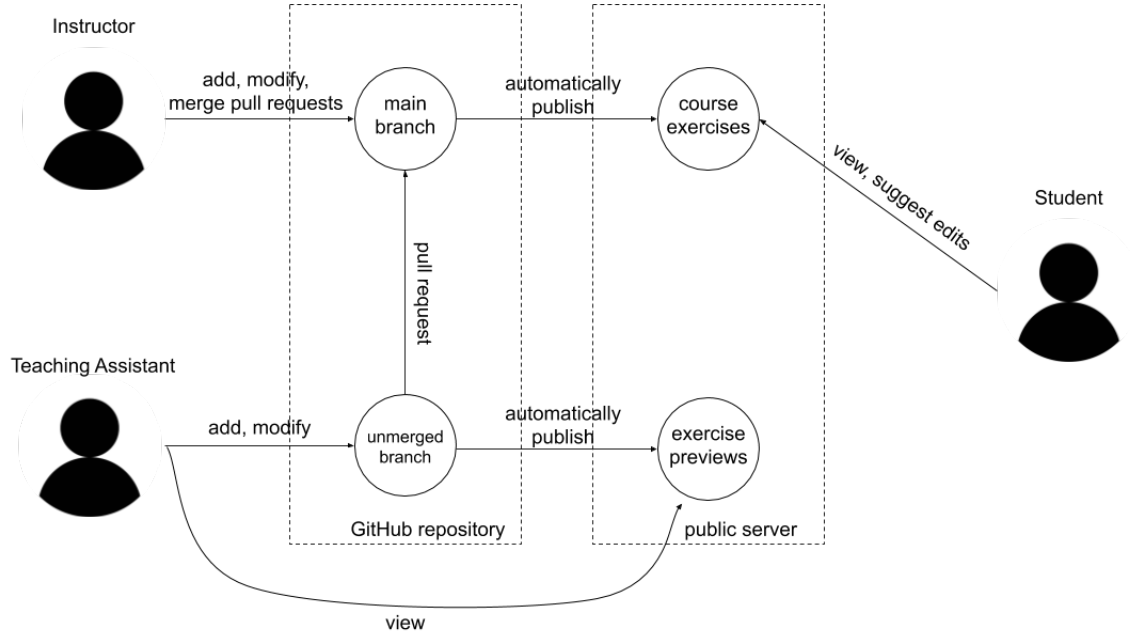


Figure 3.2: The states, data flows, and operations for fill-in-the-blank exercise questions after the introduction of the exercise bank system. Contrast this with 1.1.

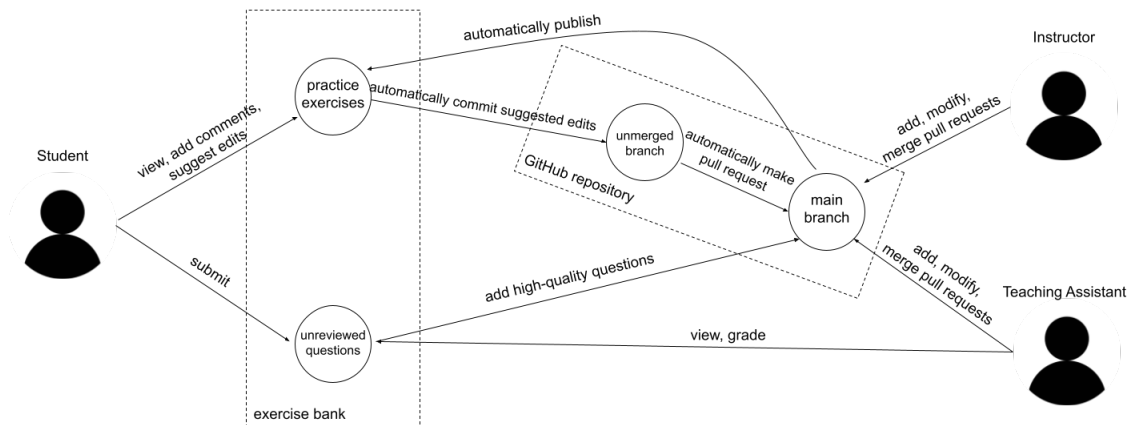


Figure 3.3: The states, data flows, and operations for submissions to the Questionable Makeup system after the introduction of the exercise bank system. Contrast this with 1.2.

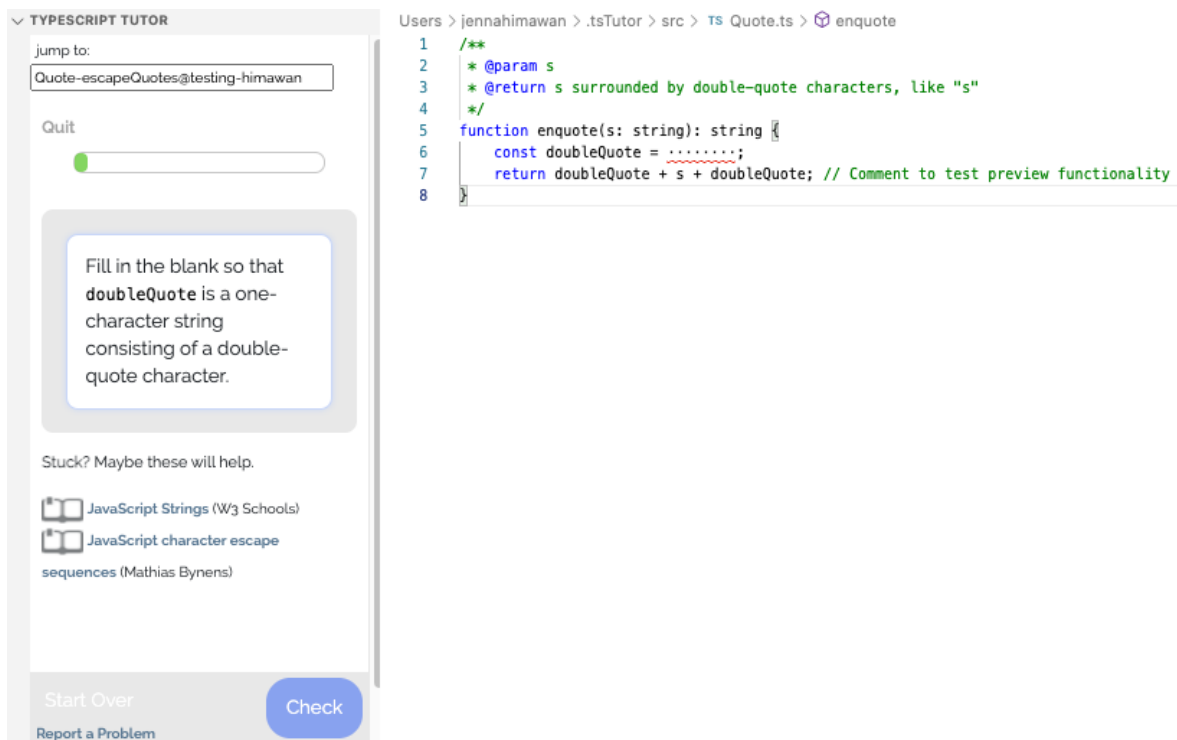


Figure 3.4: An example of the VSCode-based interface for playtesting the exercise `Quote-escapeQuotes` (see figure 4.4 for the complete metadata) on the branch `testing-himawan`. Note the "jump to" box at the upper left, containing the generated ID of the exercise.

```

Error: YAML error in src/Quote.ts: All collection items must
      start at the same column at line 1, column 5:

- id: Quote-escapeQuotes
  ~~~~~...

