QBank: A Web-Based Dynamic Problem Authoring Tool

Ann Molly Paul

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

Clifford A. Shaffer, Chair
Edward A. Fox
Lenwood S. Heath

June 10, 2013
Blacksburg, Virginia

Keywords: Formal Problem Definition, Problem Authoring Tool, Question Banking,
Computer Education, Parameterized Questions

QBank: A Web-Based Dynamic Problem Authoring Tool

Ann Molly Paul

(ABSTRACT)

Widespread accessibility to the Internet and the proliferation of Web 2.0 technologies has led to the growth of online tools for educational content creation, delivery, and assessment. In existing problem authoring tools, the problem types supported are often limited to static problems. Maintaining high quality of assessment using this medium is made more practical by using tools to author and represent a broad range of assessment problems. A survey of existing problem authoring tools uncovered two main deficiencies: (a) lack of support for authoring "dynamic" (parameterized) problems, and (b) lack of tools that are independent of a specific publishing format, persistence format, and/or authoring platform.

Dynamic problems are assessment problem templates that support parameterization of the problems by the use of variables. Variables dynamically take values assigned at random to generate different instances of a problem from a template. This provides for greater diversity of authored problems, and permits students to practice with different variations of a problem.

A formal definition of an assessment problem structure is presented. This formal definition served as a design aide for a new problem authoring system named QBank, a web-based tool that supports authoring dynamic problems. The proof-of-concept implementation of QBank supports export of questions in CSV format and the Khan Academy Exercise format. The extensible nature of the framework allows future development of features supporting export of authored problems into other publishing and/or persistence formats.

Acknowledgments

I would like to thank several people who supported and encouraged me during my time at Virginia Tech.

First and foremost, I thank Dr. Clifford A. Shaffer, the chair of my committee, for the opportunity to work with him, his guidance, support and infinite patience. I would like to express great appreciation for his painstaking efforts to help me with the writing process, review my drafts thoroughly, identify and correct even minor mistakes. I want to thank Drs. Edward A. Fox and Lenwood S. Heath for serving on my thesis committee, providing advice and suggestions for the improvement of this work. I want to thank Dr. Calvin J. Ribbens and Dr. Doug A. Bowman for making financial arrangements and for being sources of great encouragement.

I would like to thank Eric Fouh for helping me understand the features of the existing software and tools. He was always approachable and helped clarify even the small doubts that I had. My supreme gratitude goes to my brother, Paul Varkey, who taught me various approaches to best practices in coding, programming concepts, helped me solve problems, provided expert guidance in composition and technical writing, encouraged me to think instead of just to learn and, for all the financial support, love and countless hours of time that he spent with me daily over the phone. I would never have been able to attain this without you.

I want to thank my parents for believing in me and financially supporting my dreams to pursue a higher education. I thank my sister, Anisha Paul, for being a close friend, confidant and source of inspiration. I thank all my friends, particularly Rebecca Johns, Zach Layman, Iccha Sethi, Noha El Sherbiny and Matt Morris, for encouraging me and cheering me up during rather stressful periods. Lastly, I also want to thank the Graduate Christian Fellowship and Grace Covenant Church in Blacksburg for loving and caring for me like a little child in their family.

Contents

1 Introduction **1**

2 Literature Review **3**

2.1 What are Authoring Tools? 3

2.2 Standards for Authoring Content (and Problems) 5

2.3 Existing Problem Authoring Tools 6

2.3.1 Programming Problem Authoring Systems 7

2.3.2 Generic Problem Authoring Tools 8

2.3.3 Learning Management Systems (LMS) 12

2.3.4 Summary 13

3 A Formal Definition for Problems **15**

3.1 Introduction 15

3.2 Problem Definition 16

3.3 Types of Problems 18

3.4 Sample Problems 20

4 QBank Requirements Analysis and Design **25**

4.1 Requirement Analysis 25

4.1.1 Support for Dynamic Problems 25

4.1.2 Format-independence and Extensibility 26

4.2 System Design 27

4.2.1 Separate Interfaces for Static and Parameterized Problems 27
4.2.2 Exportable Formats supported by QBank 29

5 QBank Interface **30**
5.1 Software Requirements 30
5.2 Features of the QBank framework 31
5.3 QBank Home Page 31
5.4 Write a Problem 32
5.5 Overview of User Interface Components 33
5.5.1 Problem 34
5.5.2 Variables 35
5.5.3 Variable Reference 35
5.5.4 Problem Template 35
5.5.5 Common Introduction 36
5.6 Problem Authoring 37
5.7 Exporting a Problem 47
5.7.1 CSV Format 47
5.7.2 Khan Academy Exercise Format 49

6 Tool Assessment and Usability Study **52**
6.1 Goals of the Assessment 52
6.2 Methodology 52
6.2.1 Participants 53
6.2.2 Evaluation Criteria 53
6.2.3 Procedure 53
6.3 Observations 53
6.3.1 General Feedback 54
6.3.2 Feedback Based on Evaluation Criteria 54
6.3.3 Results of the Evaluation 55

6.4 Examples of Authored Problems 56
6.5 Summary 62

7 Conclusion and Future Work **63**
7.1 Contributions of this Work 63
7.2 Future Work 63

Bibliography **65**

List of Figures

3.1 Binary Tree Example. 23

3.2 User performing a Postorder Evaluation on a Binary Tree. 24

5.1 QBank – Framework. 30

5.2 Home Screen. 31

5.3 Write a Problem Screen. 32

5.4 QBank Problem Editing Page. 33

5.5 Help Window. 34

5.6 Problem Template Interface Differences for Simple and Multi-part Problems. 36

5.7 Simple Problem Screen. 38

5.8 *Answer Key* Table with all Possible Variable Value Combinations Shown. . . 39

5.9 Variable Values Specified as a Range. 41

5.10 Multi-part Problem User Interface. 42

5.11 Multi-Part Problem Example. 43

5.12 Script Snippet. 44

5.13 Tool Specific Problem – Khan Academy Exercise Question. 45

5.14 Viewing the Exported Khan Academy Exercise Age_Word_Problem_KA in a Web Browser. 46

5.15 Example Problem in CSV Format. 47

5.16 HashMCQ1 in CSV Format Exported from QBank. 48

5.17 HashMCQ1 in CSV Quiz Format for Moodle. 48

5.18 HashMCQ1 in the File Format for the Questionmark Live CSV Import. . . . 48

5.19 Khan Academy Exercise Format of a Static Problem called HashMCQ1. . . . 49

5.20 HashMCQ1 in a Web Browser. 50

5.21 Khan Academy Exercise Format for a Dynamic Problem. 51

6.1 Writing a Static Multiple Choice Problem for Sorting in QBank. 56

6.2 The Exported Khan Academy Exercise for Figure 6.1 57

6.3 Writing a List Type Dynamic Multiple Choice Problem. 58

6.4 The Exported Khan Academy Exercise Showing a List Type Dynamic Multiple Choice Problem. 59

6.5 Writing a Tool-specific Interactive Khan Academy Exercise for Insertion Sort in QBank. 60

6.6 Viewing the Interactive Khan Academy Exercise for Insertion Sort in a Web Browser. 61

Chapter 1

Introduction

Online content delivery has become increasingly common in education. In addition to traditional learning management systems like Blackboard [21] and Moodle [13], recent and prominent examples include Khan Academy [25], Udacity [39] and Coursera [32]. Recent interest in massive open online courses (MOOCs) raises the importance of assessment in online education. Based on the research findings of Black and William [8], formative assessment is seen to facilitate the learning process and positively impacts the learning experience. The literature review by Gikandi et al. [17] emphasizes the effectiveness of online formative assessment in higher education, and how it fosters the development of an effective learning community. In the context of online education, incorporating interactive exercises within the online content and tests with automated assessment can serve the purpose of formative assessment.

The proliferation of Web 2.0 technologies in the field of education has led to the need for developing online resources which include online content and formative assessments. From an assessment resource author's perspective, an assessment is not just made up of questions but is composed of entities that are a combination of a question, its answer, the distractors (or other choices), the mechanism a user interacts with to specify their choice, and a function to compare the answer with the user's choice. This complex entity is called an assessment problem. This project focuses on the development of assessment resources by aiding assessment problem authoring. In this thesis, I will use the simpler term "problem" to mean "assessment problem"(which is explained in detail in Chapter 3).

Tools that facilitate the creation of *problems* are referred to in this thesis as *problem authoring tools* (or, *authoring tools*, in short). In order to obtain the maximum effectiveness of assessment, there needs to be a variety of problems available to students that enable them to practice and solidify concepts. The task of authoring numerous practice problems and test questions can be tedious and time consuming. Most existing authoring tools only support static problem authoring, where the problem author has to create each problem individually. They lack support for dynamic problem authoring, which uses text templates containing

dynamic variables that can take randomly assigned values in a generated problem instance.

Dynamic problems can allow students the opportunity of unlimited practice. The Khan Academy Exercise framework is perhaps the best known example of this capability. But developing problems on the Khan Academy framework is difficult since it effectively requires writing a program for each question. This motivates the need to develop an easy-to-use problem authoring framework that supports authoring of dynamic problems. Creating a large body of practice and test problems for a wide range of content areas will require tools that make it easy to author automatically assessed dynamic exercises.

In this thesis, I do the following:

1. Survey existing problem authoring tools, and, in particular, investigate the implicit and explicit requirements that content and problem authors have, with respect to usability and intuitiveness.

2. Present an abstract formalization for the concept of a *Problem definition* by specifying all the components of a problem and its solution, and describe a broad range of problem types that must be supported by an authoring tool. To my knowledge, there has been no previous attempt to develop a formal model of a problem and its solution.

3. Implement a problem authoring tool that supports both dynamic (i.e., parameterized) problem authoring and interoperability with existing tools.

The resulting tool is named QBank. QBank can be used by a wide range of users to develop good exercises with automatic assessment for online educational content. The tool's interface attempts to simplify problem construction as much as practical by making simple problem structures simple to express, while minimizing the complexity of complex structures where possible.

This document is organized as follows. In Chapter 2, I discuss the relevant literature and tools in the field of authoring systems. Chapter 3 defines a problem and formalizes the components that constitute it. Chapter 4 describes the requirement analysis and high level system design decisions for QBank. The details of the features and functionality of QBank are explained in Chapter 5. An assessment and usability study for QBank is discussed in Chapter 6. Finally, Chapter 7 summarizes the contributions of QBank and gives recommendations for future work.

Chapter 2

Literature Review

In this chapter I systematically explain and classify what problem authoring tools are, present some required standards that they adhere to, and describe many existing tools.

2.1 What are Authoring Tools?

The W3C initiative [2] defines authoring tools as *any web-based or non-web-based application(s) that can be used by authors (alone or collaboratively) to create or modify web content for use by other people (other authors or end users).* There are several types of authoring tools. They serve specific or multiple purposes, and can be categorized by their special features. Wilde [42] classifies such tools based on their core functionality.

Within the context of problem authoring and publishing tools, I classify tools as follows:

1. Problem Authoring Tools:

   These tools provide an interface solely for the purpose of authoring problems. They are not intended for broader usage in creating or managing learning content and coursework. They often are used to create and store authored problems in an online repository. A representative example is Quadbase [28].

2. Problem Publishing Tools:

   These tools are designed to present/publish authored problems, and can sometimes be used to maintain records of a learner's progress. Learners can use these tools to practice and assess their skills in the problems that are presented to them. These tools do not provide support for problem authoring. For example, Khan Academy [25] allows for exercise questions written in the Khan Academy Exercise framework to be displayed in a web browser to users for assessment and practice.

3. Problem Authoring and Publishing Systems:

   These systems are designed for authoring problems and creating test suites while also serving as publishing platforms that support logging of results, reports, etc. Learning Management Systems are well known examples of systems that fall under this category. They are also used for structuring online educational content within specific courses, lessons, and modules. Examples include Moodle [13] and Blackboard [21].

The user interfaces for problem authoring tools can be broadly classified as follows:

1. Text-only editors:

   These include authoring tools where the user interacts with text editors to manipulate content only. The presentation markup, layout, and organization of the problem is already preprogrammed within a template by the tool. The user cannot modify the presentation of the problem.

   These require the least coding expertise but do not provide any flexibility. Static text questions are usually authored in such an interface. Most learning management systems have text editors in their question creation interface where text can be added to create questions. Other examples are Articulate [6], online quiz makers, etc.

2. Code-based editors:

   These can be used by technically savvy programmers, giving them complete flexibility to code the layout, markup and presentation of the content. The problem author can write scripts for creating interactive exercises that allow users to better visualize and understand certain problems. Using appropriate mark up language, they can also specify how the problems will be presented to the user. Strong programming skills are required in order to author problems in these editors. Problets [27] is a good example, as is the Khan Academy Exercise Framework [25].

3. Hybrid editors:

   These combine features of a text editor and a pure code-based editor. While giving the authors flexibility to manipulate the underlying markup, authors also have the choice of making use of a word processor style editor. This accommodates the needs of novice users as well as authors with higher levels of technical expertise. Quadbase [28] is an example since it allows authors to specify some scripts, variables, etc., or they can simply fill in static text without any programmatic specifications.

2.2 Standards for Authoring Content (and Problems)

1. **Sharable Content Reference Model**

   Sharable Content Reference Model (SCORM) [23] is a collection of industry standards and specifications for e-learning systems. SCORM emphasizes *Reusability*, *Durability*, *Interoperability* and *Accessibility*. An advantage of learning modules that conform to the SCORM standard is that they are compatible with any learning management system that supports SCORM-compliant content.

   Here are ways that SCORM compliant content can be authored [3]:

   (a) For non-technical users, SCORM authoring software tools can be used. A Dreamweaver extension, SCORM runtime wrapper, is extensively used to convert single or multiple HTML pages to Sharable Content Objects (SCOs).

   (b) Technically skilled users can author content using HTML, JavaScript and SWF files. The author writes SCORM Javascript calls to build SCOs and track user information. The Advanced Distributed Learning (ADL) initiative provides templates with the necessary guidelines and structure for this purpose.

2. **Question and Test Interoperability specification**

   The Instructional Management System (IMS) [4] publishes standards that are commonly used as learning technology standards. These standards enhance the effectiveness of the wide variety of technologies that promote the learning experience. IMS Question and Test Interoperability specification (QTI) [19] is useful for problem authoring in a format not specific to any proprietary format. It defines a standard format for the following:

   (a) Representation of assessment content and results.

   (b) Exchange of this material between authoring and publishing systems with compatibility to Learning Management Systems.

   XML data binding used together with a data model that defines questions, results and their assessments comprises the QTI format.

   The terminology developed by IMS QTI to structure assessment data is as follows.

   (a) Tests are called *assessments*.

   (b) Assessments contain *sections*, which are groups of questions referred to as *items*.

   (c) The person taking the test is called a *participant*.

   (d) The question types (which include multiple choice, true/false, drag drop, etc.) are called *item types*.

2.3 Existing Problem Authoring Tools

There has been much recent and growing interest in eTextbooks, MOOCs, online courseware, and other online educational systems. A major weakness in many of these efforts is the lack of automated assessment. Often in such systems, assessment is little more than multiple choice question banks. While much useful assessment can be done with multiple choice questions, a richer collection of question types can enhance the assessment process. A full, rich collection of automatically assessed exercises of appropriate types would be a major asset in a totally online course such as a MOOC.

Progress is being made in enriching the types of questions that can be automatically generated, presented, and assessed. Khan Academy [25] includes a rich system for generating a wide range of questions [7]. Within Computer Science education, some outstanding examples include TRAKLA [29], JhavePOP [16], CodingBat [33], and Problets [27]. OpenDSA [36, 35, 10] attempts to unite all of these problem types within an online collection of tutorials and algorithm visualizations. Given this progress in exercise types that can be automatically assessed on the one hand, and the growing need for automated assessment on the other, there is value in systems that allow for easy development, collection, and deployment of rich question types. Question banks have been around for a long time in the form of content management systems such as Moodle [13]. There are standard formats for defining certain problem types such as SCORM [23] and QTI [19].

But with only a few exceptions like the Khan Academy Exercise infrastructure, these systems have only defined or stored static problems. A static problem is one that has a fixed problem statement and fixed answer choices. Some question systems allow some flexibility by allowing random selection from a set of distractors in multiple choice questions. For example, the system might give the answer and three out of five distractors selected at random, with all of the choices displayed in a randomized order. But that generally is the extent of dynamic capability that they provide. The Khan Academy Exercise Framework [7] goes well beyond this. It is an extremely flexible mechanism for specifying problems. It can define problems that, for example, will generate an equation where the values are generated at random from a specified range, and the answer is the desired computation (which the system can itself compute). In fact nearly any problem for which the various problem components defined in Chapter 3 can be computed, can be represented by the Khan Academy exercise framework.

To better understand the state of the art in problem authoring frameworks, a thorough study of the existing frameworks, models, and software tools is presented below. Maryami et al. [30] reviewed various authoring tools by classifying them based on their functionality and intended purpose.

In order of increasing generality, there are typically three kinds of problem authoring and publishing tools:

1. Programming Problem Authoring Systems: These are created to support learning and

practicing different concepts of programming languages. They are exclusively used for authoring and assessing coding problems.

2. Generic Problem Authoring Tools: These are intended mainly for problem creation and publishing.

3. Learning Management Systems (LMS): These are full-fledged, generic platforms that help create and manage content, courses, and users. They support authoring as well as publishing of content. Aggregating course related information and maintaining it consistently is a key purpose for these tools.

2.3.1 Programming Problem Authoring Systems

Two tools, with a rich set of features that include visual interaction and dynamic (i.e., templated) problem generation, include TRAKLA2 and Problets. These systems support only code-related problems.

TRAKLA2

TRAKLA2 [26] is a framework for automatically assessing visual (i.e., interactive) algorithm simulation exercises. This was developed to teach the basic concepts of data structures and algorithms to students. The learner interacts with the user interface provided and simulates the execution flow of an algorithm and changes of state, if any, in the data structure of the displayed problem. The exercises are graded based on comparison of the logged sequences made by the learner to the correct sequence(s). The model answer visualization is helpful for allowing a student to understand the concept if they do not know the solution. The authors claim that a skilled Java programmer can author an exercise in 2-3 days using this tool [29].

