The Canterbury QuestionBank: Building a Repository of Multiple-Choice CS1 and CS2 Questions

Kate Sanders
Rhode Island College
Providence, RI USA
ksanders@ric.edu

Marzieh Ahmadzadeh
Shiraz Univ. of Technology
Shiraz, Iran
ahmadzadeh@sutech.ac.ir

Tony Clear
Auckland Univ. of Technology
Auckland, New Zealand
tony.clear@aut.ac.nz

Stephen H. Edwards
Virginia Inst. of Technology
Blacksburg, VA USA
edwards@cs.vt.edu

Mikey Goldweber
Xavier University
Cincinnati, Ohio USA
mikeyg@cs.xu.edu

Chris Johnson
Univ. of Wisconsin, Eau Claire
Eau Claire, WI USA
johnch@uwec.edu

Raymond Lister
Univ. of Technology, Sydney
Sydney, Australia
raymond@it.uts.edu.au

Robert McCartney
University of Connecticut
Storrs, CT USA
robert@engr.uconn.edu

Elizabeth Patitsas
University of Toronto
Toronto ON Canada
patitsas@cs.toronto.edu

Jaime Spacco
Knox College
Galesburg, IL USA
jspacco@knox.edu

ABSTRACT

In this paper, we report on an ITiCSE-13 Working Group that developed a set of 654 multiple-choice questions on CS1 and CS2 topics, the Canterbury QuestionBank. We describe the questions, the metadata we investigated, and some preliminary investigations of possible research uses of the QuestionBank. The QuestionBank is publicly available as a repository for computing education instructors and researchers.

1. INTRODUCTION

Many computing educators find themselves wishing there existed repositories where they could find questions to copy or adapt for assessments. A resource comparable to mathematics textbooks, with their numerous end-of-chapter questions, only more extensive than an individual book, might be of value to many instructors. Computing education researchers might also find such a resource useful. For example, it could provide a testbed for exploring research questions such as "What kinds of questions do computing educators ask?" and "How well does my new classification scheme work?"

ITICSE'13 Working Group Reports, June 29–July 3, 2013, Canterbury, UK.

Teaching repositories for computing exist, but they have not become popular with instructors (see Section 2.3). One explanation is that most do not have a lot of content. While copying or adapting materials from a repository has its obvious attractions, there is little motivation for most academics to put exam questions into such repositories.

This paper is the report of an ITiCSE 2013 Working Group, which aimed to produce an easily searchable body of high quality multiple-choice questions (MCQs) on first-year computer-science topics. Instead of building the repository infrastructure first, we are focusing on content.

The key questions we wanted to explore were:

- whether we could collaboratively develop such a body of questions,
- whether we could reliably assign various types of metadata to the questions,
- how well software designed to support student collaboration would support the collaborative development of a question bank, and
- whether a substantial dataset of MCQs has the potential to support the exploration of research questions.

In this paper, we describe the Canterbury QuestionBank[1] - the body of questions we have produced so far, the indexing schemes we have considered, the way in which we plan to make the QuestionBank available, some preliminary investigations of how the collection might be used for research, and our plans for the future of the collection. First, we start with some background on repositories.

[1]named after Canterbury, England, where the working group met.

2. BACKGROUND

There have been computing repository initiatives in both teaching and research. They face overlapping but not identical issues. Shared issues include, for example, determining who should have access and how, acquiring content, getting users, and ensuring that the data in the repository are maintained.

In addition, choosing metadata and assigning it to particular items is a significant challenge. Any information other than the data itself, such as the author, date, or type of data – anything that either helps the user to find what he or she wants or makes the item more useful when it is found – is known as "metadata."

In the following subsections, we give background on teaching and research repositories in computing, on metadata, on why repositories have succeeded and failed, and on the software we used in the process of building the Canterbury QuestionBank.

2.1 Teaching repositories

Repository initiatives to share computer science educational resources are at least as old as the ITiCSE conference itself. The first such published proposal was at the first ITiCSE conference in 1996, for a software/courseware repository [10]. The stored entities were to be software artifacts (i.e., courseware) used to support computer science education. Submissions, in a fashion similar to scholarly research, were to be subject to peer review prior to inclusion.

By 1997, the SIGCSE community had begun implementation of a similar resource for storing laboratory assignments, the SIGCSE Computing Laboratory Repository. The Lab Repository also stressed the importance of peer review. The initial phase – in which labs were reviewed by a single gatekeeper – was felt to be insufficient, and plans were made for a more formal journal-like peer review system [14, 17]. In addition, the Lab Repository outlined a categorization scheme to aid repository users in their searches. This scheme was limited to a cover sheet with required fields (e.g., Content area, Level, Topics) allowing submitters to provide free-form text. Other specialized collections such as the Algorithm Visualization repository [1] were being developed at the same time.

By 1998, the Lab Repository had evolved into one of the forming partners of the Computer Science Teaching Center (CSTC) [18]. The CSTC was designed to merge various more-focused initiatives: the SIGCSE Computing Laboratory Repository, the Algorithm Visualization collection, and efforts to build a digital library. CSTC resources were to include peer reviewed courseware, presentations, small assignments, large assignments, and on-line courses. Additionally, the CSTC was to contain various non-peer reviewed resources, such as syllabi, text lists, and a letters-to-the-editor section with teaching advice [11].

By 2002, the CSTC had been absorbed by CITIDEL, which was part of the National Science Digital Library (NSDL), an ambitious project funded by the US National Science Foundation [19]. CITIDEL broadened the scope of its collection to include almost any type of resource, from course syllabi to resources for K-12 educators to material supporting a computer science virtual history museum. Resources in CITIDEL were no longer peer-reviewed.

By 2010, CITIDEL was absorbed into the Ensemble Computing Portal. The Ensemble Computing Portal is now the NSDL access point for various computing education resources [2]. Like CITIDEL, it aims to bring all specialized collections under its umbrella, along with software (such as Violet (`http://horstmann.com/violet/`) and Piazza (`https://piazza.com`)) that can be used to support computing education. In addition, it includes "communities" (discussion groups on various topics).

2.2 Research repositories

DCER, a data repository for computing education research, was proposed in 2008 [28]. Next, Simon et al. [32] and Morrison et al. [25], working with a dataset of 76 data structures exams from 14 different institutions around the world over a 36-year period, demonstrated that it is possible for researchers to obtain interesting results from others' data. The Canterbury QuestionBank could support the investigation of similar research questions about what topics educators assess and how.

In addition, DCER and research repositories in other fields provide insights into the repository building process:

Motivating contributors As noted in Section 2.1, some teaching repositories have sought to motivate contributors by making contribution comparable to writing a journal article. This has proven to be more of an obstacle than an incentive. Research repositories suggest some alternatives.

First, journals and funding agencies can be influential. Some biology journals, for example, require data to be submitted to a repository before articles based on those data can be published. In the United States, funding sources such as the National Science Foundation require data storage plans in grant applications.

Second, if contributors know their work will be cited, that can be a powerful motivator. Each research dataset is associated with the name of a relevant publication, which is then cited in any scholarly work based on that dataset.

Third, access to the data is itself a motivator. Anyone who wants access can be required (or encouraged) to contribute his or her own data.

Attracting users Community matters. The users and contributors of research data typically belong to the same research community. They may have already met or corresponded. There are mechanisms in place, such as conferences, for building and maintaining the community.