at parseYAML (upload-exercises.js:927:15)
at upload-exercises.js:980:41
at Array.forEach (<anonymous>)
at loadEmbeddedExercises (upload-exercises.js:975:73)
at loadExerciseFile (upload-exercises.js:958:35)
at loadAndUpdateExercises (upload-exercises.js:1003:19)
at main (upload-exercises.js:1063:15)
at Object.<anonymous> (upload-exercises.js:1077:5)
at Module._compile (internal/modules/cjs/loader.js:1063:30)
at Object.Module._extensions..js (internal/modules/cjs/
  loader.js:1092:10)
at Module.load (internal/modules/cjs/loader.js:928:32)
at Function.Module._load (internal/modules/cjs/loader.js
  :769:14)
at Function.executeUserEntryPoint [as runMain]
(internal/modules/run_main.js:72:12)
at internal/main/run_main_module.js:17:47

```

Figure 3.5: An example email about incorrect indentation in a concept map file. Note that the problematic filepath, line number and column offset, as well as a preview of the erroneous section, are provided.

6.031 Practice Exercises

Please select a class

Choose the class you want to get practice questions for.

Next

Practice History

Below are the questions that you have previously answered.

Class	Exercise Nickname	Most Recent Viewing
01: Static Checking	behave-java-int	2/10/2021, 12:56:56 PM
01: Static Checking	kind-error-will	2/10/2021, 12:56:45 PM
01: Static Checking	error-does-snippet	2/10/2021, 12:56:35 PM
01: Static Checking	sequence-operations-final	2/9/2021, 3:46:45 PM

Figure 3.6: The home page of the exercise bank, containing a dropdown to choose a topic, as well as a list of previously-attempted questions, in reverse chronological order.

Question 1

The code below aims to store the Hailstone sequence starting from 3 in the array `a` :

```
int[] a = new int[5];
int i = 0;
int n = 3;
while (n != 1) {
    a[i] = n;
    i++;
    if (n % 2 == 0) {
        n = n / 2;
    } else {
        n = 3 * n + 1;
    }
}
a[i] = n;
```

What type of error (if any) occurs when the above code is run?

- ☐ No error and correct output
- ☐ No error and incorrect output
- ☐ Static error
- ☐ Dynamic error

CHECK

Click here to [leave feedback](#) on or [suggest edits](#) to this exercise.

Question 2

What line will produce what kind of error (if any)?

```
String i = new String("go");
```

Figure 3.7: The practice page, containing three multiple-choice questions from the exercise bank. Note that beneath each problem are some options to leave feedback or directly make suggestions about edits.

Question 1

The code below aims to store the Hailstone sequence starting from 3 in the array `a` :

```

int[] a = new int[5];
int i = 0;
int n = 3;
while (n != 1) {
    a[i] = n;
    i++;
    if (n % 2 == 0) {
        n = n / 2;
    } else {
        n = 3 * n + 1;
    }
}
a[i] = n;

```

What type of error (if any) occurs when the above code is run?

☒ No error and correct output
☐ No error and incorrect output
☐ Static error
☒ Dynamic error

The sequence starting from 3 has length 8, but array `a` has size 5, so the index `i` would be out-of range, which is a dynamic error.

CHECK

EXPLAIN

Click here to [leave feedback](#) on or [suggest edits](#) to this exercise.

Cancel

Submit

Figure 3.8: After a student clicks on the "leave feedback" option, a box appears underneath the question.

Click here to [leave feedback](#) on or [suggest edits](#) to this exercise.

I don't think there should be a hyphen in "would be out-of range".

Thank you for submitting feedback.

Figure 3.9: Once a student clicks "submit", the box turns gray and the cancel / submit buttons disappear.

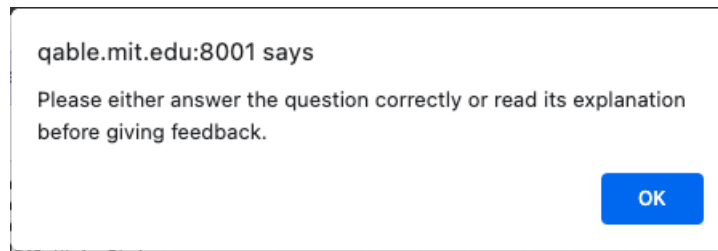


Figure 3.11: If a student does not attempt a question, they are prevented from making comments or suggesting edits, and a browser alert appears.

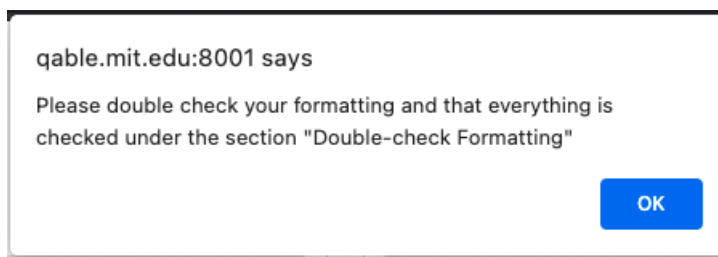


Figure 3.12: If a student does not complete the formatting checklist, they are not allowed to suggest an edit.

Semester to take exercises from:

fa20

Class	Status
14: Recursion	Contains
14: Recursion	Filter...
15: Equality	accepted
15: Equality	rejected
15: Equality	accepted
15: Equality	accepted

Figure 3.13: The add exercises page.

22: Message-Passing (fa19)

status: accepted

Message-Passing

What can be changed for this code to run correctly? Consider each answer choice independently.