Problets

Problets [27] is a problem-solving platform for learning and assessment of programming concepts in C++, Java, C# and Visual Basic. The type of problems include program evaluation for output and code debugging.

Features:

1. Error detection in answer attempts.

2. Textual and/or visual feedback.

3. Progress is displayed on a bar graph superimposed on the learning objective of the exercise.
4. Playback controls allow a learner to interact by progressing step-by-step through each line of code.

Authors can use the 200 existing problem templates or define a custom problem template to include in Problets. The problem template must be written in a BNF-like notation. In order to author questions, the values can be changed in the problem template, which generates a unique problem instance.

2.3.2 Generic Problem Authoring Tools

eLearning XHTML editor

To assist web content authoring, the eLearning XHTML editor (eXe) [14] project developed an open source authoring application to help teachers and academics with publishing of web content *without* the need for proficiency in HTML or XML markup. It is free and can be downloaded to computers. The instructional content authored is standards compliant.

Features:

- Provides an intuitive WYSIWYG environment where authors can see what their content will look like in a browser when published.
- Resources authored in eXe can be exported in IMS packages, SCORM compliant packages, or as web pages.
- Supports three types of questions - Multiple Choice, Multi-select (Multiple response), and true/false questions.

Quadbase

Quadbase (Question and Answer Database) [28] is an online tool for authoring and storing exercise questions that can be used in developing teaching resources. The main purpose of Quadbase is for educators to author questions or use existing questions from the repository by embedding the questions directly in their learning resources. This system is not meant for learners to work through questions and obtain grades/reports. All questions authored are licensed under the Creative Commons Attribution License, which allows for legal reuse of authored questions.

Features:

- Support for static multiple choice questions and grouping of questions in the multi-part question type.
- Versioned access to questions via permalinks.
- Reuse of exercises stored in the repository is done by using the edit option provided by the tool.

Limitations:

The support for dynamic questions is still not fully functional and so is the support for the matching question type.

Course Lab

Course Lab [41] offers a WYSIWYG environment for creating high-quality interactive online content. This tool does not require the author to have any programming knowledge to use. This tool is similar to Adobe Captivate (see below) in terms of features, but it is free to use.

Features:

- WYSIWYG environment–no HTML or other programming skills required.
- It supports questions like single select, multiple select, fill in the blanks, ordered items and three types of matching questions: matching pairs, matching groups and multiple matching groups.
- It also supports import of questions by using external QTI 1.2 files.

Khan Academy Exercise framework

The Khan Academy Exercise framework is an open source framework for building academic exercises [25]. Since the questions authored in this framework rely heavily on JavaScript and framework-specific mark up, the questions created can be embedded with online content to serve the purpose of automated and continuous assessment. The type and complexity of questions that can be authored is dependent on the programming skills of the author. It provides the essential markup and support for generating dynamic exercises by the use of variables. This makes it powerful from an author's perspective, since it can generate numerous questions that can be used for making effective learning resources. This ensures that students cannot identify patterns in the order of questions or memorize the answers to different questions. This dynamic capability of the Khan Academy Exercise framework makes it robust and facilitates freshness of content.

Features:

- Supports writing of automatically assessed exercises.
- Excellent framework for authoring interactive exercises on graphs, geometry and other complex math and computer science related problems.

Limitations:

- Authoring questions require extensive knowledge of HTML, JavaScript and Khan Academy's exercise API.
- The coding conventions are rigid and not consistent. The lack/excess of specified delimiters fails to render an authored exercise in the browser.

Articulate

Articulate [6] provides three different products to the e-learning community: Articulate Storyline, Articulate Studio and Articulate Online. Articulate Storyline and Studio are tools for content authoring and quiz creation, while Articulate Online solely publishes authored content and keeps track of viewers' activities. The main differences between Storyline and Studio reflect the needs that they are designed to meet. Articulate Storyline is a single standalone product that allows you to create interactive content, simulation, and any type of quiz. On the other hand, Articulate Studio is a suite of three different products - QuizMaker '09, Presenter '09, and Engage '09 - that allows you to author in the PowerPoint environment using the individual products. Articulate Quizmaker is designed for creating tests and questionnaires.

Features:

- User friendly and intuitive interface to generate quizzes and surveys that support 21 types of static questions.
- Supports export to Web, SCORM-compliant LMS, and Articulate Online.

Limitations:

- Single image per question, which restricts the possibility of choosing from multiple images as answer selections.
- One blank per question in fill-in-the-blank questions.

Adobe Captivate

Adobe Captivate [38] allows for creating E-learning content with simulations, quizzes, and other engaging experiences without any programming experience. Questions are authored using question slides which are templates for different question types that the author can use to modify the placeholders and generate static questions.

Features:

- No programming or multimedia skills required.
- Remediation is a key feature of this tool, which allows the learners to revisit the relevant section on answering a question incorrectly.
- In addition to common problem types like multiple choice, fill in the blanks, matching and true/false, Captivate offers two additional unique question types called Likert and Hot Spot. Likert is an ungraded question type where a user can specify their level of agreement with the statement. Hotspot is a stimulus-based interactive question where interaction of a pointer over an uploaded image on the question slide can indicate the user's selection.

  Limitation: Question slides do not support dynamic questions. Those slides provide a consistent structure to the question.

Questionmark

Questionmark [37] provides comprehensive question authoring, publishing and report analysis in two different products: Questionmark OnDemand and Questionmark Perception. Questionmark OnDemand is a *software as a service* (SaaS) solution that allows authors to use the features offered without installation, upgrades, and maintenance expenses. Questionmark perception is a standalone, installed application. It allows users to create questions and organize them into quizzes and tests for online delivery.

Features:

- Provides standard assessment screens, controls, and login in 20 different languages.
- Questionmark OnDemand is a scalable platform capable of load sharing with a promised 99.99+% uptime for large volume delivery needs.
- Maintains revision history with support for sharing questions with other authors for review and editing.
- Supports Likert and Hotspot type of question in addition to the commonly used question types, similar to Adobe Captivate.

2.3.3 Learning Management Systems (LMS)

From the statistics on LMSs presented in [1], I have summarized below a few of the widely used open source and proprietary LMSs.

Moodle

Moodle [13] is one of the most widely used and adopted LMSs by various educational institutions. Being free, open source software, it is inexpensive to incorporate into the learning system. It is also competitive with proprietary tools. This software is continually evolving with an active developer community. The ability to add plug-ins and tailor the infrastructure to a specified audience is a feature specific to Moodle.

Features:

- Questions can contain images, video or other media files.
- It can import course content that is SCORM 1.2 [23] compliant, and can export quiz content in QTI format [19].
- It supports migration from the BlackBoard system.

Moodle also supports importing questions in CSV format [12]. The file format compatible with Moodle is:

"Category";"Question text";"CA 1";"CA 2";...;"CA n";"";"WA 1";"WA 2";...;"WA m",

where *CA* stands for correct answer and *WA* stands for wrong answer. Empty cells or values separate the correct answers from the wrong ones.

Sakai

The Sakai Project [15] is a coordinated higher education open source community project launched in 2003.

Features:

- It can import and export course content using the IMS Content Packaging standard.
- Instructors can create automatically scored true/false, multiple-choice, multiple answer, matching, and fill-in-the-blank questions. Short answer/essay, audio recording answer, and file upload questions are not automatically scored. Assessment settings can be modified in order to specify the grading rubric. Questions can contain images and audio files.

Edmodo

Edmodo [31] is a social learning platform that is essentially a social network in the domain of online education for teachers, students and parents to interact. It is free and has an easy-to-use interface similar to Facebook. Edmodo gives the option to create quizzes and polls. It also supports upload of Microsoft Word, Excel or PowerPoint files.

Features:

- No download required. It can be accessed anywhere you have Internet access and is also supported on mobile devices through apps on Android and iOS.
- Quizzes can be graded or practice quizzes. Graded quizzes can be used to keep track of the progress of a learner whereas practice quizzes emphasize learning concepts without assigning any grade to the learner. Edmodo supports true/false and multiple choice questions.

Blackboard

Blackboard [21] is a course management system that maintains student records and authentication information. It is web-based, which makes it universally accessible. Blackboard is not free of charge. It can be installed on local servers or hosted by Blackboard Managed Hosting [22].

Blackboard Learn [20] is a foundation for a rich education experience by delivering technology and services. Tests, quizzes, surveys and polls are the different types of assessment available.

Features:

- Support for quiz creation, automated grading and generated progress reports.
- Supports multiple question types - either/or, jumbled sentence, ordering, opinion scale, quiz bowl, short answer, etc.

2.3.4 Summary

This chapter captures the relevant literature related to this project and motivates the necessity of defining the notion of a problem formally in order to effectively design a standard user interface for different types of problems. Most of the existing tools supports static question authoring but do not handle dynamic question authoring. Some of the key features that I consider desirable when developing a problem authoring tool are as follows:

1. Easy and intuitive to use for authors with any level of expertise (novice to highly skilled).
2. Functionality to support dynamic problem authoring.
3. Automated assessment for authored problems or support for converting problems into question formats with automated assessment.
4. Option for developers to further modify the tool in order to tailor it to specific problem publishing formats if needed.

Chapter 3

A Formal Definition for Problems

3.1 Introduction

Our main goal is to design a user-friendly tool for authoring a wide range of problems. While the authoring tools surveyed in Chapter 2 each support various problem types, none of them combine support for a wide variety of dynamic problems with ease of use. If we want to develop a tool that can handle a wide range of problem types, it will help to have a firm understanding of what are the fundamental components of a problem. In particular, we want to understand the core abstract components of a problem specification (where a complete problem specification defines both a problem and its corresponding solution) in its most general form, and to capture that understanding in precise definitions. Specific problem types can then be expressed as sub-types of this general definition of a problem. This formalized understanding provides authors with a consistent structure when undertaking the task of authoring problems.

Formulating the definition of a generic problem will guide the design for a system that lets users provide problem specifications that are general and complete in terms of allowing for full parameterization of the problem statement and specification of a parameterized solution function that generates the specific solution for any specific problem instance. However, increased generality is sometimes attainable only by specification of function definitions via programming constructs. But since many problem types greatly restrict or eliminate specific components of a problem specification, we can hope to fashion convenient user interfaces to support specific problem types more quickly once we have a better understanding of the general form.

Another advantage to defining problems in a generalized form is that, given a problem specification, it can then be translated into an existing problem representation if the problem fits within the target system's constraints. For example, we can realize our goal of a nicer user interface to specify questions that are mapped into the Khan Academy question system for

implementation. Or, more restricted types of problems can be mapped into the Quadbase [28] or IMS question banking systems. For systems like Quadbase that do not support parameterization, a concrete problem instance (where parameters have been bound to values) could still be output.

When trying to formally define the components of a problem, one typically encounters two difficulties. The first arises from focusing too much on a particular problem type, in which case it can be hard to recognize the features of problems in general. This is because many problem types do not obviously appear to have all of the necessary components. For example, when thinking about problems in an automated assessment context, many people immediately think of multiple choice questions. In that case, one is unlikely to consider the fact that some problem types require a more complex user interface or a mechanism for computing the correct answer. The other difficulty is that the stark differences in problem types can hide their relationships. For example, a matching problem (where there are two columns of items, and each item in the first column should be matched to an item in the second column) seems to have little in common with a multiple choice question.

We must also account for concepts of dynamic problem generation. That is, we would like to be able to account for systems that create problems such as the addition of two numbers, where each number is drawn from a specified distribution within a specified range. Clearly this immediately leads to the complication that the answer can be computed only after the variables have been bound to particular values. While not many problem specification systems today can support dynamic problem generation (most only support static problems where all problem text is specified in advance), any system for creating rich problem sets must support dynamic generation.

We approach the issue in two stages. First, we define at a high level a set of components that collectively characterize problems of all types. Given the wide range of potential problem types, these components must necessarily be fairly abstract. We then show how various problem types map into the formalism. Most problem types will simplify one or more of the generalized components.

3.2 Problem Definition

In this section we focus on the main issue to be discussed: What comprises a problem?

Problem:

A **Problem** consists of the following components:

- **Problem Statement**: A 2-tuple ⟨**Problem Template**, **Problem Instance Generator**⟩ where the **Problem Instance Generator** is a function that generates a

Problem Instance.

- **User Interface**: A mechanism that a user interacts with to create a **Student Answer**.
- **Model Answer Generator**: A function that takes a **Problem Instance** and generates a **Model Answer**.
- **Answer Evaluator**: A function that compares the **Student Answer** to the **Model Answer** to determine whether the **Student Answer** is correct or not.
- **Variables**: These carry information from the **Problem Statement** to the **Model Answer Generator**.

Problem Instance:

The **Problem Instance** is a static statement of a problem in a form that can be presented to the user. It is generated from the **Problem Statement** by binding any **Variables** to specific values. If there are no **Variables**, then the **Problem Instance** is merely a copy of the static **Problem Statement**.

Problem Statement:

The **Problem Statement** is defined as a 2-tuple: ⟨**Problem Template**, **Problem Instance Generator**⟩.

The **Problem Template** consists of text and, optionally, **Variables**. A simple example of a **Variable** might be a uniformly distributed random variable within a specified range, such as for an addition problem.

The **Problem Instance Generator** is a function that takes the **Problem Template** as an input and returns a **Problem Instance**. In richer systems, a **Problem Template** includes one or more **Variables**. This function binds the **Variables** to specific values to generate the final text for the **Problem Instance**, which is then presented to the user. For any problem without **Variables**, the **Problem Instance Generator** is an identity function that merely returns the **Problem Template** as the **Problem Instance**.

Student Answer:

The **Student Answer** is constructed by a user through manipulating the **User Interface**.

User Interface:

The **User Interface** is the mechanism within which the user operates to specify the **Student Answer**. Often, the **User Interface** is nothing more than a pull-down menu, radio buttons,

or other selection mechanism between a fixed set of choices, or a simple textbox in which to type a number or string. In other cases, it might be something where the user must manipulate the objects in the interface to construct the **Student Answer**, such as clicking on some values within an array to cause a series of swap operations to simulate the operation of a sorting algorithm.

Model Answer:

The **Model Answer** is generated by the **Model Answer Generator** from the current **Problem Instance**.

Model Answer Generator:

The **Model Answer Generator** is a function of the current **Problem Instance** (which was generated from the **Problem Statement** with a specified set of values for the **Variables**). Another way to characterize the **Model Answer Generator** is that it is a function of the **Variables** as bound for some **Problem Instance**. The output is a **Model Answer** for the current **Problem Instance**.

Answer Evaluator:

The **Answer Evaluator** is a function which takes the **Model Answer** and **Student Answer** as inputs and determines if the **Student Answer** matches the **Model Answer**. The output might be YES vs. NO, or a score. Often, this function is a simple comparison (such as whether the student selected the right choice from a set of options, or typed the right number). If the **User Interface** required arranging the values in an array, then the **Answer Evaluator** might verify whether the values in the array for the **Student Answer** are in the same order as the values for the **Model Answer**. If the **Problem** requires a short sentence or essay as the answer, then there might be significant natural language processing required.

Variable:

The **Problem Template** can include one or more **Variables** that take on varying values. Creating the **Problem Instance** involves binding a value to each **Variable**. The **Model Answer Generator** must be able to handle any possible combination of variable values that might appear in the **Problem Instance**, so that the **Model Answer** corresponding to the **Problem Instance** will be correct.

3.3 Types of Problems

We find it useful to categorize problems into different types. Our focus here is on characteristics of problems that relate to how a system to specify such problems would be implemented.

The problem types are expressed based on components from the formalized problem definition in Section 3.2. Section 3.4 will show examples for the problem types.

In the following descriptions, $\mathcal{R}(x)$ refers to the range of values that a variable x can take.

1. **True/False problem**:
   (a) **Model Answer** $\in \{True, False\}$
   (b) **User Interface**: a mechanism to select between 'True' and 'False'.
   (c) Many systems support static T/F problems. In such cases, the **Model Answer** is a literal provided by the developer of the question and the **Problem Instance** is merely a literal copy of the **Problem Template**. But a richer system could include **Variables** within the **Problem Template**. This will require a function to transform the parameterized **Problem Template** into a **Problem Instance** when needed. Depending on how the problem is written, this might or might not require a function to generate the corresponding **Model Answer**.

2. **Multiple Choice problem**:
   (a) The **Problem Template** includes two special variables designated n and *choices*, where n represents the number of choices and *choices* represents the particular set of choices for that **Problem Instance**. We define *choices* to be the set of all possible choices, i.e., $\{choices \mid choices \in \mathcal{R}(choices)\}$.
   (b) **Model Answer** $\in$ *choices*, is the correct choice for the given **Problem Instance**.
   (c) **User Interface**: a mechanism to select a single correct answer between all the displayed choices. A **Problem Instance** might have fewer than all possible choices. It might or might not require including the correct answer. In such cases, 'None of the Above' must be a choice.

3. **Multiple Response problem**:
   (a) **Problem Template** is defined the same as for **Multiple Choice problem** above.
   (b) **Model Answer** is a subset of *choices*, consisting of all (correct/applicable) choices for the given **Problem Instance**.
   (c) **User Interface**: a mechanism to select multiple options from all the displayed choices.

4. **Multi Part problem**:

(a) The **Problem Template** consists of mandatory introductory text, a variable p that represents the number of parts, and a variable *type*, that is an ordered sequence of p **Problem** types, to represent the problem type of each part, in order.

(b) The **Problem Instance** consists of a concrete realization (instantiation) of all p **Problem** types for this **Problem Template**.

(c) The **Model Answer Generator** consists of a sequence of p correct answers for each of the parts, in the order they appear in the **Problem Template**.

(d) **User Interface**: a sequential or synchronous mechanism that shows all the **User Interfaces** of all the parts, in any order.

5. **Summative problem**:

   (a) The **Problem Template** consists of an **Introduction Text**, a list of previously authored **Problems** and a variable *weight* that specifies the probability for drawing the next **Problem**. This concept is supported by various problem systems that "relate" multiple **Problems** with a shared **Introduction Text**.

   (b) The **Introduction Text** is a static string used to specify a problem overview. This is optional.