2.3 Creating a successful repository

Two recent papers have addressed the questions of why computing teaching repositories have failed [24] and succeeded [8]. The first paper, by Mitchell and Lutters, focused on users. Based on a survey of 119 faculty in the United States and Canada, they concluded that faculty were often unaware of the online repositories available, and to the extent that they had tried them, they were generally dissatisfied. "None indicated that the use of a repository always met their needs, 9% indicated their needs were almost always met, 62% indicated that they were sometimes met, 24% indicated that they were seldom met, and 5% indicated that they were never met" [24, page 4]. These users' failure to use repositories was not because they weren't searching

for new course materials. On the contrary, they frequently looked for such materials, most often using a simple web search.

Asked what were the most important features in an online repository, they placed "Easy to locate materials" first, closely followed by "Materials can be trusted," "Materials available at no cost," "Materials at appropriate level," and "Copyright usage explained." The other eight features, including teaching tips and feedback from other users, were ranked much lower.

Fincher et al. start by proposing an overall framework for teaching repositories [8]. All repositories, according to this framework, must address the following issues:

Control Who can contribute data, who can use it, and how is this enforced?

Contributors How are people motivated to contribute? How are they rewarded?

Catalogue How will users find what they are looking for in the repository?

Community Is there a community of contributors? Of users? How are these communities built and sustained, and what roles do they play?

Curation How is the data maintained?

Content Are the items in the collection focused on a specific topic or more general purpose? Is the content based on a pre-existing collection or created specifically for the repository (or both)?

While the article focuses on teaching repositories, these issues apply equally to research repositories.

The teaching repositories discussed in Section 2.1 face these issues, some implicitly and some explicitly. The question of how to get **contributors** is discussed repeatedly, and various incentives have been used. Peer review was seen as a way of giving recognition for the work involved in making a contribution, but may have just posed an additional hurdle for contributors. One theory is that university faculty are not motivated to help repositories succeed (i.e., submit a resource), because creating and submitting teaching materials generally does not count toward tenure; publications count. For pre-tertiary instructors, on the other hand, success in teaching is paramount. This leads to them being stakeholders in the success of repositories focused on them.

This theory explains why some faculty fail to contribute resources, but not all. It disregards the fact that, at least in the United States, many faculty work at institutions where teaching is valued more highly than research, or two-year community colleges, where research doesn't count at all. Moreover, there are some faculty at research institutions who already have tenure, and even some who do not, who conscientiously invest substantial amounts of time in teaching. Why don't these faculty contribute their resources to the teaching repositories? One possible explanation has to do with the lack of interest shown by *users*, as found by Mitchell and Lutters. It is hard to be motivated to contribute to a repository that no one will use.

Fincher et al. then describe two successful teaching repositories, the Nifty Assignments collection and the Greenroom. Both have a significant number of users, and the Nifty Assignments collection has been in use since 1999. Both are relatively focused – the Nifty Assignments collection on programming assignments and the Greenroom on course materials for use with Greenfoot. But they address the six issues listed above in different ways. For example, Nifty Assignments keeps very tight control over who can submit – submissions are accepted and peer reviewed once a year only – but allows anyone to view the submissions. In the Greenroom, on the other hand, contributors and users are part of the same community. They must be interviewed and demonstrate that they are computing educators before they can use the materials. But anyone who is a user can contribute, and more than that, anyone can *edit* any of the materials in the repository. Both provide guarantees of quality, but in different ways. Nifty Assignments uses a traditional peer review mechanism; the Greenroom allows users to modify and improve any of the resources, and gives credit for doing so.

2.4 Catalogue: defining and using metadata

One of our questions was whether we could reliably assign metadata to the items in the QuestionBank. We hypothesized that this might be easier for items of small granularity such as MCQs.

One possible method is to extract the metadata automatically. Thompson et al. [33] experimented with automatically extracting metadata from educational resources on the web. Classification was fairly accurate; information extraction was more challenging. Tungare et al. [34] proposed to build on this work, using a large collection of computing course syllabi, gathered for inclusion in CITIDEL. While an interesting challenge for machine learning, and somewhat feasible for assignments or syllabi, this is not likely a promising approach for short MCQs.

Another possibility is to have the contributors assign metadata from a predefined list, possibly allowing optional extra tags. Edwards et al. [7] take this approach in their proposed design for a repository of programming assignments. Their metadata were carefully designed to balance the value of information to the user against the burden on the contributor. Required metadata included only the title (of the assignment), abstract, programming language (in which the program was to be written), author, date (when submitted), language (the natural language of the abstract, etc.), copyright, and license (the name of the license that governs the use of the assignment, e.g., Creative Commons). Recommended metadata included (among other things) topics, prerequisite knowledge, and other prerequisite assignments that are also in the repository. Optional metadata included (among other things) difficulty, estimated time the students would need, and any citations to relevant materials.

Like Edwards et al., we chose to have the contributors assign metadata to the items in the Canterbury QuestionBank. In addition, since we had content for the QuestionBank, we were able to examine the reliability of this tagging. Edwards et al. were unable to investigate reliability, as that project was still in the design phase. Their required metadata – items such as author, title, date, and the original programming language of the assignment – would likely have high reliability, but the recommended and optional data include items such as topic and difficulty level that are less objective.

2.5 Software support for contributors

Unwittingly, by deciding to initially create content instead of repository infrastructure, we set for ourselves a recursive

problem: what environment would we use to collect and store our content so that we could ourselves search and analyze its contents? The solution we settled on was to use PeerWise to store our questions. PeerWise [6] is a web-based tool that has been very successful in enabling students in a class to create, review, and answer each other's MCQs concerning the course material. It is designed to support "Contributing Student Pedagogy," defined in [12] as: "A pedagogy that encourages students to contribute to the learning of others and to value the contributions of others."

PeerWise promised significant benefits – it is designed to support precisely what we were doing, collaborative development of MCQs. Moreover, Paul Denny, one of the authors of the software, helped us to set up the site and even coded modifications to support our work.

Still, it was clear that we would be pushing the limits of the software. PeerWise was designed for short lists of tags, and for students entering a small number of questions at a time; our working group would be using it with the goal of writing 100 questions each and tagging each one with 15 different tags, one of which had 72 possible values.

3. METHODOLOGY

The working group members generated a set of questions and assigned a large variety of different tags to them: some fairly standard tags and others, more novel, whose application might be considered as research questions. The process of generating and tagging the questions is described in this section.

3.1 Generating and entering the data

The data for this working group consist of the following:

1. Multiple-choice questions appropriate for a CS1 or CS2 course.
2. For each question:
   (a) The correct answer;
   (b) An explanation of why the answer is correct, sometimes including explanations why alternatives are not correct;
   (c) Quality and difficulty ratings from question reviewers;
   (d) A set of category values ("tags") that describe this question;
   (e) Optional comments and responses to comments from author and reviewers.

These were collected from the working-group participants using PeerWise [6] in the weeks before the group convened. The tasks performed by the participants were as follows.

Entering the questions Each author entered questions into PeerWise via a web form. Each question has 2-5 alternative answers, plus an explanation. The web form provided support for text (typed in or cut-and-paste), formatting using an HTML subset and interactive LaTeX editor, and importing images from files. Notably, there was no support for uploading files with multiple questions, nor support for pasting in images.

Tagging the questions The contributors tagged their own questions, entering values for 15 different categories (described in detail in Section 3.2). Each category except for Topic was assigned at most one value; the Topic category allowed multiple values.