```

BlockingQueue<Integer> requests = new ArrayBlockingQueue<>();
try {
    requests.put(123);
    System.out.println(replies.take());
} catch (InterruptedException ie) {
    ie.printStackTrace();
}
System.out.println("done");
System.exit(0); // ends the program
        
```

☐ use a LinkedBlockingQueue instead of an ArrayBlockingQueue

☒ initialize requests with a size

☐ use request.add() instead of .put

☐ state the type of the queue elements: ArrayBlockingQueue<Integer>

If we use a LinkedBlockingQueue, size will not have to be specified so the code will run correctly.
 If we continued with ArrayBlockingQueue, we would need to specify its fixed size
 BlockingQueues can use both .add or .put
 ArrayBlockingQueue can infer Integer from the left hand side

CHECK

Comments and Playtests (3)

I agree that asking for all possible fixes would be clearer, although personally, I understood the intended meaning. I think this could be a good question for comparing ArrayBlockingQueue and LinkedBlockingQueue with some additions e.g. showing how one could cause a deadlock using one instead of the other.
-staff comment

I don't think this is creative to be honest. Doesn't really test my understanding. Also, I would argue the selection is wrong because it seems like LinkedBlockingQueue alone will fix it: Question should have said select all possible fixes.
- [blurred]

It would be helpful to indicate that the answer choices should be considered independently of each other.
- [blurred]

Very interesting problem! I liked this one because it tests the difference between Linked Blocking and Array Blocking queue
- [blurred]

What can be changed for this code to run correctly? Consider each answer choice independently.

```

...
BlockingQueue<Integer> requests = new ArrayBlockingQueue<>();
try {
    requests.put(123);
    System.out.println(replies.take());
} catch (InterruptedException ie) {
    ie.printStackTrace();
}
System.out.println("done");
System.exit(0); // ends the program
...
        
```

☒ use a LinkedBlockingQueue instead of an ArrayBlockingQueue
☒ initialize requests with a size
☒ use request.add() instead of .put

This question has already been submitted to the exercise bank. If you would like to make changes, please visit [the exercise review dashboard](#).

If you do not believe this should be added to the exercise bank, please write a comment explaining why:

Reject from exercise bank
Add to exercise bank

Figure 3.14: A page for an individual exercise which is being considered for addition to the exercise bank. Student usernames have been blurred out.

Reading	Last Reviewed	Pull Requests	# Answerers	% Correct	Average # Attempts
Code Review	2/18/2021, 12:14:08 PM		1	0.00%	5.00
Mutability & Immutabil...	2/19/2021, 2:18:59 PM		2	0.00%	2.00
AF&RIs	2/10/2021, 5:04:05 PM		5	80.00%	1.20
Mutability & Immutabil...	2/19/2021, 1:34:32 PM	#1	5	20.00%	2.20
Recursion	3/5/2021, 9:37:08 AM	#1 , #2	7	71.43%	1.29

Figure 3.15: The review exercises page.

Mutability & Immutability

Read the code below to answer the question.

```
public static void main(String[] args) {
    List<Integer> list1 = Arrays.asList(1,2,3);
    List<Integer> list2 = new ArrayList<>(list1);
    list1.set(0, 5);
    System.out.println(list2);
}
```

What will the code print?

☐ [1,2,3]
☐ [5,2,3] ☒
☐ [0,2,3]
☐ will produce an error

It will not produce [1,2,3] because list2 is a SHALLOW copy of list1, meaning that if list1 is mutated, list2 will be mutated since it points > to the same object as list1.

For the reason explained above, setting index 0 to 5 in list1 will do the same to list2, thus yielding [5,2,3] for list2.

This is more of a way of throwing off the reader, we are setting index 0 to 5, we are not setting any element to 0.

While this code is not SFB because it is mutating a list that we did not intend to mutate, this code will still produce an output, just not a > correct one.

CHECK

Read the code below to answer the question.