   (c) The **Problem Instance** is a concrete instantiation of the selected **Problems**.

   This allows for grouping already existing **Problems** on the same topic into one cluster. This serves two purposes.

   (a) Reusing previously authored problems.

   (b) Testing students using different **Problems** on the same topic beyond simply varying the values of **Variables** in the **Problem Statement**.

3.4 Sample Problems

In the following examples, $\mathcal{I}(x, y)$ refers to the *identity* function that compares variable x with variable y and returns *true* if they are the same, and *false* otherwise.

A variable is denoted by using the symbol $ to precede a string.

1. **True/False problem**

(a) Example 1:

Introduction Text: A greedy algorithm is an algorithm that makes the locally optimal choice with a hope of finding a global optimum for a particular situation.

Problem Instance: Topological Sort is a greedy algorithm.

User Interface:
Choose one of the following:

- True
- False

Model Answer: True

Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

(b) Example 2:

Introduction Text: Data Structures

Problem Template:

i. Text: The relation between the push and pop operations is such that the $a is a $b data structure.

ii. n: 2

iii. Choices
   - True
   - False

Variables:
$a =“Stack”, “Queue”
$b =“LIFO”, “FIFO”

Problem Instance: The relation between the push and pop operations is such that the Stack is a LIFO data structure.

User Interface:

- True
- False

Model Answer: True

Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

2. **Multiple Choice problem**

(a) Example 1:

Introduction Text: Simplify the following:

Problem Template:

i. Text: log 8 + log 3 - log 6

ii. n: 6

iii. Choices

- log 5
- log 4
- log 18
- log 12
- log 3
- log 10

Problem Instance: log 8 + log 3 - log 6

User Interface:

- log5
- log4
- log18
- None of the above

Model Answer: log4

Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

(b) Example 2:

Introduction Text: Addition

Problem Template:

i. Text: Find the sum of $a +$b.

ii. n: 4

Variables:

$a = range(3, 8) // takes a value between 3 and 8

$b = list(200, 1, 7) // takes any of the three values – 200, 1 or 7

Problem Instance: Find the sum of 5 + 200.

User Interface:

- 208
- 205
- 206
- 210

Model Answer: 205

Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

3. **Multiple Response problem**

   Introduction Text: A greedy algorithm is an algorithm that makes the locally optimal choice with a hope of finding a global optimum for a particular situation.

   Problem Instance: Which of the following have greedy algorithms?

   User Interface:

   (a) Dijkstra's Shortest Path algorithm.
   (b) Bubble sort algorithm
   (c) Kruskal's algorithm
   (d) Prims algorithm
   (e) Matrix chain multiplication

   Model Answer: a,c,d

   Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

4. **Multi Part problem**

   Introduction Text: Given the tree structure below.

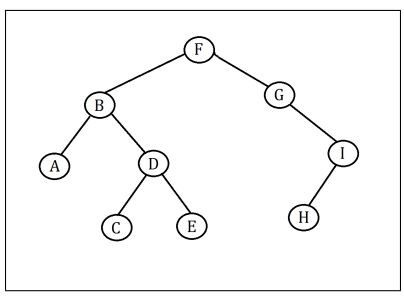


Figure 3.1: Binary Tree Example.

- **Problem Instance**: Show a preorder evaluation for the tree.
  User Interface:
  (a) F, B, A, D, C, E, G, I, H
  (b) A, B, C, D, E, F, G, H, I
  (c) A, C, E, D, B, H, I, G, F
  Model Answer: F, B, A, D, C, E, G, I, H
  Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)
- **Problem Instance**: Show a postorder evaluation for the tree.
  User Interface:

(a) F, B, A, D, C, E, G, I, H
(b) A, C, E, D, B, H, I, G, F
(c) A, B, C, D, E, F, G, H, I

Model Answer: A, C, E, D, B, H, I, G, F
Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

- **Problem Instance**: Click on the nodes in the correct order to show a postorder evaluation in the given binary tree.
  User Interface:

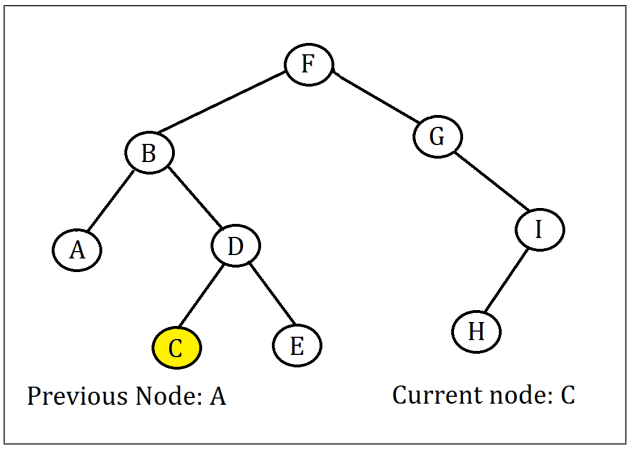


Figure 3.2: User performing a Postorder Evaluation on a Binary Tree.

Model Answer: A, C, E, D, B, H, I, G, F
Answer Evaluator: $\mathcal{I}$(**Model Answer**, **Student Answer**)

5. **Summative problem**

   List of **Problems**:

   (a) **Problem** 1 [$weight = 1$]
   (b) **Problem** 2 [$weight = 1$]
   (c) **Problem** 3 [$weight = 2$]

   For instance, a new **Problem** 5 can be created by selecting **Problem** 1, **Problem** 2, and **Problem** 3. The user will be dynamically provided with a **Problem Instance** of either **Problem** 1, **Problem** 2 or **Problem** 3, selected at random with **Problem** 3 having *twice* the probability of being posed as a question. If generated in series, this will allow the user to be tested on a series of different questions, not just variations of a single question with different values to the variables.

Chapter 4

QBank Requirements Analysis and Design

From the literature survey in Chapter 2 and our analysis of the key components of a problem in Chapter 3, we understand that the support of *dynamic problem* authoring is key in developing a powerful problem authoring tool.

Additionally, it would be desirable for a problem authoring tool to be generic, extensible and not tied to any particular publishing platform format or persistence format. In other words, it should have the capability of exporting problems in any desired format. This would promote seamless interoperability between authoring tools, allowing problem authors to focus on creating and assessing content rather than gaining competence in a particular tool (or, tools) – which is time-intensive and non-transferable.

4.1 Requirement Analysis

In the following two sections, we analyze these requirements and explain how they influenced the design decisions for QBank.

4.1.1 Support for Dynamic Problems

Evaluation and assessment is known to be an integral part of the learning process [9]. Practicing numerous problems solidifies newly learned concepts. This calls for the authoring of numerous creative and challenging questions. The time it takes to write an effective question makes it difficult for authors to produce different problem sets for learners. Another challenge for authors is ensuring the accuracy of questions. This requires a finely tuned form of critical thinking in order to account for all possible outcomes. Questions also have to be

in accordance with the level of knowledge of the student. An insufficient pool of questions is a drawback for learners who seek to assess their knowledge and need direction for further progress. Also, repetition of the same questions leads to practices that do not help them to learn, like copying answers from other students.

A *dynamic problem* can be expressed using a problem template with variables. The problem template is identical across problems but the variables change for every problem instance. The values the variable can take needs to be specified by the author.

In essence, one well written question can take several variable values that generate numerous question instances. Additionally, in most cases, a correct answer to a parameterized question can be calculated by a formula or executed by a standard language compiler without the need for a teacher or author to provide it. In Section 2.3.1, I discussed two examples of systems that support this form of assessment where the answer is dependent on the variables in the problem instance generated.

A set of three studies explored the effectiveness of continuous assessment for online learning by the use of quizzes, visualizations, etc. Multiple choice quizzing was seen to produce positive effects in two out of the three studies since the choices give the necessary direction for users to identify incorrect answers [11, 18]. The variation in the questions from student to student ensures that the students work on solving problems independently and are not posed the same question again. For instructors, writing one problem template and specifying different values for the variables significantly simplifies the task of problem authoring. A templated problem is more complex to write, but provides many questions as a result.

The results of these studies serve as the inspiration for affording a special treatment to *dynamic problems* within the framework. This ensures the diversity of problem instances based on the parameters specified in the problem template.

4.1.2 Format-independence and Extensibility

To enable reuse of authored questions in other publishing platforms, there is a need to ensure that problems are not limited to being authored in any particular persistence/ publishing format. Problems need to be represented by the software in a generalized format independent way which allows the software to identify the components supported by a target output format. The reason that Khan Academy Exercise format was selected to demonstrate proof of concept is because it represents the most flexible output question representation that we are aware of and it is also being used by the OpenDSA project that QBank supports.

4.2 System Design

QBank aims to provide a system that is able to accommodate the different technical skills of an author as well as equipping the author with a standard interface that can be used easily to leverage the power of the tool. A user first identifies the type of problem to be authored, and then uses an appropriate interface for the problem types. This motivates the need to formalize the definition of a problem and specify what the different commonly used problem types are, which was done in detail in Chapter 3. This formalized definition serves as a design aid while developing the user interfaces for different types of problems supported by QBank.

4.2.1 Separate Interfaces for Static and Parameterized Problems

QBank tool provides several distinct interfaces. For example, it provides separate interfaces for authoring static multiple choice questions vs. dynamic questions. One can argue that a single interface that supports both static and dynamic questions would be ideal. But differences in the way a question is authored in these two cases have resulted in this design decision. Some objections that may be raised to this design decision are listed and addressed below.

1. **Additional support for dynamic features can be incorporated as hidden components.**

   The similarities between the two interfaces are limited to the problem template. The model answer seems similar but varies in functionality depending on the type (static or dynamic) of the problem.

   For a simple static problem, static choices are explicitly written by a problem author (unlike dynamic problems). Depending on the type of the variable, the user interface varies. In the list type dynamic problem, the values of the solution corresponding to the different variable combinations can be included as choices (distractors), since the answers for list variable problems are closely related in terms of context. There needs to be additional interface features for authoring of this type of problem, which is shown in Figure 5.8.

   Since range type problems are intended for ranges of integers, an interface that explicitly lists all possible values is impractical. Instead, range variables are typically used in a calculation to generate the model answer. This is shown in Figure 5.9.

2. **Is additional support for dynamic problems really required?**

   QBank is a problem authoring tool that tries to make problem writing as straightforward as possible. In order to do so, I studied the differences between dynamic and

static problem types. To make QBank a powerful yet intuitive authoring tool, additional support has to be added to dynamic problem types for defining the values that variables take.

For instance, if the variables take a list of static values, then the answer could be specified as a list of values based on the possible combinations of values a problem instance can have. In order to make this task easy and ensure that an author does not incorrectly order the answers, a button generates all possible combinations for variables in the correct randomized order that they will appear. The author only needs to provide a list of answers in that given order that corresponds to the problem instance with a specified set of values.

This added feature makes QBank more user friendly by simplifying the task of authoring dynamic problems.

3. **The interface can be the same for different types of values a variable can take.**

   In the most simplified case of a dynamic problem, the value that a variable takes can be of two types. They can either be instantiated using strings separated with commas or using a function that specifies a range of integers which at run-time gets randomly assigned a number within that range.

   In the case where the value of a variable can be specified as a range of integers, it would be tedious for an author to specify the solution exhaustively for each value that a variable can possibly take, since the range of values can be as extensive as 1 to 1000 or -10,000 to 54 or as simple as 0 to 4. On the other hand, in the case where a variable is specified by a list of comma-separated strings, the author should be able to specify the solution for all the possible combinations of variable values. This is essential in the case of non-numeric strings.

   A button to generate an answer key to help order the list of answers correctly is essential for a list type of problem but is impractical for a range problem. For a range problem, the solution can be the calculation performed in the problem template with variables. When the problem instance is generated, the model answer generator uses the values assigned to the variables to generate the model answer (or solution).

4. **A single UI entry-point would suffice.**

   The differences between the problems make these interfaces so disjoint that the only way to integrate them would be to ask the user to select if the problem was going to be a static or dynamic problem at the start. By separating the interfaces, the author need not go one level deeper to make that choice. This also ensures that the author is not confused by details and features not needed for a static problem.

4.2.2 Exportable Formats supported by QBank

Format independence and support for outputting authored problems to different formats is a key feature of QBank. This is shown by support for (a) the generic CSV format and (b) the Khan Academy Exercise format.

1. **CSV**

   Authored problems are stored in the database and can be exported to a CSV format. This enables the sharing and translation of problems to other formats.

2. **Khan Academy Exercise Format**

   QBank can export problems to a compatible HTML format with appropriate Khan Academy Exercise markup. That allows the problem to be displayed on any web browser. This format was chosen to serve as a proof of concept of how QBank can be modified to output problems in tool-specific formats.

QBank can be easily extended further by developers to support other formats, as explained in the QBank Developer Manual [34].

Chapter 5

QBank Interface

Based on the formalized definition for a problem developed in Section 3.2, the QBank framework has specific user interfaces to simplify authoring of different problem types.

Figure 5.1 shows the list of problem types with specialized user interface support.

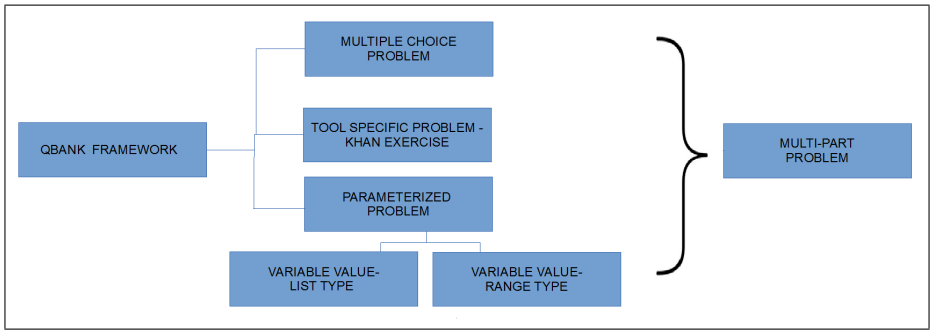


Figure 5.1: QBank – Framework.

In this chapter, I describe in detail the framework of the QBank tool, specifically explaining the features, components, and example use cases for QBank.

5.1 Software Requirements

QBank was developed using Django 1.5, a free and open source web application framework written in Python. It uses the Model-View-Controller architectural pattern which is modified to a Model-View-Template framework in Django. Django simplifies creating complex database-driven websites following the rapid development principle. For the database back end, QBank uses MySQL.

5.2 Features of the QBank framework

Some features that distinguish QBank from other existing authoring tools are:

1. Dynamic (or parameterized) problem authoring.
2. Authored problems can be exported to different formats such as CSV and Khan Academy Exercise format.
3. Tool-specific user interface that accepts inputs in the Khan Academy Exercise format, to assist authors who are familiar with using their specific markup.

5.3 QBank Home Page

Figure 5.2 shows the QBank homepage. Users have four options to start with. Authors can (a) Write a problem, (b) Search the repository for previously authored problems, (c) Export problems in different formats, and (d) Read the documentation to learn more about QBank and how to use the tool.

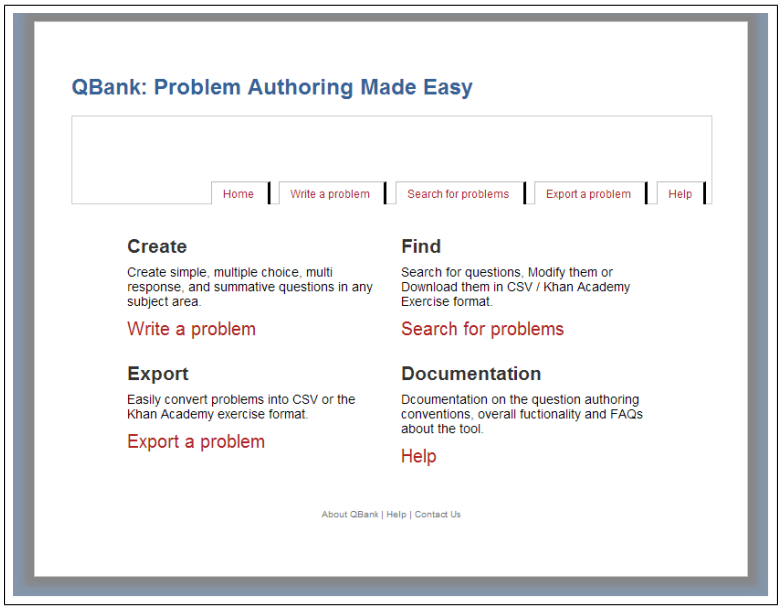


Figure 5.2: Home Screen.

In the following sections, we look in detail at how a user interacts with QBank and give a step by step process of authoring different types of problems.

5.4 Write a Problem

When the user clicks on *Write a problem*, he/she is provided with options of various types of problems that can be written.

QBank supports five different types of problems as seen in Figure 5.3.

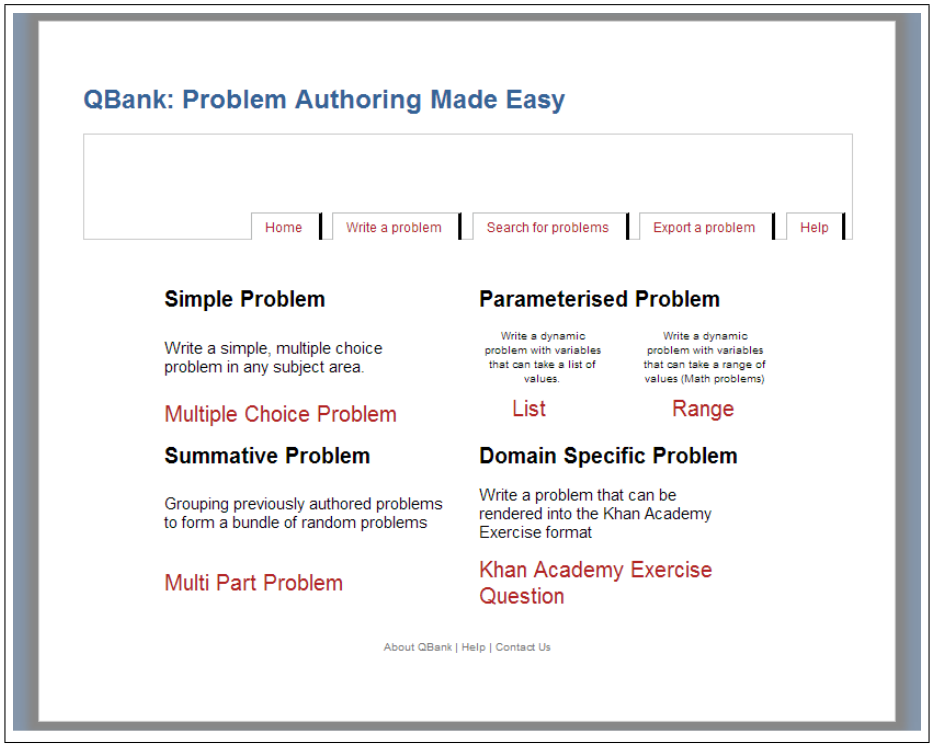


Figure 5.3: Write a Problem Screen.