Answering/rating the questions Each of the working group participants answered a subset of the questions submitted. For each of these, they answered the questions, and verified that they agreed (or disagreed) with the question's author. They then rated the question in terms of difficulty (easy, medium, hard), and quality (poor, satisfactory, excellent).

Comments on questions Whenever someone answered a question they had the option of adding a comment. Question authors were informed by PeerWise when any of their questions were commented upon. Authors and question answerers could also comment on the comments.

Once the questions, tags, and comments were added to PeerWise, the participants used PeerWise during the meetings in Canterbury to access questions or groups of questions based on category values.

3.2 Categories and Tags

As noted above, the members of the working group tagged their questions. This was done relative to 15 categories, each of which had a number of possible tag values. We were particularly interested in languages and categories for talking about exam questions, so we included categories and tags from a wide variety of schemes proposed in the literature. In the remainder of this section, we describe each of the categories and its tags.

3.2.1 *Contributor*

This category, like the next two, provides straightforward information about the question, in this case, the name of the author.

3.2.2 *CS course*

This category gives the course for which the question would be appropriate; possible values are CS1, CS2 and CS-Other.

3.2.3 *Language*

The language category describes what computer language is used in the question. Its possible values are Alice, C, C++, C#, Java, Perl, Python, Scheme, VB, and none.

3.2.4 *Multiple Answers*

PeerWise requires that the contributor designate a single correct answer for each question. This requirement can be a problem for peer instruction questions, which sometimes have more than one correct answer. Within PeerWise, it is necessary to select one of the correct answers, and then designate in the explanation (provided with each question) which other answers are also acceptable.

To provide additional support for peer instruction questions, we added this category. Its tags are true and false, true if two or more of the alternative answers to a question are correct. The intent was to enable us to identify and examine these questions and allow future users to search for them.

3.2.5 *Nested Block Depth*

This category is the maximum nested block depth of the conditionals and loops in any code contained in the question.

Possible values for the maximum nested depth are 0, 1, 2, 3, and 4 or more.

This value is a measure of the complexity of the question (or at least the code contained in it): for nested conditionals, the number of possible paths increases with depth, and for loops, the total number of operations is proportional to the product of the number of times through each loop. This is similar to the use of Average Block Depth as a code metric in Kasto and Whalley [15].

3.2.6 Abstraction Transition Taxonomy (ATT)

The Abstraction Transition Taxonomy [5] was developed by analyzing a number of multiple choice questions used in a CS0 class, then applied to questions from a number of different sources. It has two categories: transition level, which is based on the abstraction levels in the question, and purpose, which relates to what the student is asked to provide.

Transition level describes the relation between the language abstraction levels of the question and the answer. The language abstraction levels considered are English, a natural language description, CS-Speak, a description using technical language with meaning specific to computer science, and Code, a description expressed in a programming language. The category values reflect these transitions: English to CS-Speak, English to Code, CS-Speak to English, CS-Speak to Code, Code to English, and English to Code. For questions that are asked and answered at the same abstraction level, they defined ApplyCode (e.g. tracing code to get a variable value), Apply CS-Speak (e.g. what CS constructs are used in a given algorithm), and Define CS-Speak (e.g. reflect on what some CS construct provides). **Purpose** refers to the question, and has two possible values, How and Why.

Our use of this taxonomy has two purposes: first, to investigate whether the tags would be useful as metadata, and second, to explore how the QuestionBank might be used as a research repository. Testing a novel classification scheme such as ATT is one way in which the QuestionBank could be used for research.

3.2.7 Block model

We also used tags based on Schulte's Block Model [29]. The Block Model is a model of program comprehension, originally developed to help student teachers design lesson plans about computing. It has recently been used to classify short-answer program-comprehension questions from a CS1 exam [36]. The authors conclude, "The main advantage of the Block model is that it provides us with a way of describing these novice programming tasks that gives us a level of granularity which allows us to distinguish between similar tasks in a way that SOLO or Bloom's taxonomy cannot." [36, page 73]

As with the ATT Taxonomy, our use of the Block Model has two purposes: to investigate whether the tags would be useful as metadata, and also to illustrate how the QuestionBank could be used as a research repository, by providing data with which to test the Block Model. While the Block Model is not so new as ATT, the QuestionBank still enables us to apply the Block Model to both a new question format (MCQ) and a wider variety of topics.

The Block Model includes four "scopes" at which a program can be examined and three ways in which each scope can be considered. These are summarized in Table 1. The rows indicate scope, from atoms (or statements), to blocks, to relationships between blocks, to the entire program. Each column is a way of comprehending a piece of code: on the left, as static text; in the middle, in terms of dynamic behavior; and on the right, in terms of its purpose.

	Text surface	**Program execution (data flow and control flow)**	**Functions (as means or as purpose), goals of the program**
Macro structure	Understanding the overall structure of the program text	Understanding the algorithm of the program	Understanding the purpose of the program
Relations	References between blocks, e.g. method calls, object creation, accessing data, etc.	Sequence of method calls	Understanding how subgoals are related to goals
Blocks	Understanding the text of a block	Understanding the operation of a block	Understanding the purpose of a block
Atoms	Language elements	Understanding the operation of a statement	Understanding the purpose of a statement

Table 1: Schulte's Block Model

Thus, each column is a hierarchy. So for example, consider the left-hand column. At the most basic level (the bottom cell) the student understands the text of an atom or statement; above that, the text of a block; then text that involves relations between blocks; and finally the overall structure of a program. Similarly, the middle column is a hierarchy from understanding the behavior of a statement to understanding the behavior of blocks, relationships between blocks, and entire programs, and the right-hand column captures the purpose of code, from the purpose of an individual statement up to the purpose of the entire program.

3.2.8 Bloom taxonomy

The contributors also tagged questions using Bloom's hierarchy. This taxonomy, drawn from education research, has been widely discussed in computing education research for some years [3, 21, 9]. Although there is disagreement about how well it fits computing topics [9], it is well known and has been used in the past to tag questions.

The possible values are Knowledge, Comprehension, Analysis, Application, Synthesis, and Evaluation. Briefly, Knowledge refers to the ability to recall basic information; Comprehension is the ability to explain individual pieces of information; Application is the ability to use abstract knowledge in specific situations; Analysis is the ability to break something down into parts and understand the relationships of the parts; Synthesis is the ability to put the parts together; and Evaluation is the ability to discuss whether something is fit for a particular purpose. Before tagging, contributors

completed an online tutorial on the Bloom hierarchy (available at `www.progoss.com`).

The remaining categories are all drawn from a paper by Simon et al. [31] and the expanded version in a Working Paper, also by Simon et al. [30]. Some of these, like Topic, are natural metadata; others are more exploratory. They are discussed in alphabetical order.

3.2.9 Code Length

Code Length is a set of three categories (low, medium, high) based on the number of lines of code in a question.

In the original, the definition was somewhat imprecise:

- low: up to about half a dozen lines of code
- medium: between half a dozen and two dozen lines of code
- high: more than two dozen lines of code

We translated that definition into the following tags. The overlap between low and medium is a deliberate attempt to reproduce the imprecision of the original:

- 0-6 lines (low)
- 6-24 lines (medium)
- 25 lines and over (high).

Besides these three, we added one more category, so the scheme would also work for questions that do not involve explicit code:

- Not applicable (i.e. no code),

3.2.10 Conceptual Complexity

Conceptual complexity, the second of the Simon et al. categories [31], is a category used to measure the types and amount of interaction between concepts that are required to answer the question, evaluated as low, medium, or high.