```
public static void main(String[] args) {
    List<Integer> list1 = Arrays.asList(1,2,3);
    List<Integer> list2 = new ArrayList<>(list1);
    list1.set(0, 5);
    System.out.println(list2);
}
```

What will the code print?

☐ [] [1,2,3]
☒ [x] [5,2,3]
☐ [] [0,2,3]
☐ [] will produce an error

> It will not produce [1,2,3] because list2 is a SHALLOW copy of list1, meaning that if list1 is mutated, list2 will be mutated since it points > to the same object as list1.

>

Submit Handx Changes

Mark this exercise as reviewed

Remove this exercise from the exercise bank

Pull Requests (1) Comments and Playtests (3)

Pull Request #1

I'm confused bout why this isn't [1,2,3], because I ran the code in eclipse, and it actually returned [1,2,3], and not [5,2,3]. I get the premise of the question though.	<div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div>
-	
This question was very easy to understand. The key concept here is that list1 and list2 are aliases to the same (mutable) list, and thus we see [5,2,3] when we print list2, reflecting the operation list1.set(0,5). I think the explanation is easy to understand and clear, but it would be nice to also add the word "aliases" describing list1 and list2, since it is a key concept when talking about mutable vs. immutable data types.	<div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div>
-	
This was a good question but the explanation "just not a > correct one" is a little difficult to understand.	<div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div>
-	

Figure 3.16: The page which allows staff members to review a single exercise. Student usernames have been blurred out.

Chapter 4

Implementation

This section discusses the implementations of two exercise banks used in 6.031, for fill-in-the-blank exercises and student-authored MCQs. Both use Git to store exercise contents and histories, and use MongoDB to store the current state of the exercises along with other information that cannot easily be determined using Git. The fill-in-the blank exercises rely on a webhook that responds to push events to the Git repository and automatically reflects those changes in MongoDB to facilitate iteration and debugging. The exercise bank for student-authored MCQs uses a web app written in JavaScript with a NodeJS backend that uses MongoDB and calls to the GitHub API.

4.1 Data Storage

Both exercise banks implemented as part of the work of this thesis rely heavily on Git and on MongoDB for storage. This section discusses our approach and provides details about the way that these exercise banks organize their data, including directory structures, schemas, and models.

4.1.1 Fill-In-The-Blank Exercises

The data for a fill-in-the-blank exercise is stored in two places. The first is a concept map containing:

- names of concepts, such as `if::else-if`, `list::length`, and `return`
- levels, which are groups of concepts such as `basic`, `arrays`, and `functions`
- elaborations on tricky parts of each of these concepts, such as places where they differ from similar concepts in other languages
- tutorials with names, associated concept IDs, and URLs

Excerpts from a concept map can be seen in Figures 4.1, 4.2, and 4.3. The concept map is edited only when changes to the concepts or tutorials are being made, which happens relatively infrequently.

The second is an exercise file, consisting of code that will be presented to the student and some metadata at the top, which specifies the following:

- a name for the exercise
- a list of concepts that it covers
- a prompt to be shown to students that tells them the task they are expected to complete
- a list of code snippets to be turned into blanks for them to fill
- a list of regexes to match acceptable answers
- a list of regexes which match common incorrect answers, and a hint to show for each one

- an explanation to show to the student to reinforce their understanding after they complete the exercise

The rate of change of exercise files is likely much greater than that of the concept map. The most common operations are likely to be adding new exercises, which requires adding more source files or adding more blanks to existing code, editing an exercise prompt to reduce confusion, adding additional regexes for other acceptable answers, and supplying new regexes and hints for common mistakes. All of these leave the concept map untouched, instead modifying the exercise files.

The concept map and exercise files are stored in a Git repository. This provides many benefits for simultaneous collaboration and permissions for groups of users. GitHub also supplies a UI that easily allows for small edits, such as correction of typos, directly in the browser, as seen in Figure 4.5.

The server that hosts the exercises, however, does not directly use Git as its database, in part due to efficiency concerns but also because it needs to store information aside from the content of the exercises, such as which exercises a given student has completed. Thus, the server has a clone of the Git repository and a script that parses the concept map, populates a MongoDB database, and updates the source code for the exercises. This thesis did not modify this infrastructure and thus will not describe it in detail.

4.1.2 Exercise Bank for Student-Authored MCQs

The exercise bank for student-authored MCQs similarly uses Git to store the iterations of its questions. However, the exercise bank is built on top of Questionable [11], which relies on MongoDB and Mongoose. This section first outlines some relevant details of the existing MongoDB collections and Mongoose models, then discusses the

modifications that support integration with Git and GitHub as well as features that will be discussed later on in the Implementation section.

Student-authored MCQs are recorded in a **questions** collection in the Questionable database. The corresponding Mongoose model, **Question**, originally had the following attributes.

- **timestamp** (required): the **Date** of submission of the question
- **reading** (required): the ID of the reading (the topic) that the question deals with
- **handx** (required): the string content of the question, written in Handx [6], a Markdown-like language used to create multiple-choice questions in 6.031
- **grader** (optional): the user ID of the 6.031 staff member who graded the MCQ
- **gradeTimestamp** (optional): the timestamp at which the grader graded the submission
- **notesToStudent** (optional): notes that the staff grader provided to the student upon grading
- **notesToStaff** (optional): notes that the staff grader provided to future graders about the student's submission

Note that, in the Questionable database system, the **handx** field is the content of the MCQ. Since no user has any opportunity to change student-authored MCQs once they have been submitted, **handx** is simply text. However, associating each **question** document in Questionable with a file in GitHub would allow a constant exercise identity to have changing contents, be associated with a version history, and even have several

different versions in existence at a time. This thesis accomplishes this by adding optional `filename` and `handxTimestamp` fields.

The `filename` field is the name of the file in Git that is associated with the question. While the implementation could have used some form of UUID, instead the Git repository aims to be as easily human-readable as possible. This makes the system as a whole easier to understand, and it allows people to make edits directly through the GitHub web UI, as pictured in Figure 4.5. The exercise bank implementation divides the repository into directories, one per reading, and assigns a unique human-readable name to each file upon its addition to the exercise bank. This assignment is made by lowercasing the `handx` text and removing punctuation, then removing commonly-used words from the `stopword` Node library [12] and words that appear frequently in student-authored MCQs, such as "consider the code below" and "select the statements that best match", and then concatenating the first three words with hyphens as a delimiter. If a filename with this name already exists, a number is added onto the end of the name; this number is incremented until the filename is unique. Once this process is complete, the entire filepath can be constructed by using the name of the reading as the directory and the `filename` field as the name of the file within that directory. `filename` is null for all questions that have been submitted but are not part of the bank. The names of directories and files can be seen in Figure 4.6 and Figure 4.7.

The `handxTimestamp` is the timestamp of the most recent update to the file on the main branch. The timestamp is kept up-to-date via a webhook that is notified of pushes to the exercise bank Git repository.

In addition to the changes described above, which were required to introduce Git integration, some additional fields were necessary to support reviewing and revising exercises inside the exercise bank. The fields added to the `Question` model are as follows:

- **reviewStatus**: true if the submission was accepted to the exercise bank, false if it was rejected, null/undefined if there was no decision
- **reviewComment**: the comment provided by the staff member who reviewed an exercise, if any
- **reviewTimestamp**: the timestamp of the most recent review of an exercise

4.2 User Interface Implementation

This section discusses the implementation of the user interfaces for fill-in-the-blank exercises and the exercise bank for student-authored MCQs, which were discussed in Section 3.5. It also details some of the more challenging aspects of these implementations.

4.2.1 Fill-In-The-Blank Exercises

This thesis contributes automatic parsing, error reporting, and serving of draft exercises. This is accomplished by a webhook written in TypeScript that reacts to push events to a Git repository. When a push to branch **branchName** modifies the exercise with ID **exerciseName**, the webhook parses the changes. If an error occurs, the webhook finds the email addresses of the author and committer, then constructs an email containing the contents of `stderr` and sends it to them and one of the instructors. If no error occurs, the webhook creates a new exercise with ID **exerciseName@branchName**. The exercise author can then use the fill-in-the-blank exercise interface to playtest the question. Because of this automated system, previewing an exercise to verify its correctness no longer requires setting up a local database and development server. It even allows users to avoid cloning the repository at all. Instead, it is possible to make edits

directly through the GitHub UI, which is shown in Figure 4.5. The exercise author can iterate by editing, committing to an unmerged branch, looking at the exercise, and repeating until it looks correct. When a branch is deleted from the Git repository, the webhook reacts by finding all exercises whose IDs end in `@branchName` and deleting them from the system. As shown in Figure 4.8, the webhook can be monitored using the GitHub UI, which lists push events and responses.

4.2.2 Exercise Bank for Student-Authored MCQs

Students and staff members interact with the exercise bank for student-authored MCQs through a web app. This web app is built using HTML, CSS, and JavaScript. This section provides an overview of the implementation of the web app. This discussion divides the pages into two categories: action pages, which display content for one or a small set of exercises and allow users to perform operations on them, and navigation pages, which have lists of links to action pages.

The majority of action pages have a URL query parameter specifying an exercise ID. JavaScript uses this ID to fetch data from the API, then populates the page accordingly. The user can use buttons and textboxes on the action page to trigger post requests to the API and modify the exercise. The only exception to this pattern is the practice page, which presents the student with a set of three exercises. The JavaScript on the practice page makes a call to the API to fetch the three least-frequently seen exercises about the topic the student wants to practice, then displays and allows the student to interact with them.

Each navigation page contains a table consisting of links to action pages and information about each one. This additional information helps the user filter, sort, or search among the action pages to find the ones they need. The navigation page makes a call to the API to fetch data for each row of the table, then populates the table and makes

it visible to the user. The most challenging navigation page to implement was the one which allows the user to find exercises that should be revised. This page displays the percentage of students that answered each exercise correctly on the first attempt and the average number of attempts taken, but only considers statistics for the most recent version of the exercise. In the first attempt at implementing this page, the backend queried all attempt data, sent requests to GitHub API to determine the timestamp of the most recent exercise version, and calculated statistics using JavaScript. However, this proved to be too time-consuming, and the page did not load because the API consistently timed out. Adding a `handxTimestamp` field to the database and using a MongoDB aggregate query sped up this operation significantly and ensured that this page was usable.