The user interfaces for the various problem types follow the structure of the formal definition of a problem given in Section 3.2. Some special problem types have added features and components that make it easier for a user to author a more complex type of question.

5.5 Overview of User Interface Components

There are several conceptual components in the framework that are common among different problem types and some that are specific to particular problem types. This section provides an overview of each of these components. The interaction with each problem-type specific user interface of QBank is explained in detail in the following sections. Figure 5.4 shows the general appearance of a user interface for writing a problem in QBank. The look and feel of the user interfaces across different problem types is consistent but differ with respect to the components they include.

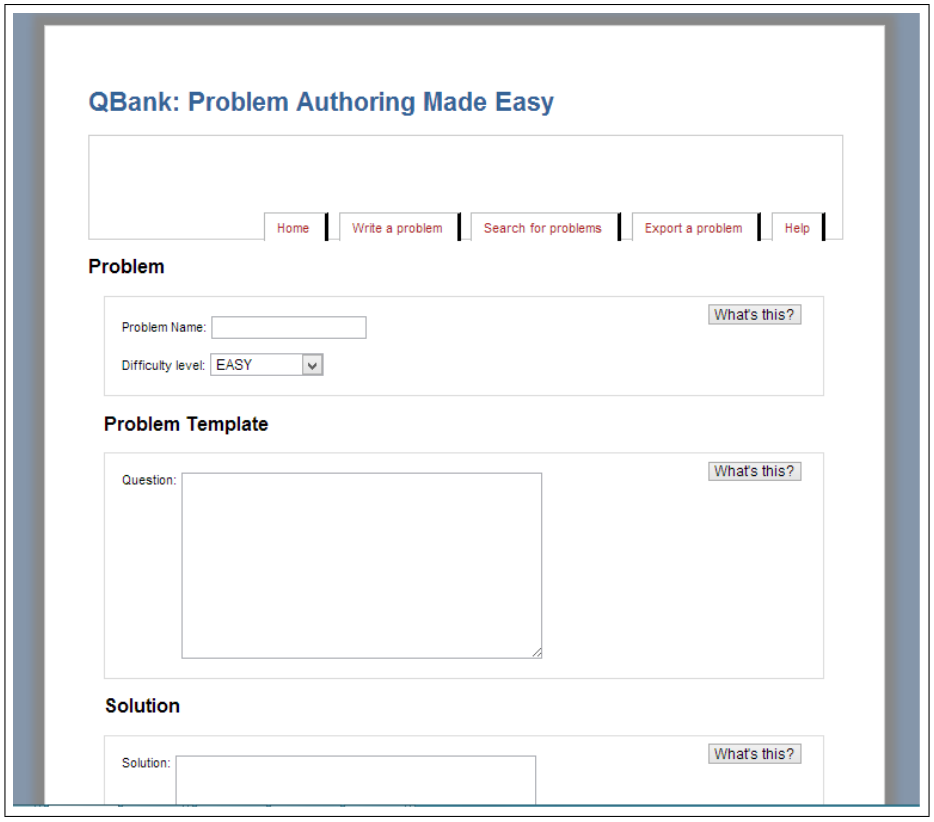


Figure 5.4: QBank Problem Editing Page.

Each *What's this?* button gives helpful indicators on the purpose of the associated text boxes and explains in detail the content, functions, additional markup elements, and parameters that can be included in that particular text box. Figure 5.5 shows the help window for the *Question* text area in the *Problem Template* for a multiple choice problem.

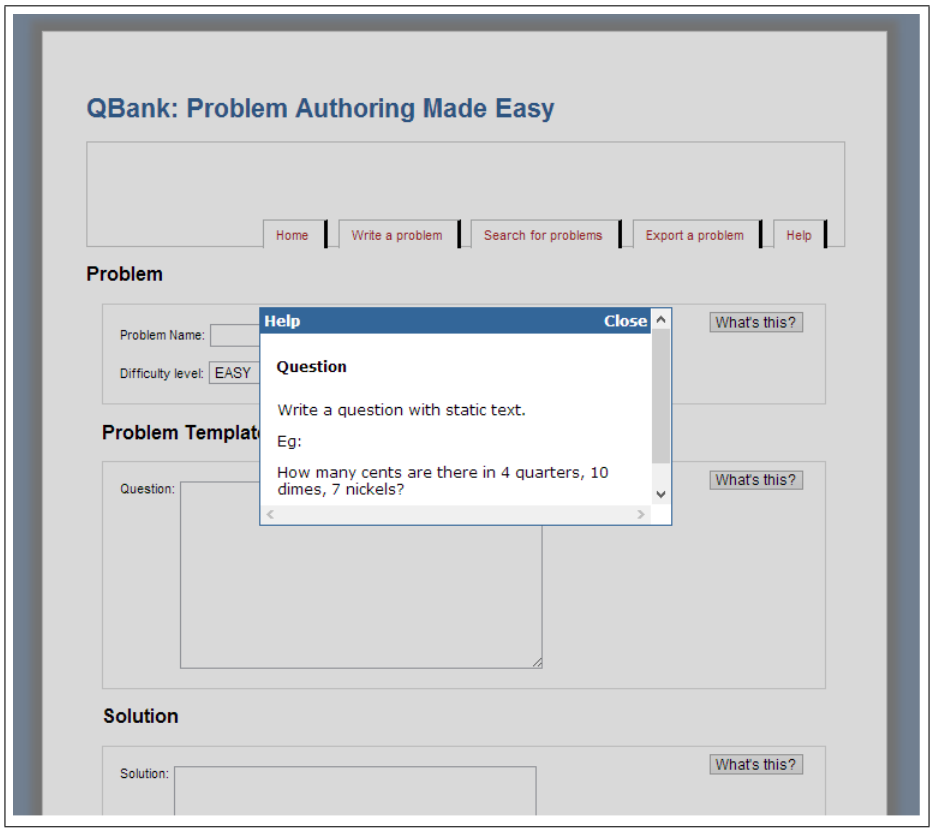


Figure 5.5: Help Window.

In following sections, I list and explain generic and specific components in the user interface for different problem types in QBank.

5.5.1 Problem

1. Problem Name

   A unique identifier that is used to (a) store a problem in the database, (b) refer to the problem when exported to different parsed formats, and (c) refer to the problem when creating a multi-part problem that aggregates existing problems.

2. Difficulty

   This value allows authors to classify problems as easy, medium, or hard. This information can be used by an Intelligent Tutor System [40] that bases difficulty of the next question posed to the user on the correctness of previous questions.

5.5.2 Variables

This allows for generation of different *problem instances* based on a static problem template with variables that take on different specified values.

Variables are typically used by specifying a <var> ... </var> element. The content of a <var> ... </var> block is executed as JavaScript.

1. Variable Name

   This is an ID for the variable that the author can use to refer to while writing a dynamic problem.

2. Variable Value

   Values that the variable can take are specified here. This includes comma separated values or functions that can be accepted by the publishing tool or parsed into a compatible format.

5.5.3 Variable Reference

Inside of <var> ... </var> delimiters, the author can refer to a previously defined variable by referencing the *variable name*. This can be used in different components of the problem like problem template, model answer (solution), choices (distractors), etc.

For example, we can define variables AVG and TIME and then multiply them together, storing them in a third variable DIST.

Variable Name: AVG
Variable Value: 31 + rand(9)
Variable Name: TIME
Variable Value: 1 + rand(9)
Variable Name: DIST
Variable Value: AVG * TIME

The **Model Answer** can either be <var> DIST</var> or <var>AVG * TIME</var>. This example shows how variables are referenced while writing the answer for a dynamic problem.

5.5.4 Problem Template

A user can specify static text for a question or text with *variables* for a dynamic problem or problem names for grouping existing problems together.

The problem template component in the user interface for a simple problem differs from that of a multi-part problem. This can be seen in Figure 5.6.

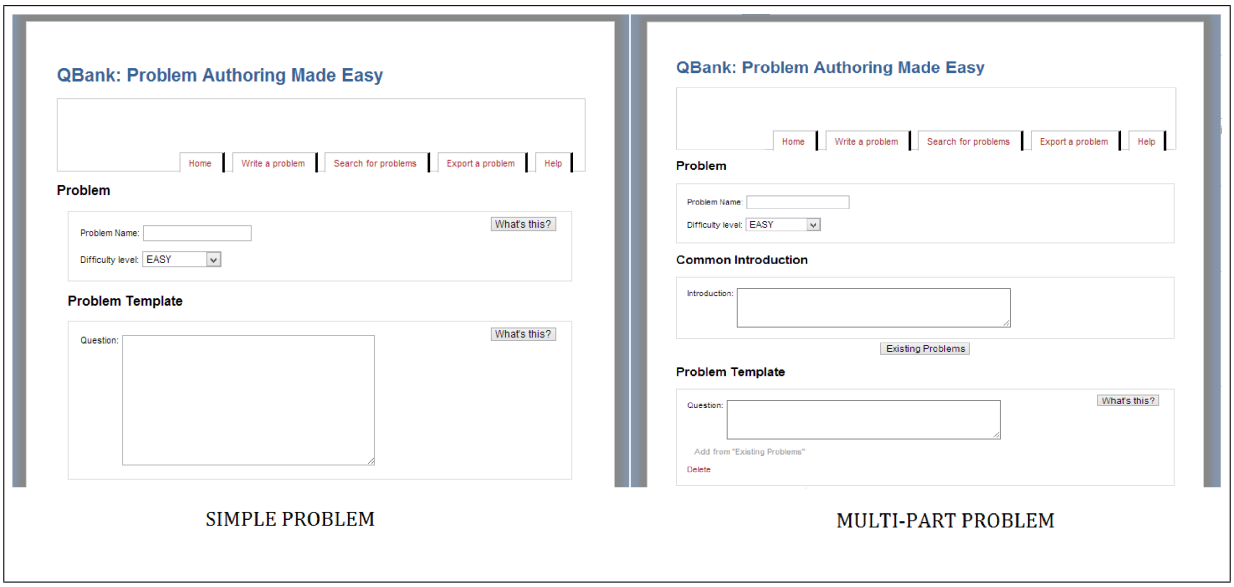


Figure 5.6: Problem Template Interface Differences for Simple and Multi-part Problems.

5.5.5 Common Introduction

The common introduction component is specific to a *multi-part problem*. This is useful since the problems grouped together may have information that is not explicitly part of the statement of the question but is common to the problems in the group. For example, in a math or physics problem, formulas that are helpful in performing a calculation or the values of universal constants can be included as part of the common introduction.

Section 5.6 explains the different problem-type specific user interfaces in QBank. We will also look at example problems for each problem type.

5.6 Problem Authoring

1. **Simple Problem**

   A simple multiple choice problem comprises static text for the question, solution, and list of distractors. In the user interface, the solution gets added as one of the choices along with some specified number of distractors.

   The *simple problem* user interface can be used for authoring true/false and multiple choice problems. Figure 5.7 shows an example of a problem with four choices (including solution) and one hint.

   User interface components:

   (a) Problem Template: The problem statement is specified here as static text.

   (b) Solution: The correct answer is specified and this is included as one of the choices.

   (c) Distractors: Choices need to be provided as distractors from the correct answer. This is written by the author as static text.

2. **Parameterized Problem**

   Parameterized (or dynamic) problems allow for writing problems with variables in them, that dynamically take variable values when they are converted to problem instances. A parameterized problem allows creation of numerous problems by the use of templates and variables, thereby saving authoring time. QBank supports two types of parameterized problems: List and Range.

   For this problem type is that there is no component for writing choices or distractors. The choices are either a mix of the other answers provided by the author (list type) or there are no choices since the user has to fill in the correct answer in the answer text box (range type).

3. **List Variables**

   In parameterized problems, the author can use the variables defined in the problem template with <var> ... </var>. In the list type of parameterized problem, a collection of values that the variable may take is specified by a list of comma-separated values enclosed in double quotes.

   For example, we can define a variable *sort_type* that takes three comma-separated strings as values.

   Variable Name: <var>sort_type</var>

   Variable Values: "Insertion Sort", "Bubble Sort", "Quick Sort"

   The author is provided with an "Answer Key"that shows the various combinations that the variables can take in a generated problem instance. The user can fill the answers in accordance to the variable which is stored in a single dimensional array. The index

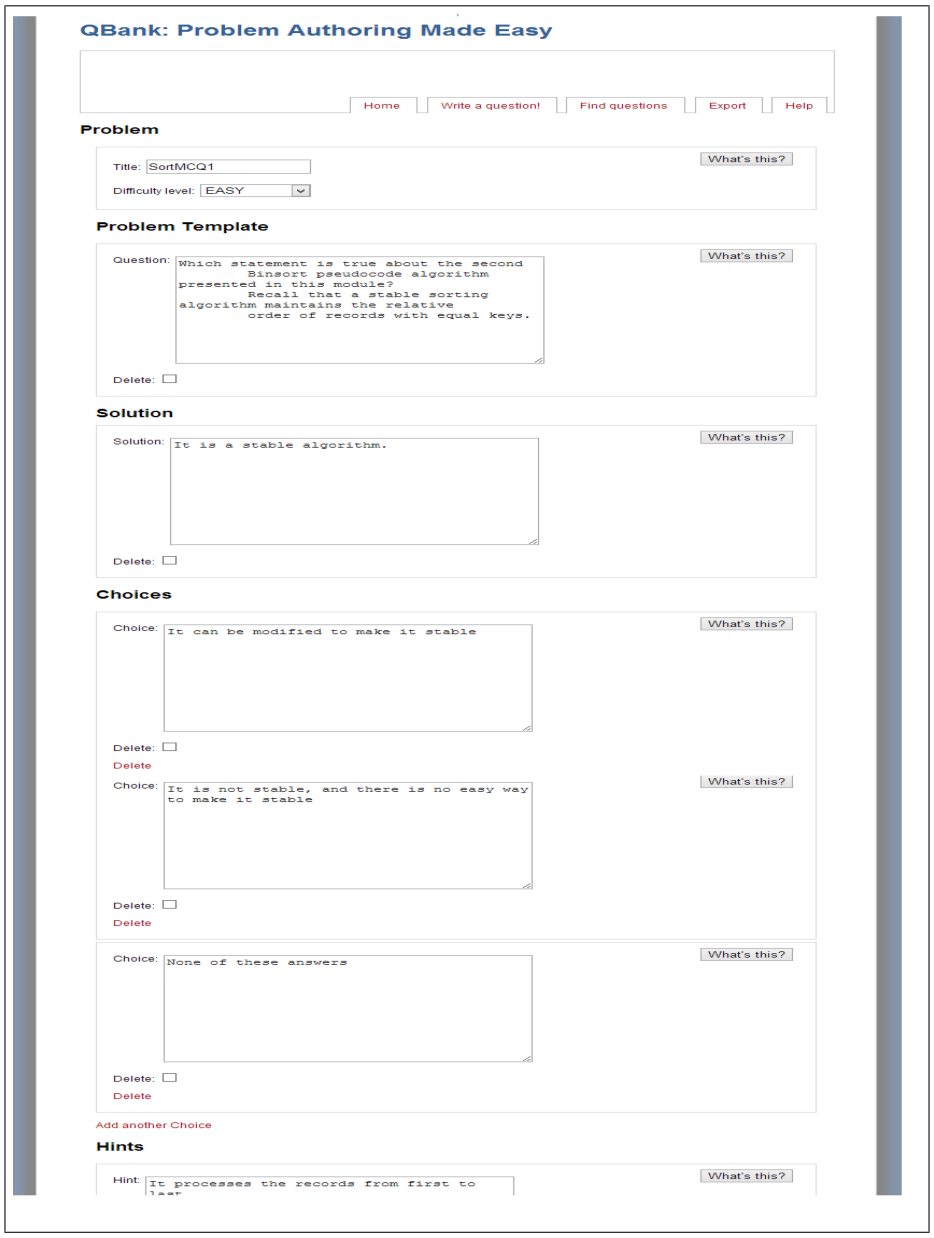


Figure 5.7: Simple Problem Screen.

of the solution in the answer array corresponds to the row in the "Answer Key"table, shown in Figure 5.8. This interface is used to write a problem with variables that take values from a specified list.