3.2.11 Difficulty

This is an estimate of how difficult the average student would find this question at the end of the course. The values are Low, Medium, and High. This was Degree of Difficulty in Simon et al. [31].

3.2.12 External Domain References

This category expresses the degree to which a question refers to things outside of the domain of the course that could be problematic for some students, such as cultural references. Its possible values are Low, Medium, and High, and it is a generalization of the two-valued Cultural References category from Simon et al. [31].

3.2.13 Linguistic Complexity

Linguistic Complexity [31] describes the sophistication, complexity, and length of the question (in terms of reading and understanding, not the concepts the question engenders). Unlike Code Length, there is no objective measure. Possible values are Low, Medium, and High.

3.2.14 Skill

Skill identifies the primary skill required to be able to answer a question. The possible values are: Analyze Code, Debug Code, Design Program Without Coding, Explain Code, Modify Code, Pure Knowledge Recall, Test Program, Trace Code (includes expressions), Write Code (means choose option here). These are essentially the same as in [31] with the following changes:

- We added Analyze Code;
- We added "includes expressions" to Trace Code so that it could handle the evaluation of complex expressions as well as longer pieces of code;
- We added "without coding" to Design Program;
- We added the comment "means choose option" to Design Program to reflect the fact that all of our questions are multiple choice.

3.2.15 Topic

Unlike most categories, this one allows up to three topics to be specified. The list of topics in Simon et al. [31] included: Algorithm Complexity-Big-Oh, Arithmetic Operators, Arrays, Assignment, ClassLibraries, Collections Except Array, Constants, Data Types And Variables, Events, Exception Handling, FileIO, GUI Design or Implementation, IO, Lifetime, Logical Operators, Loops (subsumes Operators), Methods Functions Procedures, Notional Machine, OO concepts, Operator Overloading, Parameters (subsumes Methods), Program Design, Programming Standards, Recursion, Relational Operators, Scope-Visibility, Selection(subsumes Operators), Strings, and Testing.

We added significantly to the list in Simon et al. [31]. First, we added Class Libraries and Exception Handling from the fuller description of this scheme in [30]. Second, since Simon et al.'s lists were focused only on CS1 (Introductory Programming), we added numerous topics related to CS2 (Data Structures). Finally, we added some topics as needed during the tagging process. The full list of topics is given in Appendix A.

4. RESULTS

We collected 654 questions. Most of the questions were tagged by their contributors using the scheme outlined in Section 3. Due to time limitations, some questions were tagged using only some of the categories, and some were not tagged. If we had been planning to use the tags immediately as indices for users to search on, the incomplete tagging would have been problematic. Since our initial goal was to investigate the various tagging schemes, the substantial number of tagged questions that we did have were sufficient.

In the following subsections, we describe the distribution of the tags as assigned by their contributors, followed by analyses of inter-rater reliability for key tags, followed by some observations about our question development process.

4.1 Distribution of tag values

One way to characterize the questions is by examining the distribution of the tags. The data below are based on the tags assigned by the person contributing the question.

4.1.1 Abstraction Transition Taxonomy

There are two ATT classifications, transitions and types. Types include only How and Why. Transitions include Code to CS-Speak, CS-Speak to Code, Code to English, English to Code, CS-Speak to English, and English to CS-Speak, as well as the related (but non-transitional) categories Apply CS-Speak, Apply Code, and Define Code. English to English

Transition tags ($n = 475$)		
tag	*number*	*percent*
Apply CS-Speak	164	35%
Apply Code	142	30%
Code to CS-Speak	72	15%
CS-Speak to Code	30	6%
Code to English	25	5%
CS-Speak to English	14	3%
English to Code	12	3%
Define CS-Speak	8	2%
English to CS-Speak	7	1%
English to English	1	0%

Question Type tags ($n = 457$)		
tag	*number*	*percent*
How	426	93%
Why	31	7%

Table 2: Number and percentage of ATT Classifications in the QuestionBank. For each tag type, n is the number of questions with a single value chosen.

Block Model tags ($n = 444$)						
	Text		**Control**		**Goal**	
Macro	9	(2%)	48	(11%)	18	(4%)
Relation	11	(2%)	21	(5%)	6	(1%)
Block	72	(16%)	158	(36%)	8	(2%)
Atom	64	(14%)	26	(6%)	3	(1%)

Table 3: Number and percentages of Block Model classifications in the QuestionBank

was added here for completeness. It was not used in [5], and it was only used once in this dataset.

Our results are shown in Table 2. The transition tags are dominated by two of the non-transitional categories, Apply CS-Speak and Apply Code, which account for nearly two-thirds of the questions. Looking at it from another perspective, 62% of the questions go to or from CS-Speak, 59% go to or from code, and 12% go to or from English.

The type tags are almost all How questions (93%). This is consistent with the results reported in [5]: they investigated several datasets of CS assessments and the percentage of Why questions did not exceed 15% in any of them. They provide a possible explanation:

> *How questions* are a norm in computing education. The *why questions* are a natural outgrowth of making explicit the need to rationalize, culturally, decisions of activity and tool use in programming problems [5, p. 67].

4.1.2 Block Model

444 of the questions were tagged with exactly one Block Model row and exactly one column. The distribution of these tags among the cells of the Block Model is shown in Table 3.

As can be seen in Table 3, the MCQs have been rated predominantly at the Block level, with Block Control as the most commonly rated pattern, followed by Block Text. The former fits with the intuitive notion that for CS1 and CS2 course related MCQs the primary focus would be on blocks of code, either in terms of their static code or of their dynamic behavior. The next largest category of questions is at the more atomic level of Atom and Text, which aligns with MCQs aimed at demonstrating knowledge of basic, discrete elements or concepts.

This table demonstrates the relative absence of questions with a focus on the macro or relations levels or on goals. This is consistent with the focus on "how" questions rather than "why" questions in the ATT questions discussed previously. This may result from an inherent limitation in the range of possibilities that can be expressed via MCQs. Alternatively, as noted in [4], it may reflect a lack of focus in our teaching on fully linking the hermeneutic dimensions of the Block Model in our teaching and learning activities. Perhaps more positively for CS2 questions in this repository, there seems to be some emphasis on the questions at the macro (whole program) level, with 11% of questions tagged in the macro/control block. These questions typically involved evaluating the student's ability to select or apply an appropriate algorithm to meet a defined goal.

4.1.3 Bloom

The results from the 505 questions that were tagged with exactly one Bloom category are given in Table 4.

Bloom levels ($n = 505$)		
level	*number*	*percent*
Knowledge	116	23%
Comprehension	185	36%
Application	18	4%
Analysis	165	32%
Synthesis	21	4%
Evaluation	5	1%

Table 4: Number and percentage of Bloom levels in the QuestionBank for those questions with a single value chosen.

Among these questions, the most popular categories (in decreasing order) were Comprehension, Analysis, and Knowledge. These three accounted for over 90% of the questions. There were relatively few examples of Application, Synthesis, or Evaluation.

4.1.4 Appropriate course

The contributors also indicated whether the question was appropriate for CS1, CS2, or some other CS course. Contributors selected these tags independently. Generally, CS2 included Big-Oh analysis and data structures (lists, stacks, trees, queues, etc.); CS1 included everything else.

Our results are given in Table 5 for the 512 questions that were placed in exactly one of these categories.