```

if::else-if
  uiLabel: if/else if
  /java<-python:
    diff: shallow
    explanation: <p>Though the structure of
<code>if/else</code> statements is very similar
from Java to Python, it is important to remember
to use curly braces in Java, rather than colons
and indentation. </p><p>Also, where Python uses
the <code>elif</code> keyword, Java uses
<code>else if</code>.</p>

```

Figure 4.1: An excerpt from the concept map file, showing the if::else-if concept.

```

- id: basic
  uiLabel: Basic TypeScript
  conceptIds:
    - print
    - if
    - if::else
    - if::else-if

```

Figure 4.2: An excerpt from the concept map file, showing the Basic TypeScript concept group.

```

- url: https://www.tutorialsteacher.com/typescript/
  typescript-if-else
  title: TypeScript - if else
  conceptIds:
    - if
    - if::else
    - if::else-if
  source: Tutorials Teacher

```

Figure 4.3: An excerpt from the concept map file, showing a tutorial for if, else-if, and else in TypeScript.

```

- id: Quote-escapeQuotes
  conceptIds:
    - string::literal::escape::quotes
  prompts:
    - Fill in the blank so that
      <code>doubleQuote</code> is a one-character string
      consisting of a double-quote character.
  explanation: "<p>Just like in Python, single-quote
  marks in TypeScript can be used when you need to
  write a literal string containing a double-quote;
  besides escaping, they are interchangeable.</p>"
  blanks:
    - code: '"\''
      transformers:
        - r/\\//
      triggeredHints:
        - s/`/ Template literals (<code>`...`</code>)
          should be avoided in the cases where there
          are no contained placeholders. There is a
          simpler way to do this.

```


Figure 4.4: An example of metadata at the top of an exercise file.

↩ Edit file

🔍 Preview changes

Spaces4No wrap