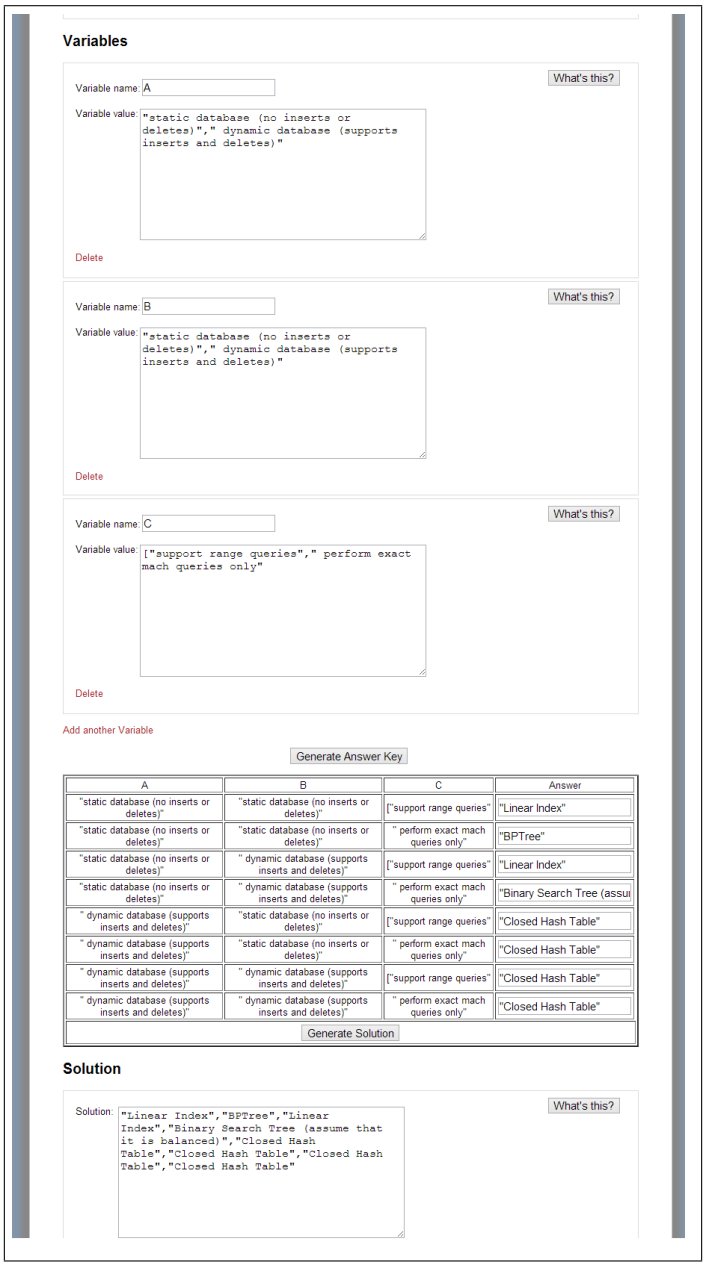
User Interface components:

(a) Variables: Values are *comma-separated* and enclosed in *double-quotes*.

(b) Generate Answer Key: This lists all combinations of the values that the variables

can take.

(c) Solution: A one dimensional array, where the array index corresponds to the answer of a particular variable combination.



Variables

What's this?

Variable name: A

Variable value: "static database (no inserts or deletes)"," dynamic database (supports inserts and deletes)"

Delete

What's this?

Variable name: B

Variable value: "static database (no inserts or deletes)"," dynamic database (supports inserts and deletes)"

Delete

What's this?

Variable name: C

Variable value: ["support range queries"," perform exact mach queries only"

Delete

Add another Variable

Generate Answer Key

A	B	C	Answer
"static database (no inserts or deletes)"	"static database (no inserts or deletes)"	["support range queries"	"Linear Index"
"static database (no inserts or deletes)"	"static database (no inserts or deletes)"	" perform exact mach queries only"	"BPTree"
"static database (no inserts or deletes)"	" dynamic database (supports inserts and deletes)"	["support range queries"	"Linear Index"
"static database (no inserts or deletes)"	" dynamic database (supports inserts and deletes)"	" perform exact mach queries only"	"Binary Search Tree (assu
" dynamic database (supports inserts and deletes)"	"static database (no inserts or deletes)"	["support range queries"	"Closed Hash Table"
" dynamic database (supports inserts and deletes)"	"static database (no inserts or deletes)"	" perform exact mach queries only"	"Closed Hash Table"
" dynamic database (supports inserts and deletes)"	" dynamic database (supports inserts and deletes)"	["support range queries"	"Closed Hash Table"
" dynamic database (supports inserts and deletes)"	" dynamic database (supports inserts and deletes)"	" perform exact mach queries only"	"Closed Hash Table"

Generate Solution

Solution

What's this?

Solution: "Linear Index","BPTree","Linear Index","Binary Search Tree (assume that it is balanced)","Closed Hash Table","Closed Hash Table","Closed Hash Table","Closed Hash Table"

Figure 5.8: *Answer Key* Table with all Possible Variable Value Combinations Shown.

4. **Range Variables**

   Range variable values are specified using the $randRange(min, max)$ function. The specification for range variable values and solutions differ from the list problem type. In the $randRange(min, max)$ function, min is the lower bound of the range of values the variable can take and max is the upper bound. This type of problem is typically for math questions where the solution is calculated.

   User Interface components:

   (a) Variables: The values the variables take fall within the *min-max* range specified in the $randRange(min, max)$ function.

   (b) Solution: The author does not need to explicitly make the calculation and write the correct answer. The solution is the calculation in the problem template specified within <var> </var> delimiters.

   For instance, a question has three variables A, B and C and it's problem template is "What is the result of <var> A</var>+ <var> B</var> -<var> C</var> ?". The solution in this example is <var>A+B-C </var>.

   Based on the values that the variables dynamically take in each problem instance, the solution is calculated accordingly. This ensures the accuracy of the answer and is useful in authoring problems with more complex calculations.

   Figure 5.9 shows an arithmetic problem authored using this interface.

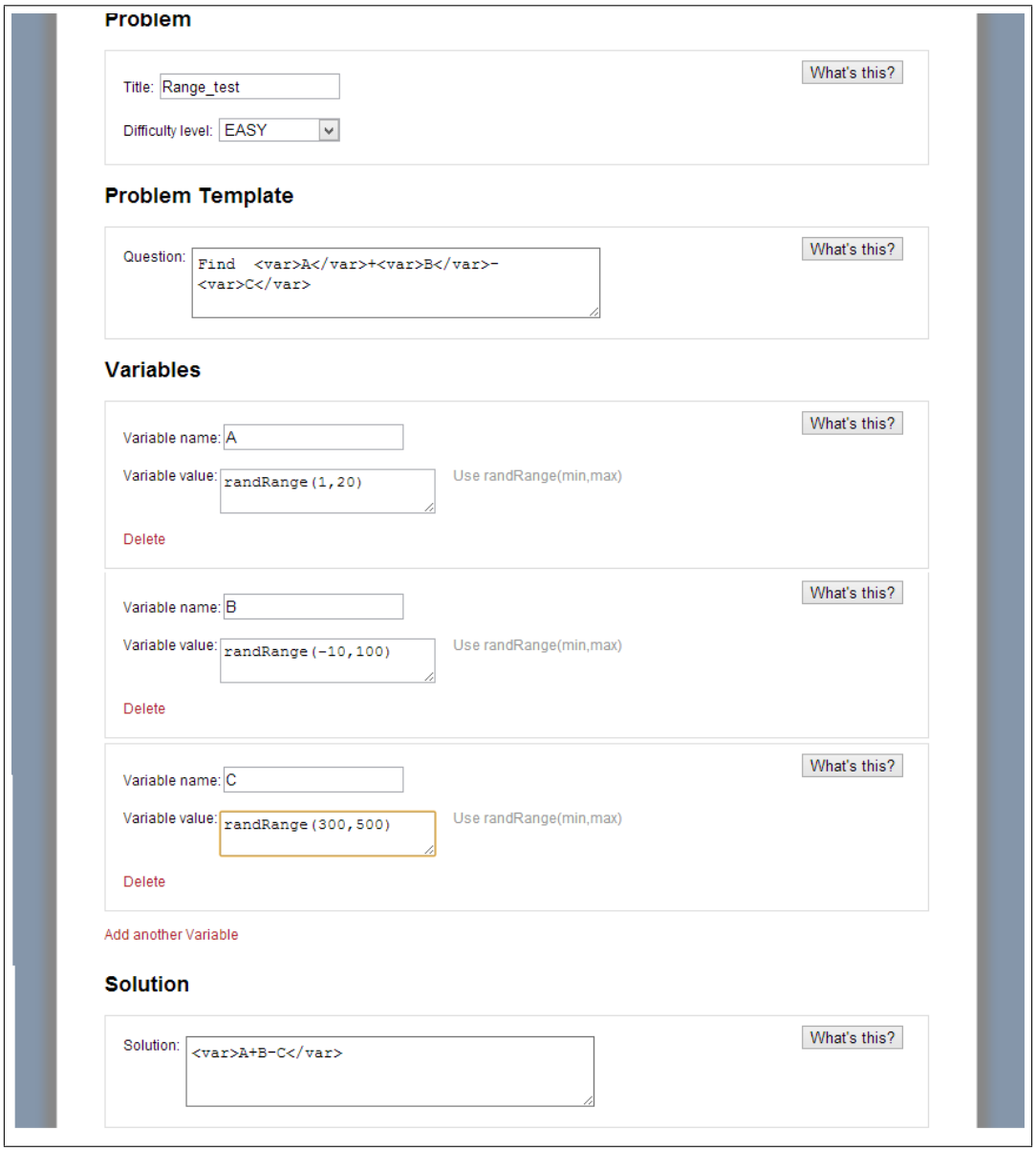


Figure 5.9: Variable Values Specified as a Range.

5. **Multi-Part Problem**

   Multi-part problems allow for grouping problems together. This allows for reusing previously written problems in a different context. These problems when presented to a learner offers them a variety of different problems in the same domain of knowledge.

   The author chooses or specifies the *Problem Name* of the existing problems that he/she wants to group together as shown in Figure 5.10.

   The *Question* in a Summative problem is essentially a *Problem name* that can be seen

in Figure 5.11.

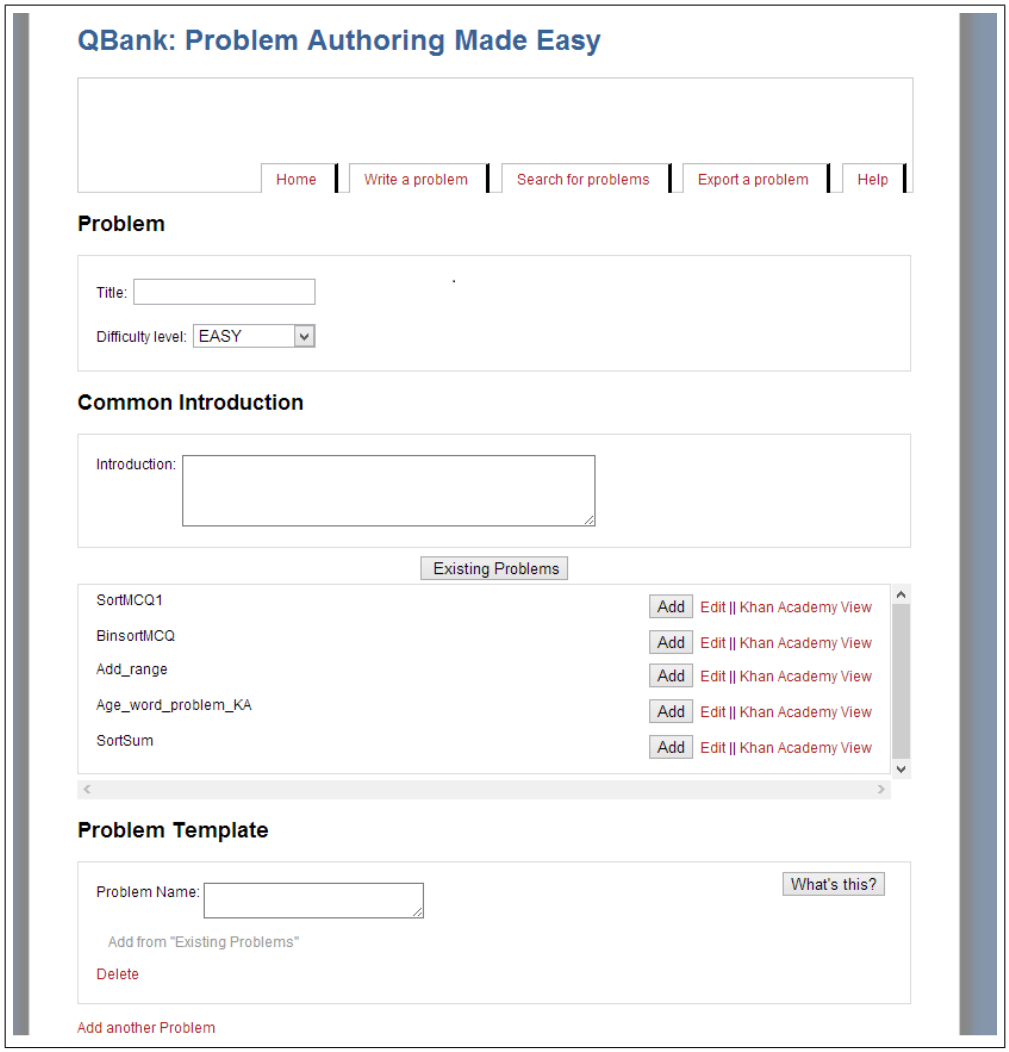


Figure 5.10: Multi-part Problem User Interface.

User interface components:

(a) Common Introduction: The common introduction generally contains information that is common to the questions.

(b) Problem Name: The author can browse the existing problems by clicking the *Show/Hide* button and can add the problem name to the group by clicking the *Add* button adjacent to a selected problem name.

QBank: Problem Authoring Made Easy

Home | Write a problem | Search for problems | Export a problem | Help

Problem

Problem Name: SortSum

Difficulty level: EASY

Common Introduction

What's this?

Introduction: Sorting Exercises

Delete: ☐

Existing Problems

SortMCQ1	Add	Edit ‖ Khan Academy View
Insertion Sort Proficiency Exercise	Add	Edit ‖ Khan Academy View
Insertion_sort_MCQ	Add	Edit ‖ Khan Academy View
ODSAeg	Add	Edit ‖ Khan Academy View
Range1	Add	Edit ‖ Khan Academy View
Age_word_problem_KA	Add	Edit ‖ Khan Academy View
SortSum	Add	Edit ‖ Khan Academy View

Problem Template

What's this?

Problem Name: SortMCQ1

Delete: ☐

Add from "Existing Problems"

Delete

What's this?

Problem Name: BinsortMCQ

Figure 5.11: Multi-Part Problem Example.

6. **Khan Academy Exercise Question**

   QBank provides a separate user interface for authors proficient in the Khan Academy syntax and markup. The user interface provided by QBank abstracts common information that Khan Academy exercises share. It also provides a front end consistent with the formalized problem definition.

   The user is provided with the extra capability of including scripts that can be utilized in authoring a problem. This type of problem demands a higher level of technical expertise in order to create a well-written problem. An example *checkAnswer* function written in JavaScript is shown in Figure 5.12.

   This function is part of an interactive problem shown in Figure 6.6. The user is provided with a visualization of an array and is expected to click on it to show the next position that the highlighted cell element will move to based on the Insertion sort algorithm. The *checkAnswer* function takes the size of the array as a parameter and verifies the correctness of the user's answer array with the model answer array by checking the type and value of the element at each index location. Depending on the author's programming skills, complex problems with more features can be created.

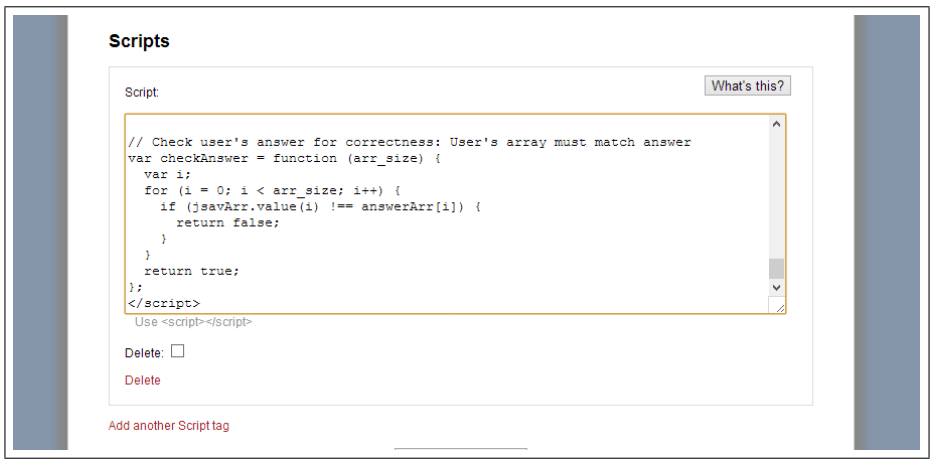


Figure 5.12: Script Snippet.

The Khan Academy Exercise framework Documentation [24] explains how to author Khan Academy exercises. Also, the *What's this* button in QBank gives a lot of direction in assisting an author while writing a problem.

For instance, Figure 5.13 shows a sample Khan Academy exercise problem authored in QBank. The name of the example problem is *Age_word_problem_KA*. This is a dynamic *Fill-in-the-Blank* problem.