Appropriate course ($n = 512$)		
course	*number*	*percent*
CS 1	270	53%
CS 2	222	43%
Other	20	4%

Table 5: Number and percentage of appropriate course values in the QuestionBank for those questions with a single value chosen.

In addition, 7 questions (not included in the table) were tagged as belonging to both CS1 and CS2.

4.1.5 Code length

Table 6 presents the results for the 509 questions were tagged with exactly one of the possible code length tags.

Code length ($n = 509$)		
range	*number*	*percent*
Low (0-6 lines)	155	30%
Medium (6-24 lines	162	32%
High (> 24 lines)	9	9%
Not Applicable	183	36%

Table 6: Number and percentage of code-length values in the QuestionBank for those questions with a single value chosen.

Interestingly, the tagging was able to work despite the ambiguities. Some questions with no code may have been tagged "0-6," but perhaps not, as the "Not Applicable" tag was the most popular overall. In addition, there were no questions tagged both Low and Medium, as would have been appropriate for a code length of 6.

4.1.6 Conceptual complexity

Only 333 questions were tagged with exactly one of the possible conceptual complexity tags; the results are in Table 7.

Conceptual complexity ($n = 333$)		
level	*number*	*percent*
Low	154	46%
Medium	174	52%
High	5	2%

Table 7: Number and percentage of conceptual complexity values in the QuestionBank for those questions with a single value chosen.

4.1.7 Difficulty

Question authors also tagged their questions for difficulty, identifying one of three difficulty levels: low, medium, or high. This tag represented the author/instructor's view of how difficult the question would be for a student to answer correctly at the point of taking the final exam for the corresponding CS course. The results are shown in Table 8.

Difficulty ($n = 520$)		
level	*number*	*percent*
Low	313	60%
Medium	187	36%
High	20	4%

Table 8: Number and percentage of difficulty values in the QuestionBank for those questions with a single value chosen.

4.1.8 External Domain References

424 questions were categorized according to the level of external domain references involved. The results are given in Table 9. Interestingly, the contributors did not tag any questions as having "High" external domain references. This may be characteristic of the particular questions in the QuestionBank, or MCQs in general, or something that other raters might disagree with.

External domain references ($n = 520$)		
level	*number*	*percent*
Low	391	92%
Medium	33	8%
High	0	0%

Table 9: Number and percentage of external domain references values in the QuestionBank for those questions with a single value chosen.

4.1.9 Languages

522 questions were tagged by the contributors as relating to exactly one programming language (or none). These results are presented in Table 10.

Languages ($n = 522$)		
Language	*number*	*percent*
Alice	0	0%
C	82	16%
C++	1	0%
C#	0	0%
Java	232	44%
Perl	9	2%
Python	36	7%
Visual Basic	9	2%
none	153	29%

Table 10: Number and percentage of language values in the QuestionBank for those questions with a single value chosen.

Two questions were tagged with two languages each. One was about the memory model of C and C++, and the other was about the exception mechanism in Java and C#. Tagging two (or more) languages was quite rare, given that quite a few questions were ambiguous (e.g. any C question that did not explicitly identify the language could also have been C++); this is likely because the contributor knew which language his or her class was using.

4.1.10 Linguistic complexity

428 questions were tagged for their level of linguistic complexity, as shown in Table 11. The great majority of these were tagged as low linguistic complexity.

4.1.11 Multiple answers

Only 8 questions were tagged as having multiple answers. This may be accurate – not all the contributors were interested in peer instruction questions, and not all peer instruction questions have more than one right answer. They just need to have more than one *plausible* answer. Because of the way the category was written, however, we could not distinguish between questions that were untagged and ques-

Linguistic complexity ($n = 428$)		
level	*number*	*percent*
Low	391	91%
Medium	34	8%
High	3	1%

Table 11: Number and percentage of linguistic complexity values in the QuestionBank for those questions with a single value chosen.

tions where the contributor considered Multiple Answers to be false.

4.1.12 Nested Block Depth

424 questions were tagged with exactly one tag in this group. The results are given in Table 12.

Nested block depth ($n = 424$)		
tag	*number*	*percent*
No conditionals or loops	268	63%
One nested	51	12%
Two nested	95	22%
Three nested	9	2%
Four-or-more nested	1	0%

Table 12: Number and percentage of Nested block depth values in the QuestionBank for those questions with a single value chosen.

Most questions did not contain any conditionals or loops. Of those that did, it is somewhat interesting that a block depth of 2 is more common than a block depth of 1, perhaps reflecting a preference for doubly-nested loops or conditional in a loop code questions.

4.1.13 Skills

452 questions were tagged by their contributors with exactly one skill each. The results are given in Table 13.

Skills ($n = 452$)		
tag	*number*	*percent*
Trace (includes Expressions)	161	36%
Pure Knowledge Recall	116	26%
Write Code (means Choose Option)	56	12%
Analyze Code	41	9%
Debug Code	24	5%
Design Program Without Coding	21	5%
Explain Code	21	5%
Test Program	11	2%
Modify Code	1	0%

Table 13: Number and percentage of Skill values in the QuestionBank for those questions with a single value chosen.

The top popularity of Trace can be partially explained by the fact that this tag was given to questions that evaluate an expression as well as those that require more extensive tracing to solve.

4.1.14 Topics

468 questions were tagged by the contributors for topics with one, two, or three tags total. In this group, of the 72 possible topics, 54 were used at least once. The results for all of the topics that were used for at least 5 questions (i.e. over 1%) are given in Table 14.

Topics ($n = 468$)		
tag	*number*	*percent*
Loops - Subsumes Operators	76	16%
Arrays	67	14%
Data Types And Variables	62	13%
Algorithm Complex Big-Oh	56	12%
Methods Funcs Procs	55	12%
Collections Except Array	49	10%
Strings	40	9%
Selection - Subsumes Operators	32	7%
Arithmetic Operators	25	5%
OO concepts	25	5%
Params - Subsumes Methods	24	5%
Assignment	22	5%
Logical Operators	21	4%
Hashing-HashTables	20	4%
Program Design	18	4%
Trees-Other	17	4%
Sorting-Quadratic	16	3%
Linked Lists	16	3%
Recursion	16	3%
Testing	15	3%
Choosing Appropriate DS	14	3%
Sorting-NlogN	12	3%
Heaps	12	3%
Scope-Visibility	11	2%
ADT-List-DefInterfaceUse	10	2%
I/O	10	2%
Relational Operators	9	2%
Graphs	9	2%
Pointers (But Not References)	8	2%
Sorting-Other	8	2%
File I/O	8	2%
ADT-Stack-DefInterfaceUse	7	1%
Searching-Binary	6	1%
ADT-Stack-Implementations	6	1%
Runtime-Storage Management	5	1%
Trees-Search-Not Balanced	5	1%
GUI-Design-Implementation	5	1%

Table 14: Number and percentage (of questions) with given topic values in the QuestionBank for those questions with 1, 2, or 3 topics chosen. Only topics that show up in at least 5 questions are included in table.

4.2 Inter-Rater Reliability

We performed a number of inter-rater reliability (IRR) tests for key categories: ATT, Block Model, Bloom level, Difficulty, and Skills. These were all done on subsets of the data using a stratified sampling approach based on the contributor's tags, with the goal that each tag was represented an equal number of times in the sample. The measures used

for each were dependent in part on the measurement scale of the tagging data. We then looked at Topic and Language with a stratified sample based on the contributor.