```
1 //<yaml>
2 // - id: Quote-escapeQuotes
3 //   conceptIds:
4 //     - string::literal::escape::quotes
5 //   prompts:
6 //     - Fill in the blank so that <code>doubleQuote</code> is a one-character
7 //       string consisting of a double-quote character.
8 //   explanation: "<p>Just like in Python, single-quote marks in TypeScript can be
9 //     used when you need to write a literal string containing a double-quote;
10 //   besides escaping, they are interchangeable.</p>"
11 //   blanks:
12 //     - code: '"\"';
13 //     transformers:
14 //       - s/'"/'\\""/
15 //</yaml>
16 /**
17  * @param s
18  * @return s surrounded by double-quote characters, like "s"
19  */
20 function enquote(s: string): string {
21   const doubleQuote = "\"";
22   return doubleQuote + s + doubleQuote;
23 }
24
```



Commit changes

Update Quote.ts

Add an optional extended description...

☒ Commit directly to the `tutorials-himawan` branch.
☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changesCancel

Figure 4.5: The GitHub web UI for editing and committing a file.







Branch: main ▾	New pull request	Create new file	Upload files	Find file	Clone or download ▾
<div>  himawan Added handx for question programming-with-adts/when-comes-writing. Latest commit 2799790 8 days ago </div>					
<div> <div>  adts </div> <div>Added handx for question adts/true-abstract-data.</div> <div>2 months ago</div> </div>					
<div> <div>  afris </div> <div>Added handx for question afris/af-abstraction-function.</div> <div>29 days ago</div> </div>					
<div> <div>  avoiding-debugging </div> <div>Added handx for question avoiding-debugging/incomplete-function-solves.</div> <div>2 months ago</div> </div>					
<div> <div>  basic-java </div> <div>Added handx for question basic-java/liststring-cities-new.</div> <div>2 months ago</div> </div>					
<div> <div>  callbacks </div> <div>Added handx for question callbacks/server-final-int.</div> <div>9 days ago</div> </div>					

Figure 4.6: Some of the names of GitHub directories in the exercise bank repository










Branch: main ▾	6.031-java-exercise-bank / adts /	Create new file	Upload files	Find file	History
<div>  himawan Added handx for question adts/true-abstract-data. Latest commit ea51cab on Feb 18 </div>					
..					
<div> <div>  adt-operations-nonvoid </div> <div>Added handx for question adts/adt-operations-nonvoid.</div> <div>2 months ago</div> </div>					
<div> <div>  adt-string-true </div> <div>Added handx for question adts/adt-string-true.</div> <div>2 months ago</div> </div>					
<div> <div>  creator-operation-always </div> <div>Added handx for question adts/creator-operation-always.</div> <div>2 months ago</div> </div>					
<div> <div>  javadocs-java-sethttpsdocs </div> <div>Added handx for question adts/javadocs-java-sethttpsdocs.</div> <div>2 months ago</div> </div>					
<div> <div>  linked-documentation-collections </div> <div>Added handx for question adts/linked-documentation-collections.</div> <div>2 months ago</div> </div>					
<div> <div>  operations-present-adt </div> <div>Added handx for question adts/operations-present-adt.</div> <div>2 months ago</div> </div>					
<div> <div>  specs-adt-oddstring </div> <div>Added handx for question adts/specs-adt-oddstring.</div> <div>2 months ago</div> </div>					
<div> <div>  true-abstract-data </div> <div>Added handx for question adts/true-abstract-data.</div> <div>2 months ago</div> </div>					

Figure 4.7: The filenames of exercises about the topic of ADTs (abstract data types)

Recent Deliveries

✓

da5a3e72-a43d-11eb-88ca-9a70df604f82

2021-04-23 10:11:52

...

Request

Response **200**

Redeliver

🕒 Completed in 2.31 seconds.

Headers

Connection: keep-alive

Content-Length: 50

Content-Type: text/html; charset=utf-8

Date: Fri, 23 Apr 2021 14:11:55 GMT

Keep-Alive: timeout=5

X-Powered-By: Express

Body

Processed deletion of testing-himawan successfully

✓

c9f8f1be-a43c-11eb-9329-09d0932c100b

2021-04-23 10:04:15

...

Request

Response **200**

Redeliver

🕒 Completed in 10.1 seconds.

Headers

Connection: keep-alive

Content-Length: 52

Content-Type: text/html; charset=utf-8

Date: Fri, 23 Apr 2021 14:04:26 GMT

Keep-Alive: timeout=5

X-Powered-By: Express

Body

Processed push event to testing-himawan successfully

✓

f00fadcd-a365-11eb-990f-2f3d78ced829

2021-04-22 08:26:17

...

✓

ef80cf7e-a365-11eb-8997-4f4dec536e60

2021-04-22 08:26:17

...

Request

Response **200**

Redeliver

🕒 Completed in 0.12 seconds.

Headers

Connection: keep-alive

Content-Length: 20

Content-Type: text/html; charset=utf-8

Date: Thu, 22 Apr 2021 12:26:17 GMT

Keep-Alive: timeout=5

X-Powered-By: Express

Body

No events to process

Figure 4.8: The webhook information page on GitHub. The push events that are sent to the webhook are listed in a most-recent-first order. The webhook prints short acknowledgements to confirm successful receipt and processing of branch deletions, pushes to new branches, and pushes that are not relevant to the fill-in-the-blank exercises.

Chapter 5

Evaluation

This section discusses the outcomes of our experiments in deploying the exercise banks in 6.031: Elements of Software Construction. The exercise bank for fill-in-the-blank exercises was used to add and modify numerous exercises for use in the course, and the staff found that the automatic alerts and draft previews were useful.

The exercise bank for student-authored MCQs was used by a large number of students, and saw an extremely large spike before an exam 6.031 held during the middle of the semester. Furthermore, it succeeded in generating exercises from a large number of student-authored MCQs, and also encouraged iteration on practice questions for the students.

5.1 Fill-in-the-Blank Exercises

Over the course of the spring 2021 semester, two instructors, five TAs and one lab assistant (LA) worked with the exercise bank for fill-in-the-blank exercises. In total, the non-instructors made 71 commits, adding 106 new exercises and 62 new tutorials, and modifying 32 existing exercises and 12 existing tutorials. This helped to balance

the work of maintaining the exercise bank between the instructors and other staff members, in addition to providing a variety of perspectives and ideas to creative tasks such as exercise design.