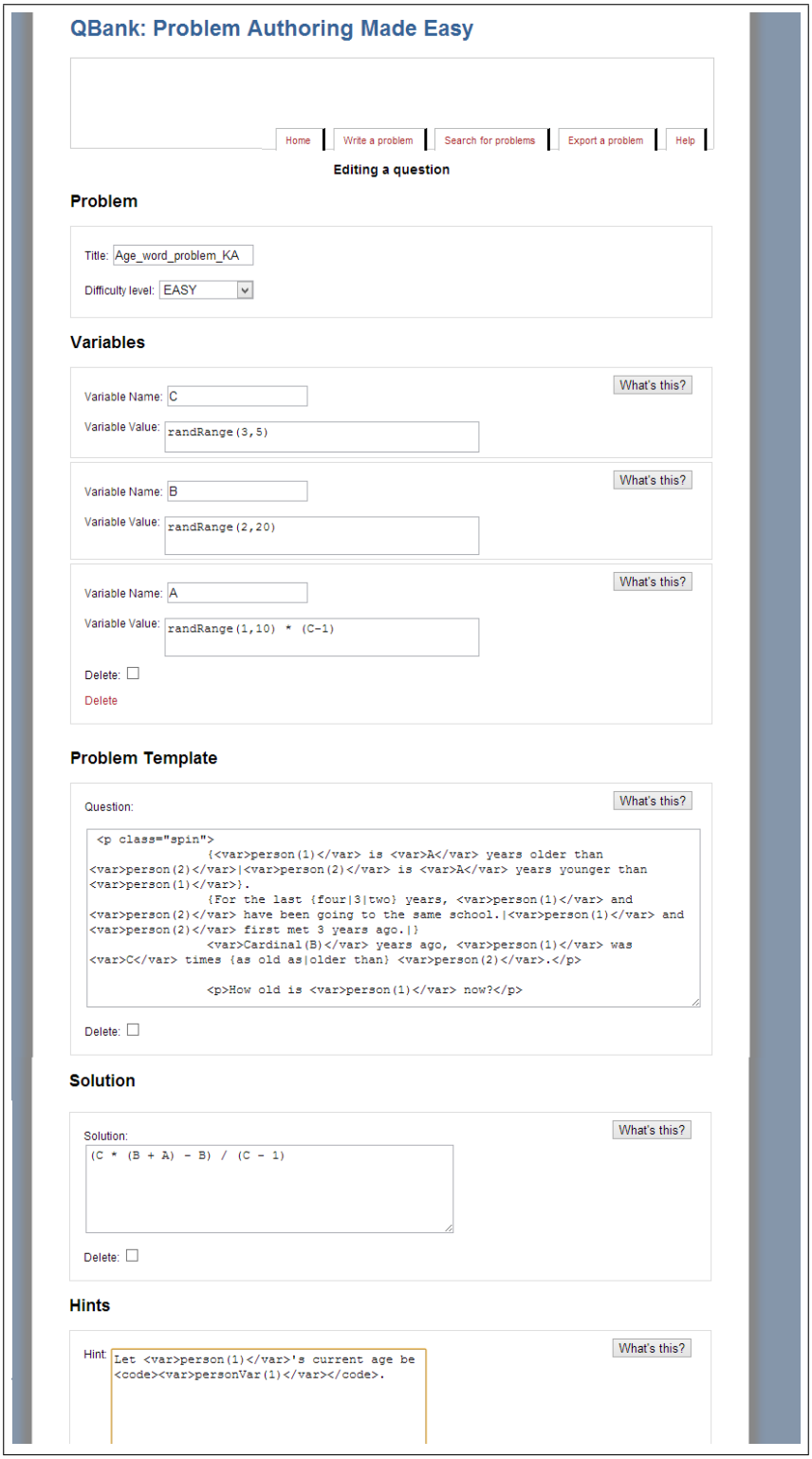


Figure 5.13: Tool Specific Problem – Khan Academy Exercise Question.

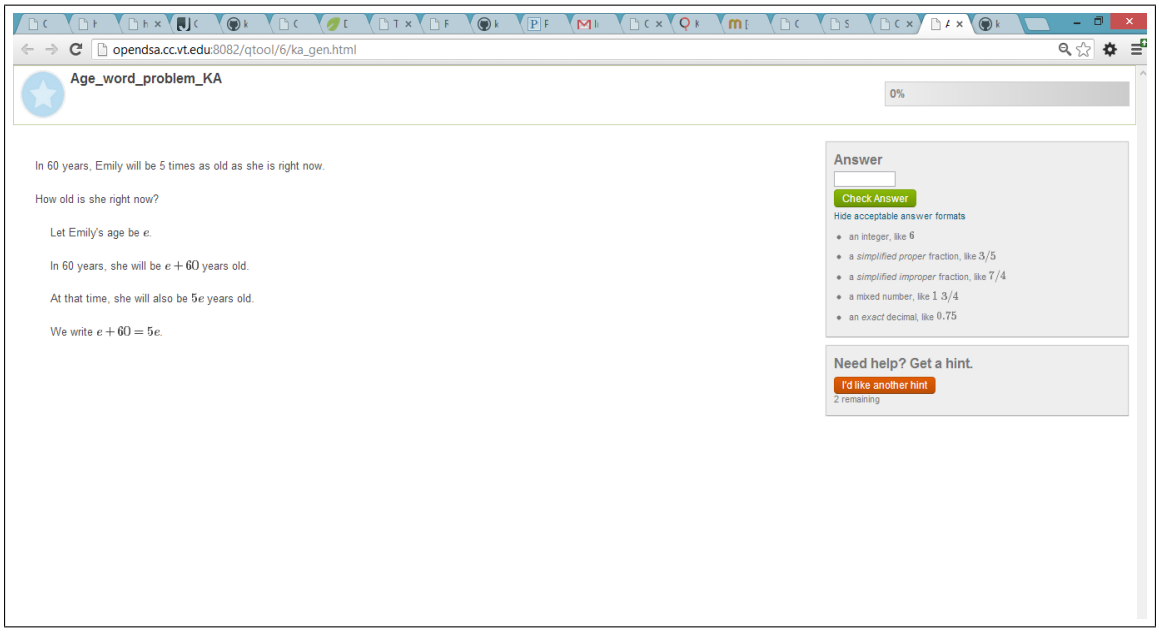


Figure 5.14: Viewing the Exported Khan Academy Exercise Age_Word_Problem_KA in a Web Browser.

5.7 Exporting a Problem

The current version of the QBank supports CSV output and the Khan Academy Exercise format.

5.7.1 CSV Format

All problems can be exported as a comma-separated file by clicking the *CSV* link in the *Export a problem* tab.

Figure 5.15 shows a dynamic problem called *Age_Word_Problem_KA* in CSV format.

```
1  TITLE,Age_word_problem_KA
2  DIFFICULTY,EASY
3  QUESTION," <p class=""spin"">
4                 {<var>person(1)</var> is <var>A</var> years older than <var>person(2)</var>|<var>person(2)</var> is <var>A</var> years younger than <var>person(1)</var>}.
5                 {For the last {four|3|two} years, <var>person(1)</var> and <var>person(2)</var> have been going to the same school.|<var>person(1)</var> and <var>person(2)</var> first
                  met 3 years ago.|}
6                 <var>Cardinal(B)</var> years ago, <var>person(1)</var> was <var>C</var> times {as old as|older than} <var>person(2)</var>.</p>
7
8                 <p>How old is <var>person(1)</var> now?</p>"
9  VAR_NAME,C
10 VAR_VALUE,"randRange(3,5)"
11 ATTR_INFO,
12 VAR_NAME,B
13 VAR_VALUE,"randRange(2,20)"
14 ATTR_INFO,
15 VAR_NAME,A
16 VAR_VALUE,"randRange(1,10) * (C-1)"
17 ATTR_INFO,
18 SCRIPT,
19 CHOICE,
20 HINT,Let <var>person(1)</var>'s current age be <code><var>personVar(1)</var></code>.
21 HINT,"That means that <var>B</var> years ago, <var>person(1)</var> was <code><var>personVar(1)</var> - <var>B</var></code> years old."
22 SOLUTION,(C * (B + A) - B) / (C - 1)
```

Figure 5.15: Example Problem in CSV Format.

The ability to download authored problem in CSV format allows it to be converted to input formats compatible with other problem publishing systems.

For instance, Figure 5.16 shows the CSV file of a multiple choice problem for hash sort authored in QBank called *HashMCQ1*.

In order to explain the capability of this feature, I convert the CSV file of the problem to two other CSV formats: (a) CSV quiz import format for Moodle [12] and (b) QuestionMark Live CSV import format [5], which are shown in Figures 5.17 and 5.18 respectively.

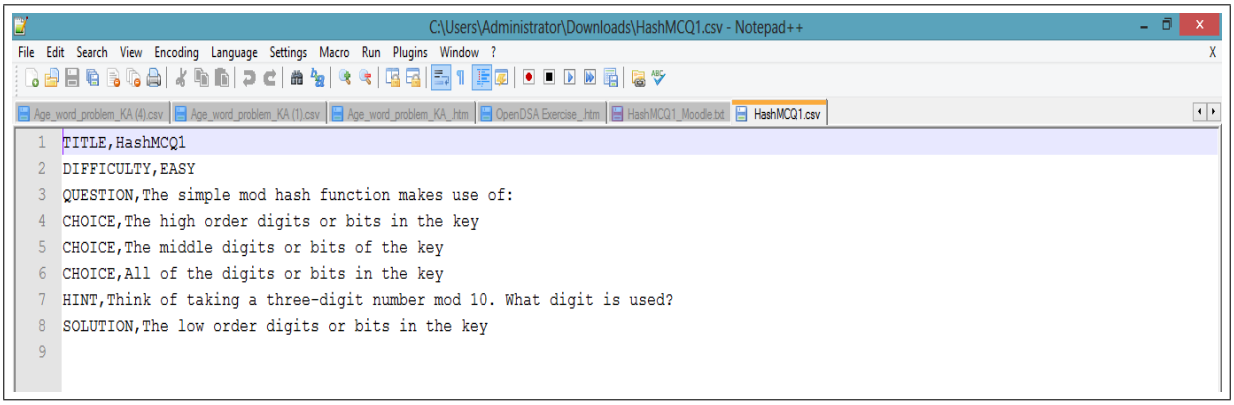


Figure 5.16: HashMCQ1 in CSV Format Exported from QBank.

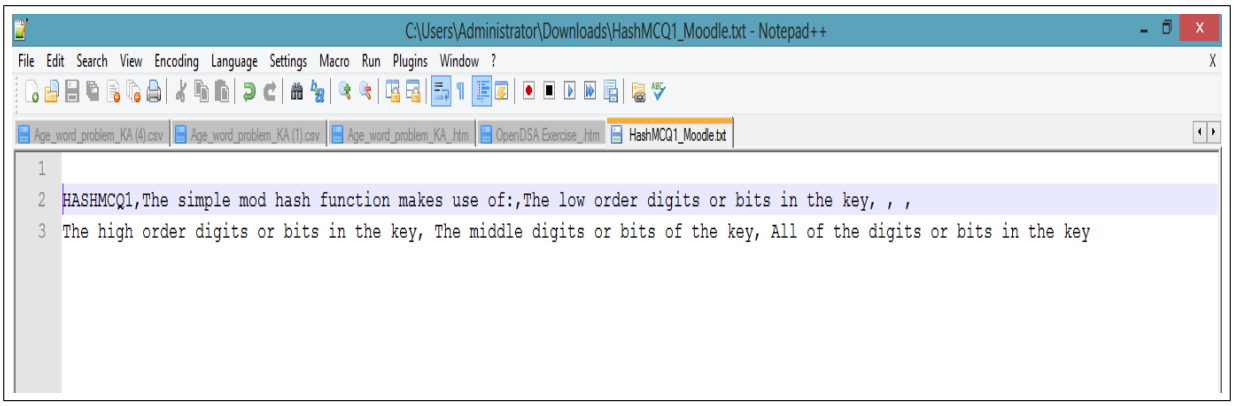


Figure 5.17: HashMCQ1 in CSV Quiz Format for Moodle.

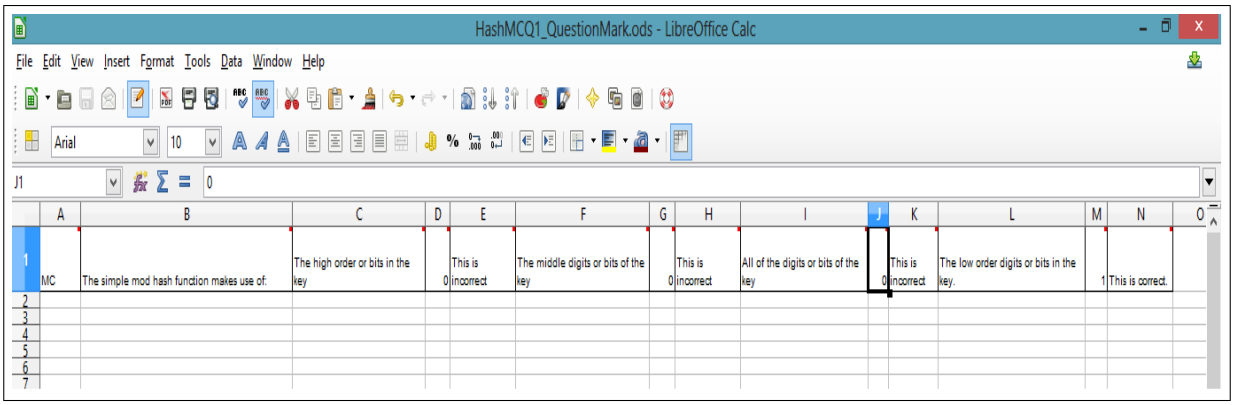


Figure 5.18: HashMCQ1 in the File Format for the Questionmark Live CSV Import.

5.7.2 Khan Academy Exercise Format

Problems can be exported to an HTML file fully compatible with the Khan Academy Exercise infrastructure by clicking on the *KA exercise* link in the *Export a problem* tab.

Figure 5.19 shows an example of a static problem called *HashMCQ1* in Khan Academy Exercise format. *HashMCQ1* exercise rendered in a web browser is shown in Figure 5.20.

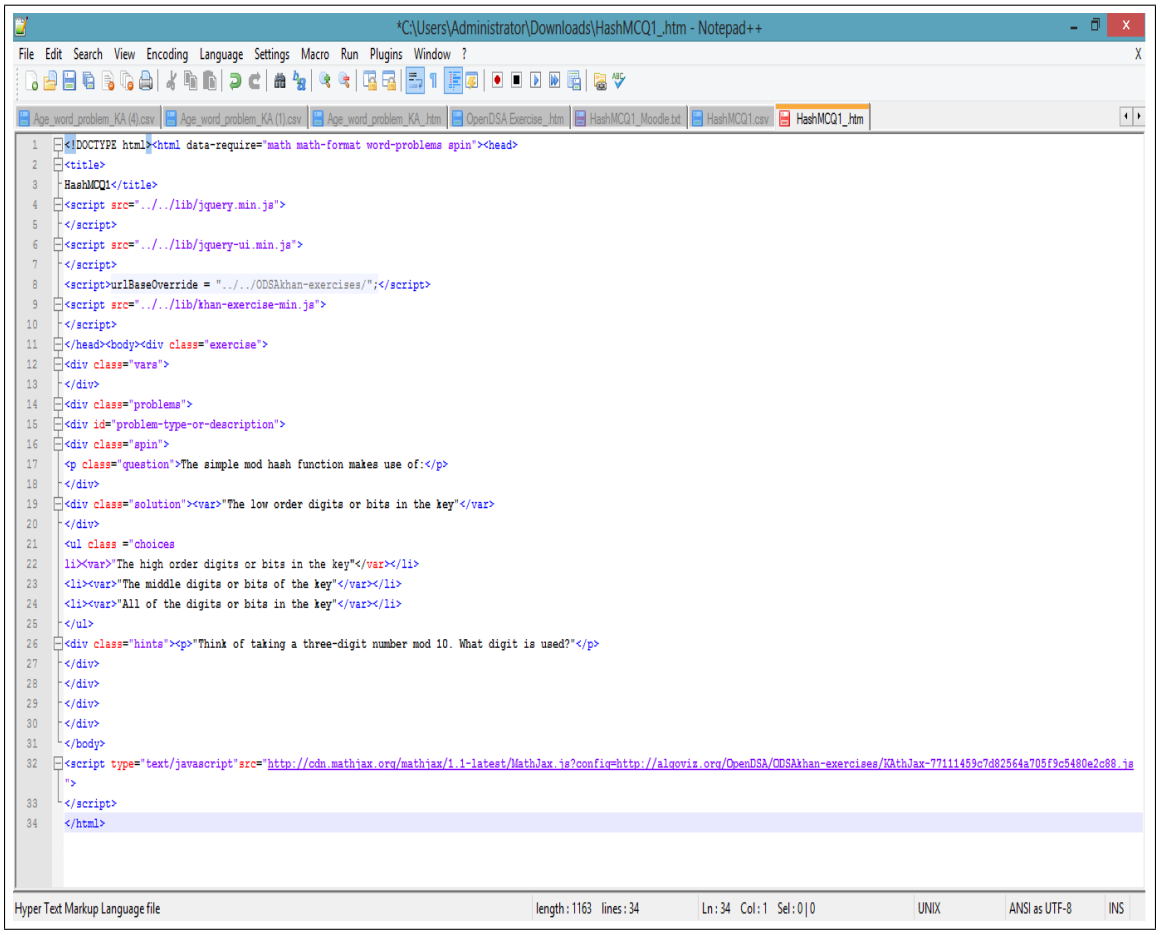


Figure 5.19: Khan Academy Exercise Format of a Static Problem called HashMCQ1.

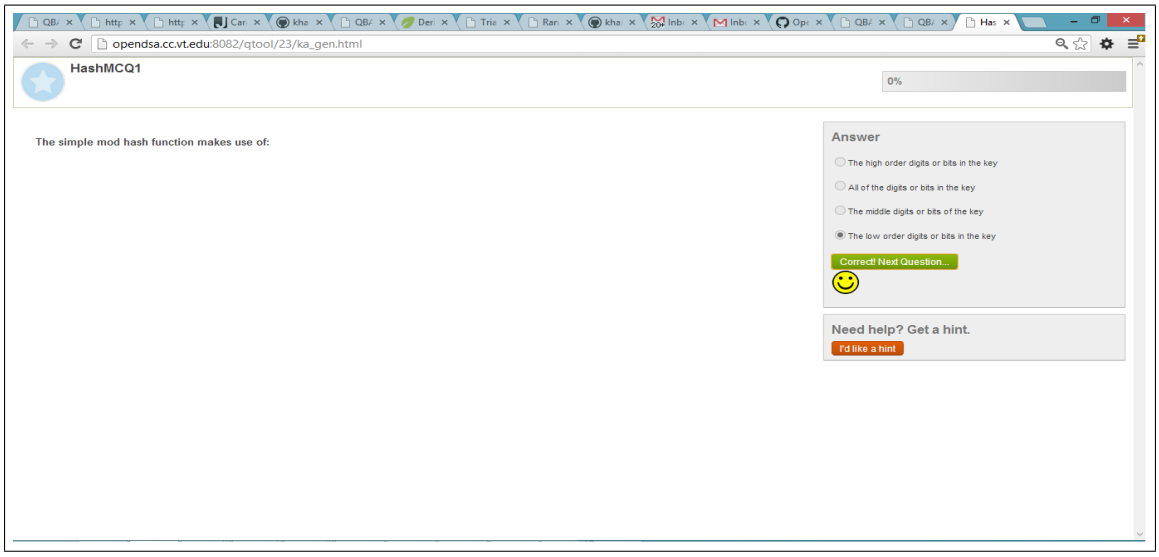


Figure 5.20: HashMCQ1 in a Web Browser.