4.2.1 Abstraction Transition Taxonomy

To estimate the IRR for the ATT-transition tags, one researcher selected 28 questions which were subsequently tagged by eight researchers. These tags were used to calculate Fleiss' Kappa, with resultant Kappa = 0.3 (z = 20; $p < 0.05$); a fair level of agreement [20].

Examining how the group used the ATT tags, most of the disagreement came from what "English" means and where the boundary lies between English and CS-Speak. There was also disagreement about the meanings of Apply CS-Speak and Define CS-Speak. Other unreliability came largely from a disagreement about what should be done about numeric responses. Consider a question showing code, and asking, what will the value of x be after the end of this code, with A = 3, B = 4, etc. Our eventual consensus was that this would be Apply Code, but initially there were people in the group who tagged this as Code to CS-Speak, because they saw numbers as being CS-Speak rather than Code.

4.2.2 Block model

Two researchers chose 36 questions, three for each cell in the model. These were then tagged using the Block model by the other 7 raters. We first analyzed the inter-rater reliability for the twelve cells in the table assuming that they were twelve distinct nominal categories. Using Fleiss' Kappa, the result is 0.191, which is at the high end of slight agreement, but not quite to the threshold for fair agreement.

We then looked at the rows and columns. Grouping the tags in each row together and using Fleiss's Kappa, we get an inter-rater reliability of 0.357 for the rows in the table (atom, block, relational, macro). Grouping the tags in each column together, we get a Fleiss's Kappa of 0.175 for the columns (text, control, function). The agreement on columns is slight; the agreement on rows is fair.

However, the measurement scale of these data is not nominal, so Fleiss' Kappa is not appropriate. The rows correspond to increasing size, while the columns progress from syntax to semantics to purpose, so Block Model data can be seen as ordinal in two dimensions. As far as we know, there is no formula for inter-rater reliability of 2D ordinal tags, so we looked for an appropriate two-dimensional distance metric.

We considered using Manhattan distance summed over all pairs of raters as a distance metric for each observation, but it is not clear how this set of distances should be combined, and what to compare it with. We observed a large range of distances, from 0 (all raters agree) to 54, which is close to the theoretical maximum of 60.

4.2.3 Bloom levels

In similar fashion, one researcher chose 42 questions that were approximately evenly split among the six Bloom categories, based on how the author tagged the question.[2] These were tagged for Bloom level by eight raters. Using Fleiss's Kappa for 8 raters with 42 subjects, the result was Kappa = 0.189 (z = 11.7, p = 0.912), which is a slight level of agreement.

[2]An even split was impossible, since some categories (such as Evaluation) were used for fewer than 7 questions.

While Bloom's taxonomy is categorical, the measurement level is more appropriately treated as ordinal, ranging from 1 (Knowledge) to 6 (Evaluation), as the categories are ordered. It follows, for example, that Knowledge (1) is "closer" to Comprehension (2) than to Synthesis (5), so a rater who tags a question as Knowledge shows more agreement with another rater who tags the same question as Comprehension than with a rater who considers the question Evaluation. Fleiss's Kappa is inappropriate for ordinal data.

We used intraclass correlation (ICC) for both agreement (i.e. do some raters show a bias towards one end of the spectrum when compared to other raters?) and consistency (i.e. do raters tend to put the same question on the same part of the spectrum?).

First, to investigate whether there were measurable differences in rating bias among raters – that is, whether some raters tended to assign questions to lower or higher Bloom levels compared to other raters – we calculated the ICC for agreement (ICC_a) to be $ICC_a = 0.231$, suggesting that there is a tendency for some raters to give lower ratings on average, while others give higher ratings on average. As with any correlation coefficient, a maximal value of 1.0 would indicate that the average ratings given by each individual rater were exactly the same, while a minimum value of 0.0 would indicate they were completely unrelated.

Second, to investigate how consistently raters rated the same questions, we also calculated the ICC for consistency (ICC_c) to be $ICC_c = 0.267$, $p < 0.001$. Again, this indicates a generally poor level of agreement, where a maximum value of 1.0 would indicate raters always agreed on any given question, and a minimum value of 0.0 would indicate their ratings on any given question were completely unrelated.

Thus, even with a training tutorial, our raters obtained only poor levels of agreement in applying Bloom's taxonomy to a set of 42 questions. This is consistent with the difficulties identified in earlier research on the use of the Bloom hierarchy in computing education [9].

4.2.4 Difficulty

To investigate interrater reliability for this tag, one working group member randomly selected seven questions tagged with each difficulty level, for a total of 21 questions. The eight other members of the working group then independently rated these 21 questions for difficulty. As with Bloom level, the data here are on an ordinal measurement scale, so we again used intraclass correlation to measure both agreement and consistency.

Examining the question of differences in rating bias among raters – that is, whether some raters tended to rate questions easier or harder compared to other raters – we calculated the ICC for agreement to be $ICC_a = 0.427$, $p < 0.0001$. This value indicates less than moderate agreement among raters, a tendency for some raters to give lower ratings on average, while others give higher ratings on average.

Examining how consistently raters rated the same questions, we also calculated the ICC for consistency to be $ICC_c = 0.471$, $p < 0.0001$. Again, this indicates less than moderate agreement.

Taken together, these ICC values indicate that, while there was some level of agreement and consistency, there was no evidence for strong agreement or high consistency when rating question difficulty. As a result, while instructor-perceived difficulty may be a worthwhile piece of metadata to use de-

scriptively on questions, it may have much less utility as a search/filtering criterion when others wish to locate questions within the collection.

4.2.5 Skills

To explore the inter-rater reliability of these tags, one researcher selected 22 questions. Six other working group members then placed those questions in categories indicating which skills were involved. We computed Fleiss's Kappa of 0.426 for 6 raters and 21 subjects, which indicates moderate agreement.

4.2.6 Topics

Measuring the agreement on Topics is also complex: a rater can provide 1-3 tags per observation, and there are 72 different topic tags. In an attempt to measure the reliability, two raters tagged a random set of questions stratified by contributor that included 5% of each contributor's questions, for a total of 33 questions. The total number of topic assignments was calculated as the sum of the number of topics mentioned for each question: if the two raters assigned the same topic to a given question, that was counted as one topic assignment. If only one rater assigned a particular topic to a given question, that was also counted as one topic assignment. Using this definition, there were 86 topic assignments, 2.6 topics per question (on average). Of these, the raters agreed on 52 assignments, or 60%.

4.2.7 Language

While testing for topics, the two raters also looked at Language, tagging the same questions sampled with Topics for Language. The tagging and discussion raised two related issues, especially if a question is restricted to one language tag value:

- Some code snippets are syntactically legal in more than one language (for example, C, C++, and Java).
- Code snippets have to be identified by someone who knows the language in question – if a rater is unfamiliar with a language it would seem that "Don't know" would be the appropriate tag.

To properly consider reliability it would be necessary to define what constitutes a match in both of these cases.

4.3 Software Support

PeerWise gave us a platform designed specifically for the collaborative development of MCQs. All collaborators could see each other's questions as soon as they were entered.

There were some features that we would recommend if PeerWise is to be used extensively for this purpose.

- Support for uploading files with multiple questions. This is probably not important for student users of PeerWise, who are not likely to enter 100 questions at a sitting, and probably don't have files of MCQs that they have already written. For our purposes, however, it would have been a big plus.
- The ability to input any subset of possible tag values and retrieve a page of links to questions with those tags, for example, asking questions such as "Give me all the MCQs we have so far that are at the Evaluation level in the Bloom hierarchy," or "Give me any questions that are at the Evaluation level in Bloom and the Macro level in Block."
- The ability to download all the questions in a format that could easily be uploaded into a Course Management System such as Moodle or Sakai.
- Support for adding UML and tree diagrams. Including images and diagrams was possible, but difficult.