Further, the exercise bank was intuitive for the other staff members to use, so it is unlikely that the benefits obtained from encouraging contributions from non-instructors were outweighed by the amount of time spent learning how to use the system. Additionally, staff members reported that the process for playtesting draft exercises was smooth and much less arduous than setting up a development server. One of these staff members reported that they were quite unfamiliar with YAML, and receiving email alerts was helpful in their debugging efforts.

5.2 Student Practice with Student-Authored MCQs

Student interest in the exercise bank for student-authored MCQs has proven to be relatively high, especially considering that participation in it is entirely optional. 106 distinct students made 2525 attempts at answering questions in the exercise bank. Since approximately 250 students took 6.031 this semester, that means that about 42.4% of the class has attempted a question in the exercise bank at least once. This indicates that students are interested in an exercise bank and are willing to try it out.

Of the 106 students, 38 answered only 1 question, and 32 students have answered more than one set of 3 questions. These participation statistics seem to indicate that lots of students have tried the system, but not many of them were interested in actually completing an activity. However, a substantial number of students have answered upward of 20 distinct questions in the exercise bank, and one particularly notable student has answered 106 questions, using 516 attempts total. On average, students spent 2.5 attempts per question.

Figure 5.1 and Figure 5.2, track the number of attempts and unique users over time. In general, usage is somewhat low but consistent across days. Tickmarks every Sunday show the relationship between student behavior and the day of the week. Exercise bank activity seems to be slightly higher during the week than on weekends, but no other periodic patterns seem to be evident. The large spike in the graph was caused by Quiz 1, which occurred on 4/5/2021, the day after the peak. Since class is held at approximately noon Eastern, it is reasonable to assume that many students were practicing before the exam as well, hence explaining the very high number of users that day.

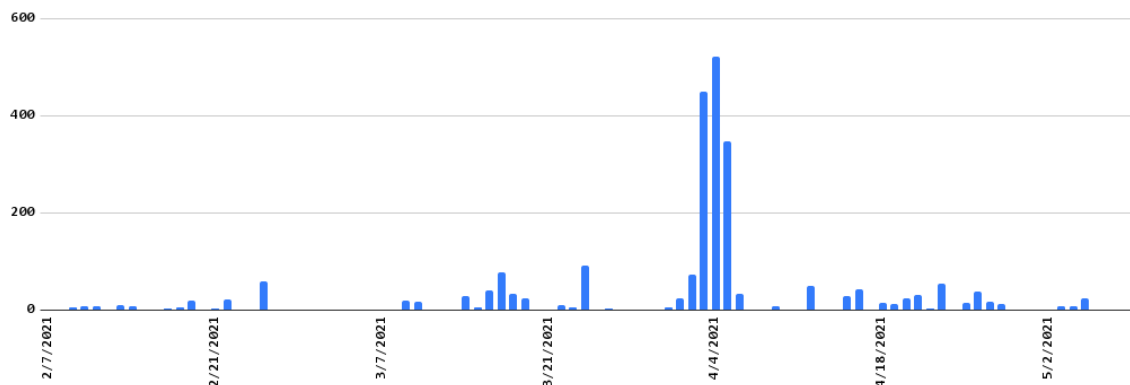


Figure 5.1: The number of attempts per day. Tick marks occur every Sunday. The large spike in the graph is caused by Quiz 1, which occurred on 4/5/2021.

5.3 Facilitation of Exercise Development

Student interest in the exercise bank, and reliance on it especially when it comes to test preparation, is clear. The following section examines how easy it is to add new exercises and make adjustments to existing ones.

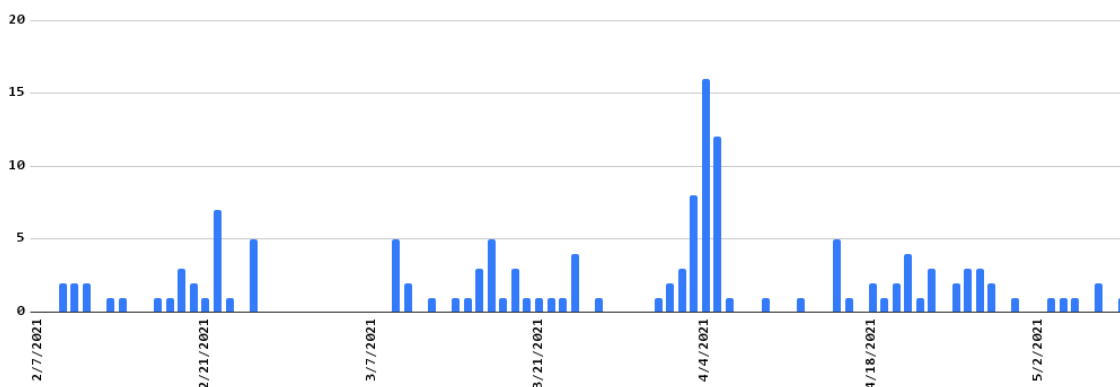


Figure 5.2: The number of users per day. Tick marks occur every Sunday. The large spike in the graph is caused by Quiz 1, which occurred on 4/5/2021.

5.3.1 Number of Contributors

The exercise bank drastically increased the number of exercise authors through learner-sourcing. The questions currently in the exercise bank are based on MCQs authored by 165 distinct students. In contrast, the 6.031 staff has about 20 members per semester. It is evident that a learnersourcing approach is much more scalable and much less time-intensive for staff members.

Staff members contribute to a question even after its original version is created. Seven TAs went through student-authored MCQs from previous semesters and added the ones that they deemed to have enough potential to the bank. They reviewed 641 exercises, of which they rejected 261 and accepted 380. TAs accepted only 177 exercises without modification. The subsequent section, Section 5.3.2, discusses the changes made to these exercises in more depth. The TAs were able to perform these tasks with minimal guidance, although there were initially a couple of questions about the user interface. On the whole, this was a very natural and smooth process.

Furthermore, staff members were not the only ones participating in the revision process. Although practicing with the exercise bank in general is optional, and suggesting an edit is not required in order to continue practice, some students were interested in

improving the exercises. Students created 15 pull requests through the suggest-an-edit feature. These suggestions are analyzed in the next section.

5.3.2 Amount of Change per Exercise

The exercise bank framework aims to make it easy to iterate on exercises but also encourage this behavior from its users. The difference between the original and most recent version of each student-authored MCQ determine whether it succeeds in these goals. Differences, when present, were labeled using the following categories. A *formatting change* involves fixing minor spelling errors or improper handx syntax. A *wording change* involves rewriting or reordering a couple of sentences to improve clarity. A *conceptual change* involves substantially modifying the structure of the exercise or the topics that it covers. Figure 5.3 shows an exercise that has undergone all three kinds of changes. The original version this exercise is displayed to the left of its most recent version.

The changes made by the TAs are broken down as shown in Figure 5.4. This includes both changes that were made at the time of an adding an exercise, and changes that were made later on. Since staff efforts were largely focused on adding new exercises to the bank this semester, only four changes belong to the latter category. Three of these are formatting changes, and one is a conceptual change. This last one was made to an exercise soon after it was accepted without changes, possibly accidentally. Although these changes are few, they demonstrate that TAs are occasionally willing to make small tweaks to exercises and add changes to an exercise despite mistakes in the initial review process.

In contrast, students submitted a very small number of suggested edits. The vast majority of these were formatting changes, but there were some wording and conceptual changes as well. The distribution is provided in Figure 5.5. Additionally, not all

What should the be filled in for the ???? in the following line of code if we wanted to add the numbers 0.0, 1.0, and 2.1 afterwards?