Figure 5.21 shows an example of a dynamic problem called *Age_Word_Problem_KA* in Khan Academy Exercise format. Figure 5.14 shows the exported Khan Academy exercise of *Age_Word_Problem_KA* in a web browser.

```
1   <!DOCTYPE html>
2   <html data-require="math math-format word-problems spin"><head>
3   <title>
4    Age_word_problem_KA
5   </title>
6   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js">
7   </script>
8   <script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.16/jquery-ui.min.js">
9   </script>
10   <script>urlBaseOverride = "../qtool/";</script>
11  <script src="../qtool/khan-exercise.js">
12  </script>
13  <script type="text/javascript"src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
14  </script>
15   <p></p>
16  </head>
17  <body>
18  <div class="exercise">
19  <div class="problems">
20  <div id="1">
21  <div class="vars">
22   <var id="C">randRange(3,5)</var>
23   <var id="B">randRange(2,20)</var>
24   <var id="A">randRange(1,10) * (C-1)</var>
25  </div>
26   <div class="question">
27   <p class="spin">
28                 {<var>person(1)</var> is <var>A</var> years older than <var>person(2)</var>|<var>person(2)</var> is <var>A</var> years younger than <var>person(1)</var>}.
29                 {For the last {four|3|two} years, <var>person(1)</var> and <var>person(2)</var> have been going to the same school.|<var>person(1)</var> and <var>person(2)</var>
                   first met 3 years ago.|}
30                 <var>Cardinal(B)</var> years ago, <var>person(1)</var> was <var>C</var> times {as old as|older than} <var>person(2)</var>.</p>
31
32                 <p>How old is <var>person(1)</var> now?</p></div>
33  <div class="solution">
34   <var>(C * (B + A) - B) / (C - 1)</var>
35  </div>
36  <div class="hints">
37   <p>Let <var>person(1)</var>'s current age be <code><var>personVar(1)</var></code>.</p>
38   <p>That means that <var>B</var> years ago, <var>person(1)</var> was <code><var>personVar(1)</var> - <var>B</var></code> years old.</p>
39  </div>
40  </div>
41  </div>
```

Figure 5.21: Khan Academy Exercise Format for a Dynamic Problem.

Chapter 6

Tool Assessment and Usability Study

This chapter outlines the assessment methodology for testing QBank, the goals of the usability study, a summary of the observations made, and conclusions from this study.

6.1 Goals of the Assessment

After QBank was developed, I conducted a usability study to evaluate the effectiveness of the tool and to ensure it achieved the objectives it set out to meet. The main goals of this study were:

1. To evaluate the capability of QBank to author dynamic problems.
2. To determine if the user interface is intuitive and easily navigable.
3. To compare the effectiveness of QBank in comparison with other authoring tools/frameworks that the participants previously used.
4. To study the usefulness of providing export to a generic CSV format and the Khan Academy Exercise format.

6.2 Methodology

In order to evaluate QBank, I approached two students in my lab group who author questions which are incorporated into OpenDSA [10], a hypertextbook for data structures and algorithms courses and two other friends outside of my lab group. This section describes the participants, and the tasks that the participants were asked to perform. Observations were made based on discussions with the participants while interacting with QBank.

6.2.1 Participants

The testing was done on four participants. Two of the participants are computer science graduate students in my lab group (one female, one male) who have some experience using Khan Academy to create exercises. Two other participants (one female, one male) are software engineers in industry.

6.2.2 Evaluation Criteria

The task was to author questions using the five different user interfaces provided in QBank. Each of the participants were asked to evaluate QBank on the following aspects:

1. Ability to understand/perceive the features of QBank.
2. Online help and documentation.
3. Usefulness and accuracy of implementation
4. Aesthetic aspects and overall design.
5. Effectiveness of navigation and positioning of elements on the interface.

6.2.3 Procedure

Participants were given a brief introduction to QBank, its key features, and a demonstration of it working. They were each asked to author the simplest questions they could think of using the five interfaces to evaluate the functioning of QBank. While performing the task, I sat with each participant individually and they were asked to speak aloud their intentions and interactions with the tool. Based on the listed points of interest, all their remarks and queries were recorded. Additional notes included their suggestion on improvement and possible future work. The studies on an average took around 50 minutes each.

6.3 Observations

This section summarizes the feedback from the participants during the usability studies and the results of the evaluation based on the criteria listed in Section 6.2.2.

6.3.1 General Feedback

The participants started with the *Write a problem* button since authoring a problem was the task assigned to them. Each of them attempted to write simple problems for each of the different problem types. Except for the Khan Academy Exercise, they were able to create simple, parameterized, and multi-part problems with support from *What's this?* help button. For the tool-specific Khan Academy Exercise, three of the participants didn't author a problem since they found the Khan Academy documentation too complex to read and understand in that short time span. Also, after glancing through the Khan Academy documentation, the only extra feature that stuck out to most participants was the ability to use *variables*.

A common comment that all the participants made was whether a separate interface for a Khan exercise was essential since *parameterized problem type* handles the use of variables. The lack of examples that showcase and highlight how interactive (or complex) exercises can be written using JavaScript functions in the Khan Academy Exercise Framework documentation made it difficult for the participants to author a question easily. One of the participants, who has experience writing simple multiple choice questions in the Khan Academy format [25], used the tool-specific interface to author a simple multiple choice question using the Khan Academy Exercise markup to test the interface for accuracy in implementation.

6.3.2 Feedback Based on Evaluation Criteria

The 'list' problem type requires a randomizer to be specified in square brackets when declaring the variable names in the problem. One participant suggested to abstract that particular detail from the problem, which makes authoring the problem more straightforward. On the other hand, the *Generate Answer Key* and *Generate Solution* buttons (which open up a table with combinations of variables specified and adds a comma separated string of answers to the solution) was seen to be very useful. One participant said that it makes the task of writing the solution easy and ensures accuracy.

A button to directly view the parsed Khan Academy exercise in the browser without having to do a two-step 'generate and view KA exercise'was requested. The participants also remarked that a separate menu tab for *Export a problem* would be beneficial for users who are only interested in exporting previously written problems to be used elsewhere.

The examples and documentation in the *What's this?* button made it easier to author a problem without looking at a separate document with exhaustive details about all other components not necessary for that particular component. Viewing the generated Khan Academy Exercise from the problem specification validates that QBank met its development goals. In terms of user interface design, the participants were satisfied with the look of QBank and found that navigation between screens was smooth.

6.3.3 Results of the Evaluation

Based on the goals of assessment listed in Section 6.1, I evaluated QBank and summarize the results in this section.

1. Capability of QBank to author dynamic problems.

   The participants commented that the tool was effective in authoring dynamic problems using the specialized interfaces since it effectively handles the complexity of including variables in problem authoring.

2. Is the user interface intuitive and easily navigable?

   Since the user interfaces in QBank are designed specific to a problem type, the participants were able to focus on the content once they determined the type of question they chose to write. They were not overwhelmed or distracted by the features specific to other problem types.

3. Effectiveness of QBank in comparison with other authoring tools/frameworks that the participants previously used.

   One of the participants who has extensive experience in authoring questions in Moodle and using the Khan Academy Exercise format found this tool useful. She pointed out that the consistency of the user interface in terms of the components associated with different problem types (static or dynamic) made it easy for her to identify the various parts of the problem and concentrate on the content required to author a simple/complex problem. She also expressed that QBank effectively abstracted the code specific details from the author while writing problems in the Khan Academy Exercise format.

4. Usefulness of providing export to a generic CSV format and a specific Khan Academy Exercise format.

   CSV format is a commonly used import format for problem authoring and publishing systems. For instance, Moodle accepts multiple choice questions in this format [12]:

   "Category";"Question text";"CA 1";"CA 2";...;"CA n";"";"WA 1";"WA 2";...;"WA m". Each row represents a question. *CA* refers to the correct answer and *WA* refers to the wrong answer. Empty cells separate correct answers from wrong ones. The problems exported in the CSV format using QBank has all the information necessary to be parsed to the CSV format supported by Moodle.

   Additionally, the provision to download an authored problem in the Khan Academy Exercise format was seen as useful by a participant who wanted to incorporate the authored problems into OpenDSA [10].

6.4 Examples of Authored Problems

This section presents three examples of authored problems. They are: (a) Static multiple choice problem, (b) Dynamic problem with list type of variable values, and (c) Khan Academy exercise question.

1. Figures 6.1 shows a static multiple choice problem authored in QBank. The problem is exported to the Khan Academy Exercise format and is displayed in a web browser as shown in Figure 6.2.

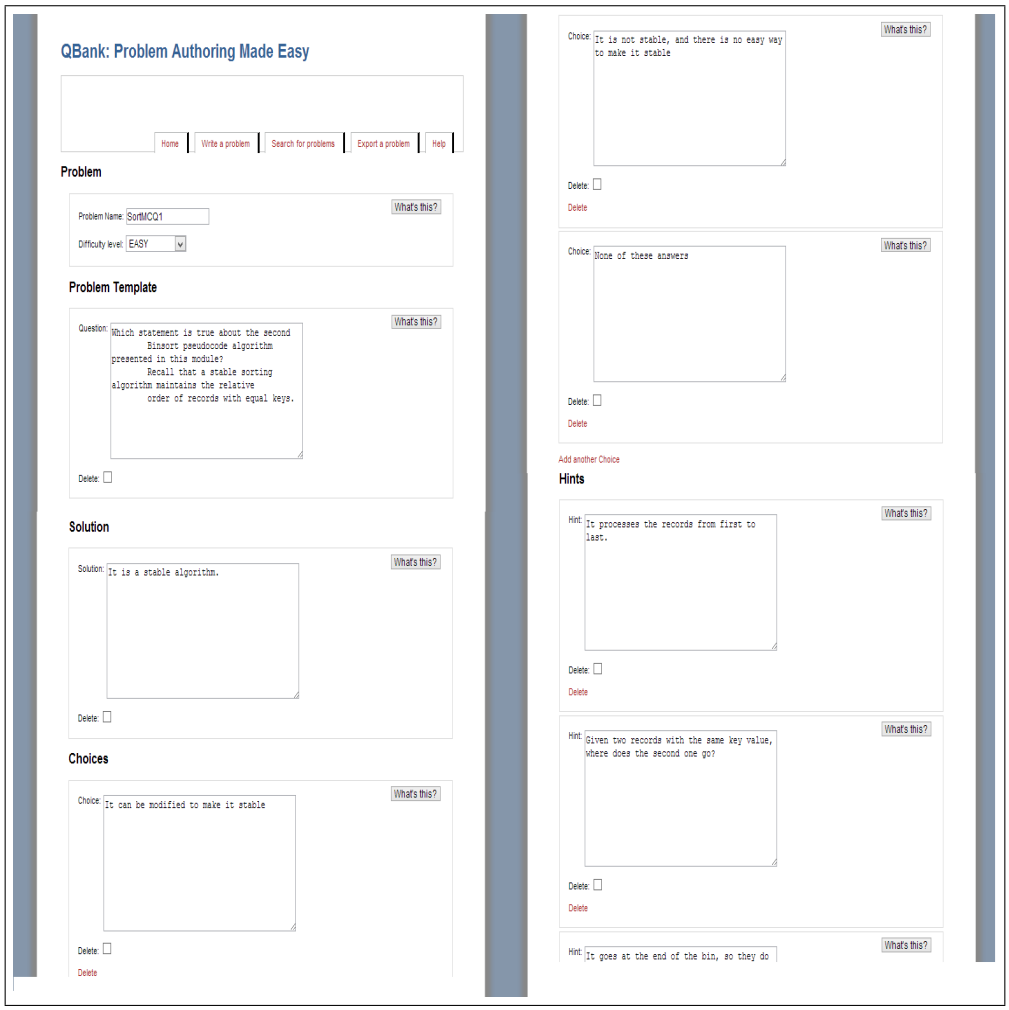


Figure 6.1: Writing a Static Multiple Choice Problem for Sorting in QBank.

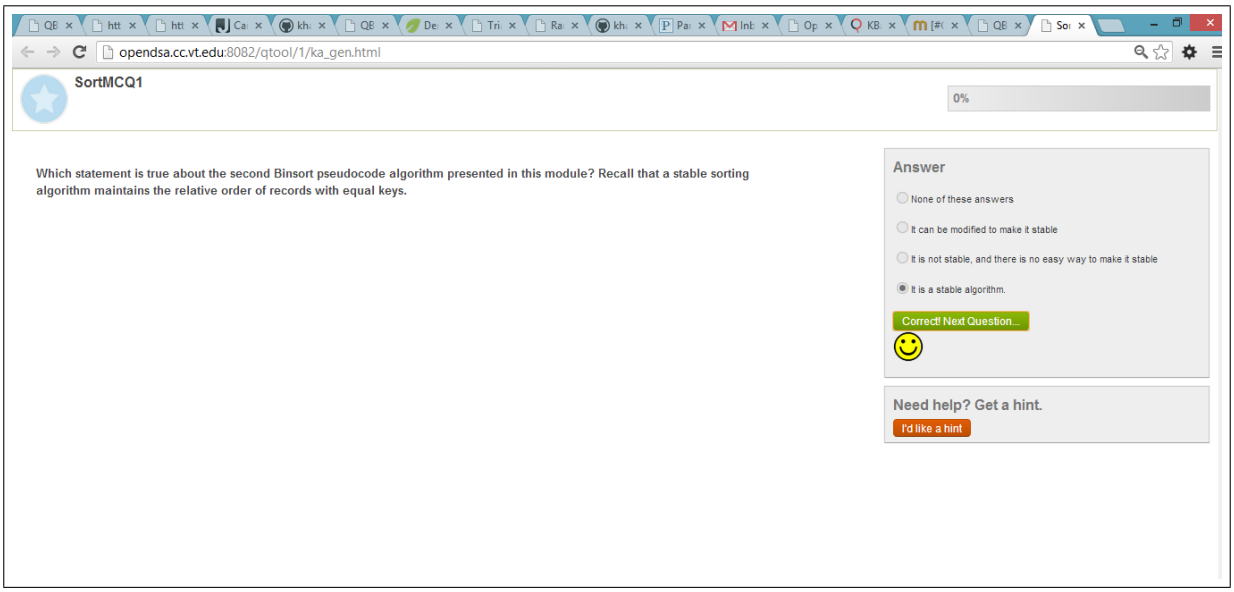


Figure 6.2: The Exported Khan Academy Exercise for Figure 6.1

2. The second example is a dynamic problem with variables that take a list of values. Figure 6.3 shows the problem authored in the QBank interface. Figure 6.4 shows the problem in a browser after converting it into the Khan Academy Exercise format.

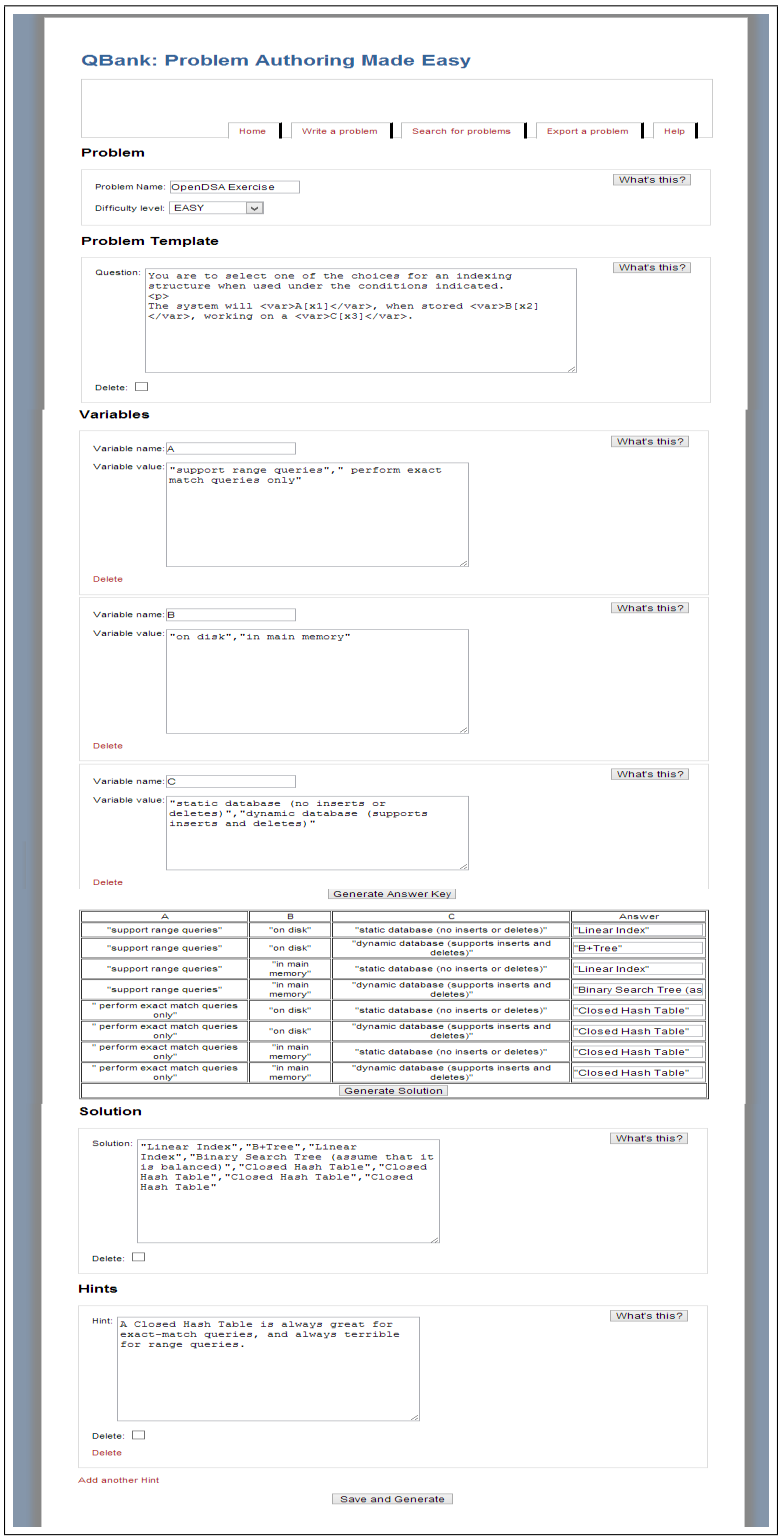


Figure 6.3: Writing a List Type Dynamic Multiple Choice Problem.