With those additions, the software would not only work well during the development phase of the repository, but would have the potential to work in the production phase when the data is made more broadly available.

5. PATTERNS

The number of potential MCQ questions for introductory computing is clearly infinite. Nevertheless, in our analysis of the question database, a number of similarities in question type/style emerged. We called these MCQ patterns.

Each pattern discerned asks students to demonstrate a specific skill (e.g. trace code, big-Oh calculation). We originally called our patterns "parameterized questions." For example, a question that presents the code for selection sort in Java and asks students to identify the name of the algorithm could be simply modified to present insertion sort in Python.

Upon further examination we discovered that our categorizations were more abstract than parameterized questions, since questions from the same pattern might differ not only in terms of "parameters," but also in terms of syntax/semantics focus, expression, and quality factor:

- Syntax or Semantics: Perhaps because the QuestionBank is limited to introductory computing, both syntax and semantic/purpose related questions emerged.
- Expression: This relates to the style of how a question is posed. For example, a question can be posed in the negative.
- Quality Factor: Questions had different foci with regard to the concept being questioned about. Some questions focused on performance, while for others, the quality factor was superior (object oriented) design.

While two questions drawn from the same pattern might appear very different, they nonetheless probe, in a similar manner, a student's understanding. For example, the question given above – given the code for selection sort in Java, identify the name of the algorithm – fits our Purpose MCQ Pattern. The Purpose MCQ Pattern might be used to query a student's ability to recognize the selection sort algorithm (in one's language of choice), or the insertion sort algorithm, or merge sort, or find largest, or depth-first search, etc. Specifically,

- a question might present students with code and ask which algorithm it is; or
- a question might present the algorithm's name and present the student with different code snippets; or
- a question might present the algorithm's name and require the student to identify from a set of code snippets which does not represent a correct implementation of the algorithm; or

- a question might present an algorithm and ask a student for which, of a variety of scenarios, the algorithm is appropriate (or not appropriate).

In a related work, Hazzan et al. [13] usefully outline 12 different types of questions one may pose. They do not claim that these are an exhaustive set, but they do provide useful patterns for consideration. Their description of type differs significantly from our recognition of MCQ patterns. For example one type of question described in [13] is to have students provide a solution (code, pseudocode, or descriptive algorithm) to a given problem. This problem type is considered an *open* question and not suitable for an MCQ, which is a *closed* question.

Of the 12 question types, Hazzan et.al. assert only 5 are suitable for MCQs. They are:

- Code tracing;
- Analysis of code execution;
- Finding the purpose of a given solution;
- Examination of the correctness of a given solution;
- Efficiency estimation.

Furthermore, they indicate three more question types that cannot naturally be presented as an MCQ:

- Completion of a given solution;
- Instruction manipulation;
- Programming style questions.

For purposes of completion the four question types which it is claimed cannot be presented as an MCQ are:

- Development of a solution;
- Development of a solution that uses a given module;
- Question design;
- Transformation of a solution.

5.1 The 12 MCQ Patterns

Four working group members independently examined distinct subsets of the question bank, identifying a pattern for each question. We then compared the resulting patterns, and quickly merged our individual patterns into 12 distinct MCQ patterns. We make no claims of completeness for our enumeration; furthermore, another set of researchers, upon examining the same questions, may discern a different set of patterns (e.g. 11 or 13).

We were struck by the ease with which we were able to combine the patterns independently extracted by four different researchers into our library of 12 final patterns. We believe that our patterns identify underlying structure that helps partition MCQs in a useful manner. However, we must note that it could be the case that the MCQs from which we extracted our patterns were particularly narrow in scope, and analysis of another bank of questions might lead to a larger, or different, set of patterns. We also did not attempt to categorize a subset of our MCQs using our pattern library, so we cannot comment on the difficulty of applying the patterns consistently. Nonetheless, we are confident that our set of 12 patterns, if not exhaustive, is sufficient to categorize the vast majority of MCQs.

For each pattern, we present a name, a short description, a sample question(s) and which of the Hazzan types it most closely matches.

The 12 MCQ patterns, in no particular order, as follows:

5.1.1 Compare and Contrast

A compare and contrast question will involve comparing and contrasting two particular pieces of code, algorithms, data structures, or the like.

Sample question:

> `What is the difference between selection sort and insertion sort with respect to worst case algorithm efficiency?`

Compare and contrast questions can examine a variety of quality factors such as memory usage, speed, style, usability, etc.

Questions can be generic, as in the sample, or more specific, as in:

> `If you are using quicksort on <list>, which partitioning method is preferred?`

This pattern supports a large variety of expression differentiation. Consider:

> `Which of the following pairs of code statements are not equivalent?`

which illustrates how negation can change the problem. Furthermore, a compare and contrast question can be flipped, as in

> `Which of the following statements properly compares and contrasts` *algorithm A* `with` *algorithm B*`?`

One can also compare and contrast syntactic or semantic elements of code. A question asking to compare `a++` and `a = a+1` in Java would be asking to compare a syntactic element. In contrast, comparing two pieces of code in terms of purpose or performance would be considered semantic.

Hazzan type: Comparing and contrast style questions are not captured in the Hazzan question type enumeration. Hence, a compare and contrast question might be, depending on the particulars, either a *code trace, analysis of code execution, examination of the correctness of a given solution, or efficiency estimation*

5.1.2 Fixed-Code Debugging

A fixed-code debugging question typically presents a fixed piece of code that contains a syntactic or a semantic error. The student is then asked to identify the line containing the error, or choose from a list of reasons why the code is wrong.

This MCQ pattern allows for either an examination of semantic or syntactic elements. For semantic errors, the purpose of the code is usually supplied. Semantic errors can include logic errors (e.g. using != instead of ==), or possibly misunderstood class/language elements (e.g. call by value vs call by reference)

Sample question (syntactic):

> `A compiler error exists in this code. Why is that happening?`

```
public class Foo {
    int fooCount;
    public static int getCount(){
       return fooCount;
    }
    public Foo(){
        fooCount ++;
    }
}
```

Hazzan type: *Code tracing*, and *Analysis of code execution*.

5.1.3 Tracing of Fixed Code

Fixed code is a question pattern that "requires the student to hand execute (trace through) some code and select, from a set of provided answers, the correct outcome or result." [23]

Questions of this type are the canonical hand execution of code; "what is generated by a `print` statement somewhere in a code snippet."

Hazzan type: *Code tracing*.

5.1.4 Algorithm and Data Structure Tracing

This pattern differs from the above tracing of fixed code, in that these questions operate at a higher conceptual level; that of algorithms and data structures, instead of individual lines of code. This pattern includes questions that require students to trace the execution of a particular algorithm on a given instance of a data structure. Often, questions of this type contain an illustration that pictorially represents the state of a data structure.

Sample question:

> `Given the above binary tree rooted at Node A, what is the order of nodes visited by an in-order traversal?`

While not exactly requiring students to "trace" an execution sequence, recognizing whether a given data structure instance is in a legal state falls under this pattern.

Sample question:

> `Which of the following is not a legal priority queue?`

Questions such as this not only test student understanding of a given data structure's invariant, but may also require students to construct a legal code sequence that would generate the provided data structure instance.