```
List<????> nums = new  
    ArrayList<>();
```

[] int
[] double
[x] Double
[] Integer

> Since we are explicitly stating that we want to add floating point numbers we

have to use the Double class since it is the wrapper for the primitive type double.

What should be written in place of ???? in the following line of code, if we want to add the numbers 0.0, 1.0, and 2.1 to the List 'nums'?

```
List<????> nums = new  
    ArrayList<>();
```

[] int
[] Integer
[] double
[x] Double

> Since we want to add floating point numbers, we can't use 'int' or 'Integer'.

> In Java, generic type arguments (including types inside of angle brackets in List declarations) must be reference types, not primitives. Thus, we have to use the 'Double' class, the wrapper for the primitive type 'double'.

Figure 5.3: An example of formatting, wording, and conceptual changes. On the left, the original handx source code. On the right, the revised version. Orange is used for formatting changes, yellow for wording changes, and blue for conceptual changes. Line breaks have been added to aid in comparison of the texts.

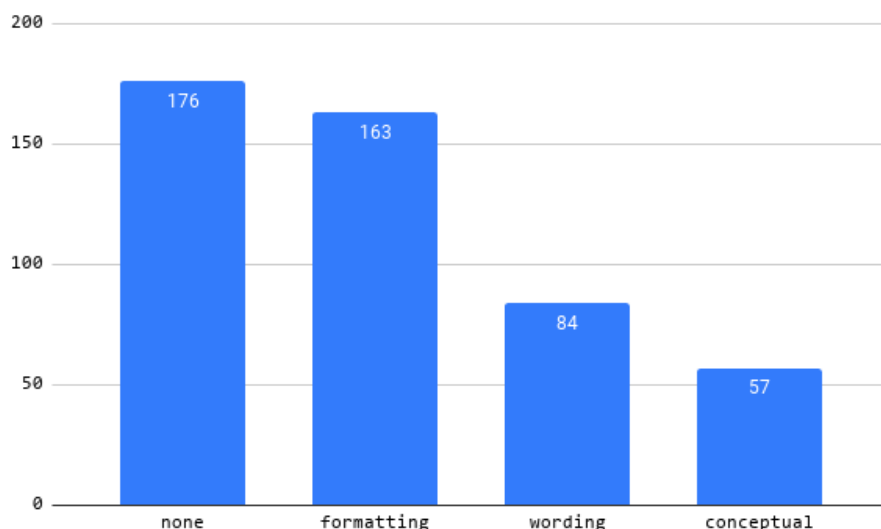


Figure 5.4: Changes made by TAs.

of the suggestions were accepted. The three rejected wording suggestions proposed the expansion of abbreviations or the addition of redundant information. While it is true that these changes may have helped the suggesters understand the question, they slightly decreased the quality and readability of the exercise for those familiar with commonly-used software engineering shorthand. It is worth noting that the small number of suggestions in general, and the low numbers of wording and conceptual changes, might be partly because TAs edited questions before students looked at them. This data indicates that students successfully find small problems that TAs have overlooked, and sometimes even correct larger issues.

TAs and students together made significant changes through the exercise bank. In fact, most of the exercises were modified at some point. The majority of changes were relatively minor, but the rates of rewording and conceptual changes are also promising. This data indicates that making changes to exercises is a natural part of the exercise bank workflow.

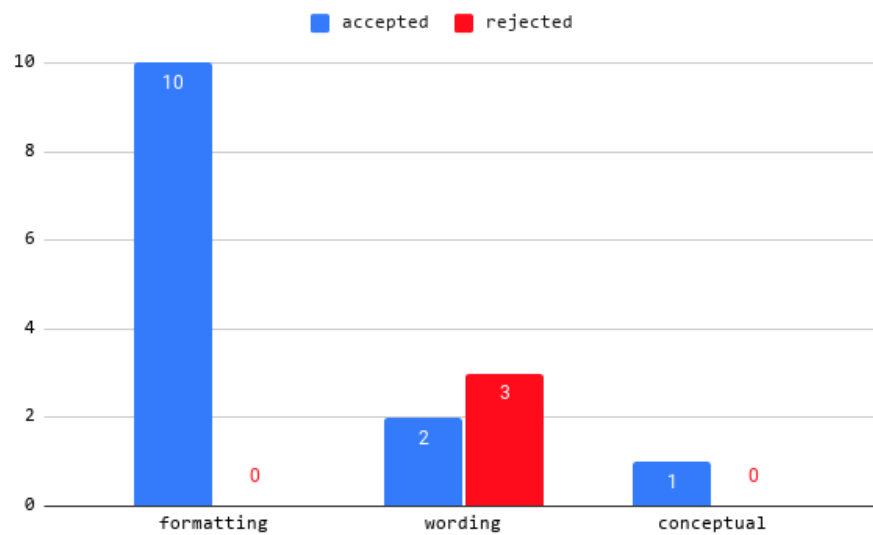


Figure 5.5: Changes made by students.

Chapter 6

Conclusion

This thesis presented the exercise bank framework, which aims to facilitate creation and revision of practice questions to provide to students. Notably, it provides a conceptual model which depicts an exercise as an object that transitions between different states—reflecting different steps in the initial writing and subsequent iteration phases—as its contents change. Two implementations of the exercise bank framework were deployed in MIT’s 6.031: Elements of Software Construction, an intermediate programming course with approximately 250 students per semester. The exercise bank was used heavily by students for test preparation, and even used outside of that, likely to help reinforce concepts from lectures and readings. The exercise bank was also successful in facilitating exercise authorship and iteration. The set of people who contributed to the exercise bank, through writing MCQs or suggesting edits, expanded from just two instructors to nearly 200 students and staff members. These contributors added much more exercise content and made substantial revisions to it in just one semester.

6.1 Further Work

Further improvements to the exercise bank can still be made. In regards to fill-in-the-blank exercises, staff members currently cannot easily determine which concept areas and exercises may be in need of improvement. Building a tool in order to figure out which concepts have very little coverage, which exercises have low correctness rates, or which incorrect answers are particularly common—thus allowing staff to supply hints and tutorials to correct those misunderstandings—might prove to be useful. The automatic email alert system could also be improved. Staff members expressed that the error messages were not easy to understand, and that there were sometimes large delays between an error being detected and an email being received.

In the domain of student-authored MCQs, `publish` and `unpublish` are currently the same as `add` and `remove` for the sake of simplicity. It might be helpful to temporarily prevent an exercise from being seen by students while it was worked on, rather than removing it entirely. Furthermore, accesses to GitHub are performed by using a Personal Access Token, which causes some information about the actual identity of the committer to get lost. The exercise bank UI could instead require staff members and students to log in with GitHub before performing operations. Additionally, the system which automatically generates names is somewhat flawed. The reader may recall that it selects keywords from the `handx` source after removing stopwords, but unfortunately it does not prevent the same word from being selected more than once, which led to the unfortunate filename of `music-music-music`.

Lastly, it is difficult to receive feedback from students about either of these exercise banks. Over the course of this semester, a couple of students posted about it on Piazza, a multipurpose ask-and-answer forum that is used for other queries in 6.031, but it would be much better to actively elicit feedback to find areas for potential

improvement.

Bibliography

- [1] Mohammad Allahbakhsh et al. “Quality Control in Crowdsourcing Systems: Issues and Directions”. In: *IEEE Internet Computing* 17 (2 2013), pp. 76–81. DOI: <https://doi.org/10.1109/MIC.2013.20>.
- [2] Philip C. Candy, Gay Crebert, and Jane O’Leary. “Developing Lifelong Learners through Undergraduate Education”. In: (1994).
- [3] Paul Denny, Diana Cukierman, and Jonathan Bhaskar. “Measuring the effect of inventing practice exercises on learning in an introductory programming course”. In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (2015), pp. 13–22. DOI: <https://doi.org/10.1145/2828959.2828967>.
- [4] Paul Denny, John Hamer, and Andrew Luxton-Reilly. “PeerWise: Students Sharing their Multiple Choice Questions”. In: *Proceedings of the Fourth international Workshop on Computing Education Research* (2008), pp. 51–58. DOI: <https://doi.org/10.1145/1404520.1404526>.
- [5] Elena L. Glassman et al. “Learnersourcing Personalized Hints”. In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (2016), pp. 1626–1636. DOI: <https://doi.org/10.1145/2818048.2820011>.

- [6] Max Goldman. *Markdown handouts + exercises*. GitHub repository. 2016. URL: <https://github.com/maxg/handx>.
- [7] Judy Hardy et al. “Student-Generated Content: Enhancing learning through sharing multiple-choice questions”. In: *International Journal of Science Education* 36 (13 2014), pp. 2180–2194. DOI: <https://doi.org/10.1080/09500693.2014.916831>.
- [8] Neil T. Heffernan et al. “The Future of Adaptive Learning: Does the Crowd Hold the Key?” In: *International Journal of Artificial Intelligence in Education* 26 (2016), pp. 615–644. DOI: <https://doi.org/10.1007/s40593-016-0094-z>.
- [9] Juho Kim. “Learnersourcing: Improving Learning with Collective Learner Activity”. PhD thesis. Massachusetts Institute of Technology, 2015. DOI: <https://doi.org/1721.1/101464>.
- [10] Agnes Koschmider and Mario Schaarschmidt. “A Crowdsourcing-Based Learning Approach to Activate Active Learning”. In: *DeLFI & GMW* (2017).
- [11] Chungmin Lee. “Question Generation Workflow: Incorporating Student-generated content and Peer Evaluation”. MA thesis. Massachusetts Institute of Technology, 2020. DOI: <https://doi.org/1721.1/127480>.
- [12] Fergie McDowall and Espen Klem. *stopword*. NodeJS Package Documentation. 2015. URL: <https://www.npmjs.com/package/stopword>.
- [13] Piotr Mitros. “Learnersourcing of Complex Assessments”. In: *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (2015), pp. 317–320. DOI: <https://doi.org/10.1145/2724660.2728683>.
- [14] Casey O’Brien, Max Goldman, and Robert C. Miller. “Java tutor: bootstrapping with python to learn Java”. In: *Proceedings of the first ACM conference on Learn-*

ing @ scale conference (2014), pp. 185–186. DOI: <https://doi.org/10.1145/2556325.2567873>.

- [15] Henry L. Roediger III and Jeffrey D. Karpicke. “Test-Enhanced Learning: Taking Memory Tests Improves Long-Term Retention”. In: *Psychological Science* 17.3 (2006), pp. 249–255. DOI: <https://doi.org/10.1111/j.1467-9280.2006.01693.x>.
- [16] Dominique M.A. Sluijsmans et al. “The Training of Peer Assessment Skills to Promote the Development of Reflection Skills in Teacher Education”. In: *Studies in Educational Evaluation* 29 (2003), pp. 23–42. DOI: [https://doi.org/10.1016/S0191-491X\(03\)90003-4](https://doi.org/10.1016/S0191-491X(03)90003-4).
- [17] Darya Tarasowa et al. “CrowdLearn: Crowd-sourcing the Creation of Highly-structured e-Learning Content”. In: *Proceedings of the 5th International Conference on Computer Supported Education* 1 (2013), pp. 33–42. DOI: <https://doi.org/10.5220/0004384100330042>.