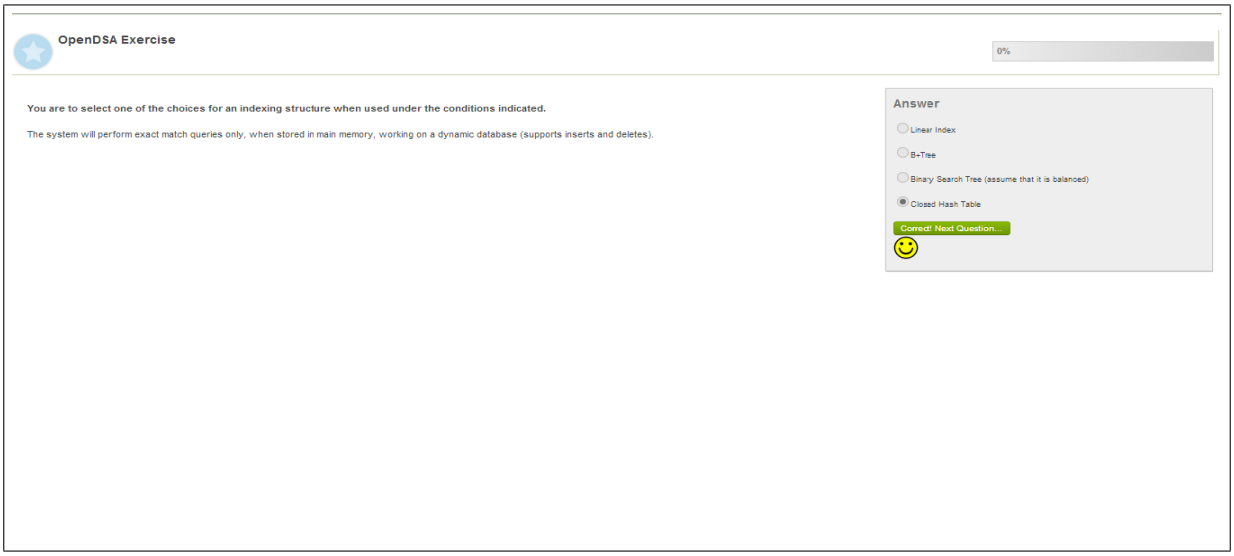


Figure 6.4: The Exported Khan Academy Exercise Showing a List Type Dynamic Multiple Choice Problem.

3. The last example is a Khan Academy exercise written using the *tool-specific* interface of QBank, shown in Figure 6.6. This problem uses scripts written in JavaScript to render an interactive algorithm visualization exercise. Also, the various components of the problem require the markup to be explicitly specified according to the Khan Academy Exercise format. Figure 6.5 displays the authored Khan Academy exercise in a web browser.

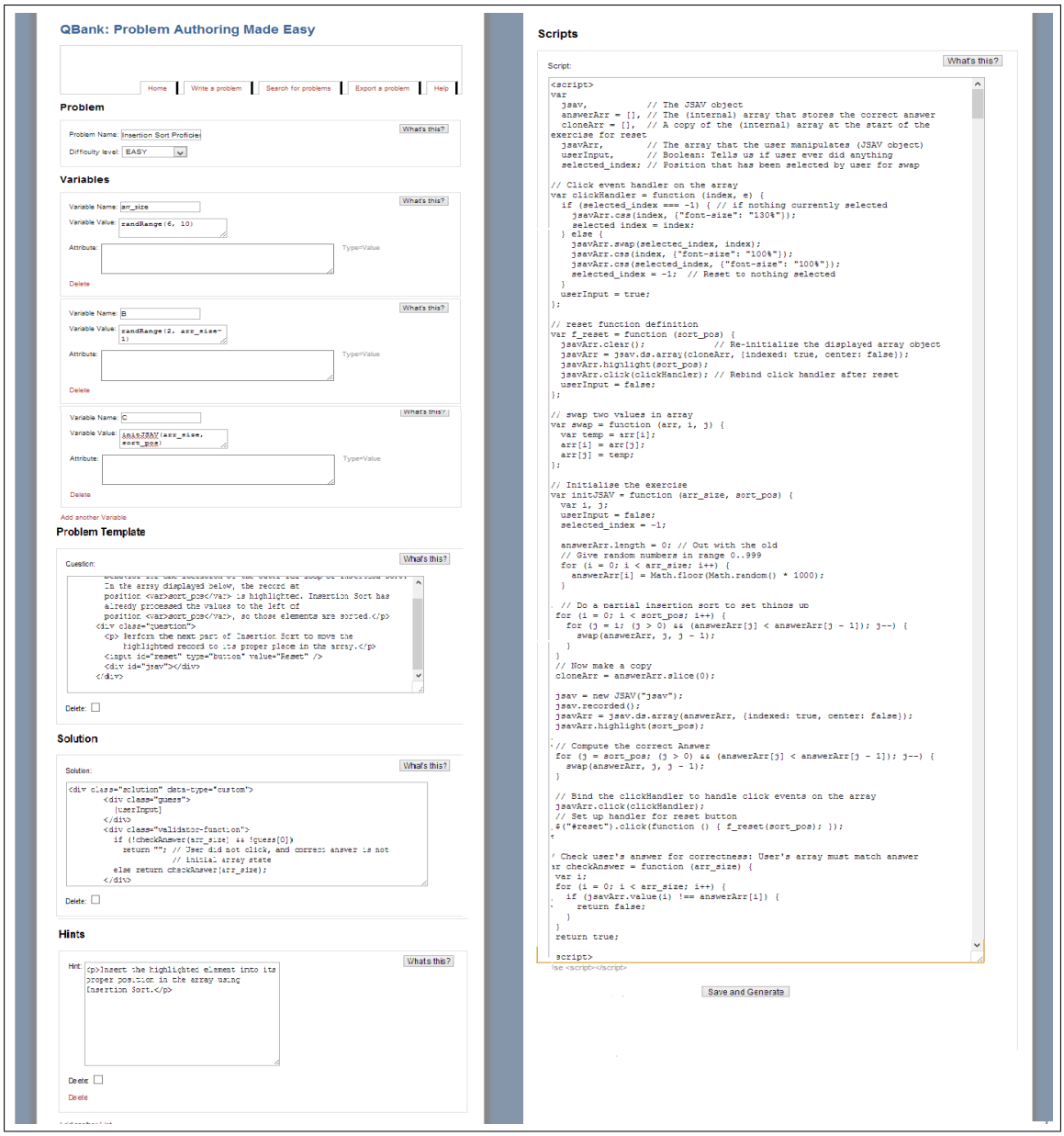


Figure 6.5: Writing a Tool-specific Interactive Khan Academy Exercise for Insertion Sort in QBank.

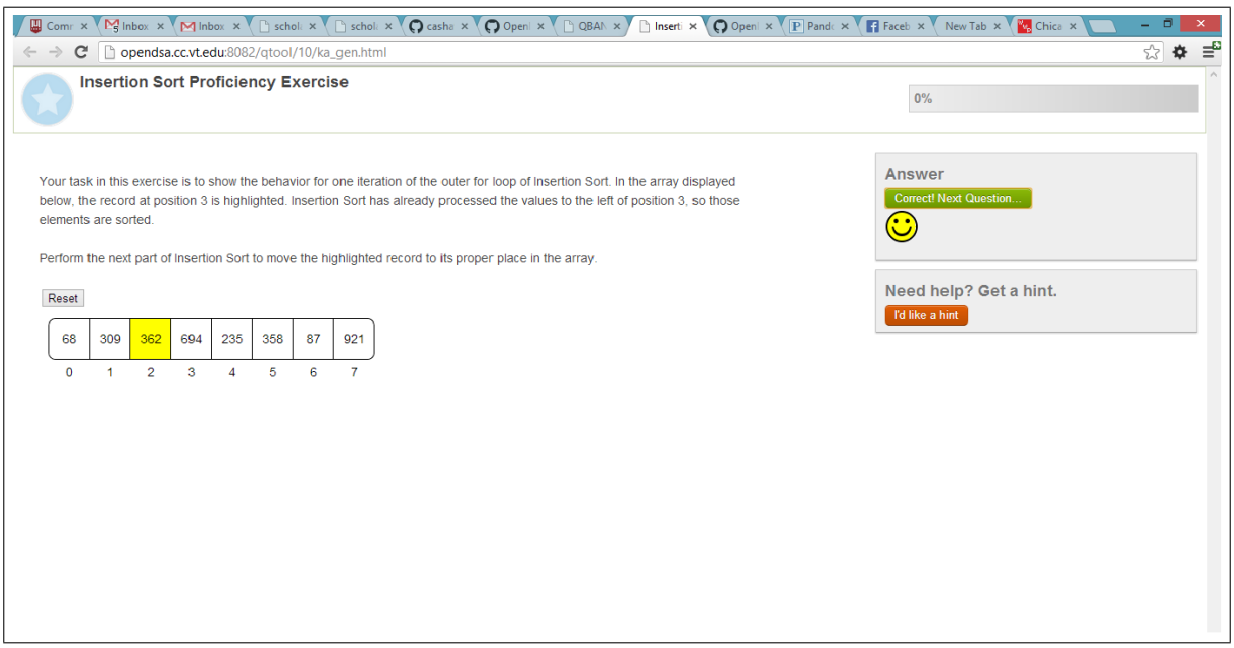


Figure 6.6: Viewing the Interactive Khan Academy Exercise for Insertion Sort in a Web Browser.

6.5 Summary

The usability study indicated that an author can only use the *tool-specific* user interface if they know the tool well enough to author questions in the format specified by that tool. The participants experienced in authoring questions using the Khan Academy Exercise format found the *tool-specific* user interface in QBank helpful since it assists the author by abstracting the formatting information common to problems. QBank offers a defined structure for authoring the tool-specific problem, and allows them to export the authored problem in a more generic (i.e., CSV) format.

Additionally, QBank could be used by experienced authors who would find useful the convenience that it offers for some common problem types but at the same time be able to author more complex questions using the tool-specific interface. Dynamic problem authoring using the *parameterized question* interface was appreciated for being intuitive and useful.

Chapter 7

Conclusion and Future Work

This chapter summarizes the main contributions of QBank and discusses some ideas for future work.

7.1 Contributions of this Work

This thesis makes three contributions:

1. A literature review of question authoring systems.
2. A formal definition for a problem and its solution.
3. A prototype problem authoring system, QBank, that supports dynamic (or parameterized) problem types.

7.2 Future Work

The work associated with developing QBank and the analysis of the usability studies helped to identify areas for further improvement and suggestions for future work. The following changes or additions could improve and extend QBank.

1. **Consolidation of interfaces**

   The need for separate interfaces for dynamic questions was clearly explained in Section 4.2.1. Research can be done for developing a user interface design that consolidates the problem type specific user interfaces together without defeating the purpose of ease of use that separate interfaces provide. Additionally, the developer must ensure that

the consolidation would not clutter the interface or render it overwhelming/confusing for a users with any level of expertise.

2. **User accounts**

   Currently, QBank shares the same database of users authenticated to use OpenDSA. This restricts access to certain functionality like editing and deleting questions which requires the user to be logged in. An additional feature that can be added is a user view where the logged user can see the problems they authored separately from other questions in QBank. This would allow users to keep track of their problems but also have access to all the other problems in QBank for reuse, etc.

3. **Rich textarea Editor**

   Incorporating a rich text editor within the textareas in the interface with paragraph alignment, highlighting, fonts and sizes would enhance the authoring experience.

4. **Version Control**

   This functionality would enable users to work on questions and add complexity to their problems with the option of revert back to an earlier version of their problem. This would encourage users to explore, for instance, the features offered by Khan Academy, or to author complex questions by working their way from simple problems and adding functions to it.

5. **Intelligent syntactic suggestions for tool-specific interface**

   To improve the user experience in the tool-specific interface, function (or code) completion and syntax suggestions can be helpful. This will show the user the different available functions that can be used. Since it is an interface tailored for a particular exercise format (e.g. Khan Academy), the editor can be enriched for the syntax specific to the format.

Bibliography

[1] 20 best learning management systems – Statistics. http://edudemic.com/2012/10/the-20-best-learning-management-systems/, 2013.

[2] Document on Authoring Tools - Definition and Specifications, 2013. http://www.w3.org/TR/ATAG20/#def-Authoring-Tool.

[3] How do i create SCORM content, 2013. http://doc.claroline.net/en/index.php/How_do_I_create_SCORM_content.

[4] IMS global learning consortium website, 2013. http://www.imsglobal.org/background.html.

[5] Questionmark live CSV import, 2013. https://live.questionmark.com/kbase/WhatIsTheCSVImportFileFormat.

[6] F. Miller A. Schwartz, A. Asch. Articulate online website, 2013. http://www.articulate.com/.

[7] Khan Academy. Khan Academy exercise infrastructure. https://github.com/Khan/khan-exercises, 2013.

[8] P. Black and D. Wiliam. Developing the theory of formative assessment. *Educational Assessment, Evaluation & Accountability*, 21:5–31, 2009.

[9] M. Bridges, M. DiPietro, M. Lovett, and M. Norman. *How Learning Works: Seven Research-Based Principles for Smart Teaching*. Wiley, 2010.

[10] T. Naps C. Shaffer, V. Karavirta. OpenDSA website. http://algoviz.org/OpenDSA, 2013.

[11] R. Castaneda. The impact of computer-based simulation within an instructional sequence on learner performance in a web-based environment. *PhD diss., Arizona State University, Tempe*, 2008. http://gsehd.gwu.edu/documents/gsehd/resources/gwuohs-onlineresources/reports/doe_evaluationonlinelearning-092010.pdf.

[12] Martin Dougiamas. Moodle CSV format for multiple choice question import, 2013. https://tracker.moodle.org/browse/CONTRIB-2446.

[13] Martin Dougiamas. Moodle website, 2013. https://moodle.org/.

[14] eXeLearning. eXe - The eLearning XHTML editor, 2013. http://www.http://exelearning.org/.

[15] Sakai Foundation. Sakai project, Online education portal for Collaboration, 2013. http://www.sakaiproject.org/.

[16] D. Furcy. JHAVEPOP website. http://jhave.org/jhavepop, 2013.

[17] J. W. Gikandi, D. Morrow, and N. E. Davis. Online formative assessment in higher education: A review of the literature. *Computers & Education*, 57:5–31, 2011.

[18] A.J. Hilbelink. The effectiveness and user perception of 3-dimensional digital human anatomy in an online undergraduate anatomy laboratory. *PhD diss. dissertation, University of South Florida, Orlando.*, 2007.

[19] IMS Global Learning Consortium, Inc. IMS Question & Test Interoperability: An Overview Final Specification Version 1.2, 2013. http://www.imsglobal.org/question/qtiv1p2/imsqti_oviewv1p2.html.

[20] Blackboard Inc. Blackboard learn, 2013. http://www.blackboard.com/Platforms/Learn/Products/Blackboard-Learn.aspx.

[21] Blackboard Inc. Blackboard learning management system, 2013. http://www.blackboard.com/.

[22] Blackboard Inc. Blackboard Managed Hosting, 2013. http://www.blackboard.com/Services/Managed-Hosting.aspx.

[23] Advanced Distributed Learning (ADL) Initiative. SCORM 2004 4th Edition Version 1.1 Overview, 2013. http://www.adlnet.gov/.

[24] Salman Khan. Khan Academy Exercise Framework Documentation, 2013. https://khan-exercises.readthedocs.org/en/latest/#writing-exercises.

[25] Salman Khan. Khan Academy website. http://khanacademy.org, 2013.

[26] A. Korhonen, L. Malmi, P. Silvasti, J. Nikander, P. Tenhunen, P. Mrd, H. Salonen, and V. Karavirta. TRAKLA2. http://www.cs.hut.fi/Research/TRAKLA2/, 2003.

[27] A. N. Kumar. Problets website. http://www.problets.org, 2013.

[28] Learning Machines Lab. QuadBase website. http://quadbase.org, 2013.

[29] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, September 2004.

[30] H. Maryami, K. Maryam, and K. Hoda. A review on authoring tools. In *5th International Conference on Distance Learning and Education (IPSIT)*, volume 12, pages 45–49, 2011.

[31] U. Preet N. Borg, J. O'Hara. Edmodo website, 2013. `http://www.edmodo.com/`.

[32] A. Ng and D. Kolle. Coursera website, 2013. `https://www.coursera.org/`.

[33] Nick Parlante. CodingBat. `http://codingbat.com/`, 2013.

[34] Ann Paul. Qbank Developer Manual, 2013. `http://algoviz.org/OpenDSA/dev/OpenDSA/Doc/manual/QBankDevManual.html`.

[35] C.A. Shaffer, V. Karavirta, A. Korhonen, and T.L. Naps. OpenDSA: Beginning a community hypertextbook project. In *Proceedings of the Eleventh Koli Calling International Conference on Computing Education Research*, pages 112–117, Koli National Park, Finland, November 2011.

[36] C.A. Shaffer, T.L. Naps, and E. Fouh. Truly interactive textbooks for computer science education. In *Proceedings of the Sixth Program Visualization Workshop*, pages 97–103, Darmstadt, Germany, June 2011.

[37] Eric Shepherd. Questionmark website, 2013. `https://www.questionmark.com/us/Pages/default.aspx`.

[38] Adobe Systems. Adobe Captivate website, 2013. `http://www.adobe.com/products/captivate.html`.

[39] Udacity.com. Udacity website, 2013. `https://www.udacity.com/`.

[40] Mark Urban-Lurain. Intelligent Tutoring Systems. 1996. `http://www.cse.msu.edu/rgroups/cse101/ITS/its.htm`.

[41] WebSoft. CourseLab website, 2013. `http://www.courselab.com/view_doc.html?mode=home`.

[42] Russ Wilde. Evaluating digital authoring tools. *The International Review of Research in Open and Distance Learning*, 5(2), 2004. `http://www.irrodl.org/index.php/irrodl/article/viewArticle/178/260`.