Sample question:

> `Suppose you have a binary search tree with no left children. Duplicate keys are not allowed. Which of the following explains how this tree may have ended up this way?`

Hazzan type: *Code tracing*.

5.1.5 Basic Conceptual

A basic conceptual question requires students to recall or apply definitional knowledge. A straight-forward application of this pattern would provide the student with a term/concept and require one to choose from among a set of possible definitions.

Sample question:

> `Which of the following is true of a Java interface?`

Basic conceptual questions allow for expression differentiation. In addition to questions posed in the negative, one might also simply reverse the question format; provide a concept's definition and have students chose from among a set of disciplinary terms.

It is even possible to construct *application* style questions which require students to use their understanding of a concept in a new context. The concept may or may not even be explicitly mentioned.

Sample question:

> `In Java, the actual type of a parameter or variable's value can be any concrete class that is:`

Hazzan type: none.

5.1.6 Basic Procedural

Like its counterpart, the basic conceptual pattern, this pattern requires students to recall or apply basic knowledge, though in this case the knowledge is procedural or algorithmic.

Sample question:

> `Which of the following sorting algorithms are` *stable*`?`

Sample question:

> `Which of the following data structures would best model the line of waiting customers at a ticket counter?`

The conceptual unit for investigation is not limited to algorithms or data structures. Alternatively, one might deal with programming patterns.

Sample question:

> `The Examine-all programming pattern is best described as...`

Hazzan type: *Development of a solution*.

5.1.7 Procedural Flexibility

Procedural flexibility, also known simply as flexibility [16], refers to knowledge of different ways of solving problems and when to use them. Examples include generating multiple methods to solve a problem, recognizing multiple methods to solve a problem, using multiple methods to solve a problem, and evaluating new methods [26].

Sample question:

> `In the hash table illustrated above, given the list of values and hash function, which collision resolution was used?`

This MCQ question pattern differs from the Basic Conceptual pattern in that students are required to apply multiple procedures and evaluate which would produce a particular result.

Sample question:

> `Given these values, which hash function will produce no collisions?`

Hazzan type: The two closest would be *Examination of the correctness of a given solution* and *Instruction manipulation*, neither of which really fully capture the multiple evaluation aspect of this MCQ pattern.

5.1.8 Synthesis

The synthesis pattern involves analyzing a problem and then synthesizing something from the analysis. This was the least common pattern in our question bank given the difficulty in writing a synthesis MCQ.

Sample question:

```
What is the size of the largest BST that is
also a min-heap?
```

For this question a student must analyze both BST and min-heap properties, synthesize a tree that has both properties – and grow it to its maximum size.

Hazzan type: none.

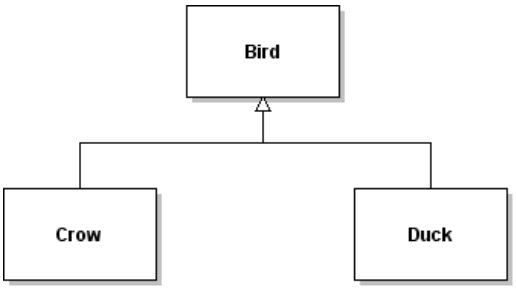
5.1.9 Design

Almost all of the design-related questions in our question bank focused on Object-Oriented (OO) design, although we can imagine design questions that focus on procedural decomposition, refactoring, or changes in representation (i.e. changing from UML to code, or from a textual description of a design to UML, etc). Design questions can also focus on the quality, readability, or maintainability of the code, although such questions must be carefully written to fit the MCQ format. Syntactic variants of these questions are likely to require an understanding of language-level features, while semantic questions focus on higher-level design questions that do not rely on specific features of the language.

Sample question:

```
Suppose you are defining a Java ProductItem
class to store information about the inventory
in a store, and you want to give the class
an instance variable for the price of that
item.  Which of the following is the best datatype
for the instance variable that models the price?
```

Sample question:




```
The simplified UML diagram above shows the
relationships among Java classes Bird, Crow,
and Duck.  Suppose Bird has a fly(Location
place) method, but we want Crows to makeNoise()
just before they take off and then behave like
other Birds.  Assuming Crows have a makeNoise()
method, the fly method in Crow should consist
of...
```

Hazzan type: *Completion of a given solution*, *Development of a solution*, and possibly even *Programming style questions*.

5.1.10 Skeleton Code

Skeleton code is a category of question that "requires the selection of the correct code, from a set of provided answers, which completes the provided skeleton code." [23]

Hazzan type: *Completion of a given solution*

5.1.11 Big-Oh

Computational complexity is a subject that is well-suited to the MCQ format. Crossing numerous algorithms with the handful of complexity classes produces a large set of potential questions. These questions are often expressed in two forms:

1. given an algorithm, select its complexity class and
2. given a task, select an algorithm or data structure that completes the task with the desired performance.

In the former case, students examine the code to inspect the algorithm and determine its complexity. In the latter case, students reason about the problem and the efficiency of its implementation.

Many questions involve Big-Oh analysis, but that is certainly not a requirement. Complexity may be expressed as the number of iterations, comparisons, memory transactions, subroutine calls, and other measures of performance.

Sample question:

```
Suppose you have a Java array of ints.  Which
of the following operations can be performed
in constant (O(1)) time?  (Circle all correct
answers.)
```

Sample question:

```
Two algorithms accomplish the same task on
a collection of N items.  Algorithm A performs
N^2 operations.  Algorithm B performs 10N operations.
Under what conditions does algorithm A offer
better performance?
```

Hazzan type: *Efficiency estimation.*

5.1.12 Purpose

Code purpose (explain in plain English) is a category of question identified in [35], which requires a student "given a code segment, to explain the purpose of that piece of code." Variants on this might include: name a method, identify pre- and post-conditions for a segment of code, name the algorithm, or write comments or javadocs for a piece of code.

Hazzan type: *Finding the purpose of a given solution.*

5.2 Using patterns

The goal of our pattern discovery exercise is that we believe that it can aid in the creation of new MCQs. Given the above examples and those found in the question bank itself, it is hopefully a straightforward exercise for an MCQ author to decide on an MCQ pattern - what kind of concept should the question test (e.g. basic concept, Big-Oh, algorithm tracing). Once the pattern is picked, the author would meld the concept to be tested with the pattern; e.g. algorithm tracing and searching. Finally, questions can be modified or fine tuned based on one of the three orthogonal dimensions; change the question from syntactic to semantic, or maybe pose the question in the negative.

6. MAKING THE QUESTIONS AVAILABLE

While developing this collection of questions has been a learning process, the ultimate goal of this effort is to produce a resource that can be used by others – educators and researchers – in the community. As a result, the working group also devoted time to planning an appropriate way to provide access to the collection, and also to allow contributions to the collection so that it can grow and develop over time.

A side-effect of the goal that the questions be available to others was that we were unable to use questions from copyrighted material such as textbooks and the sample CS AP exam.

6.1 Licensing

One question that faces every group that intends to provide a resource for others is licensing: what rights will others have, and what restrictions will be imposed? Also, while there are many discussions regarding license choices for software projects, the same is not true for other educational resources, such as this question bank.

In choosing a license, the working group wanted to preserve open access for educators who want to use any questions from the question bank in the classroom, but also wanted to protect the question bank from being reused wholesale by publishers or commercial entities who might prefer to provide only closed access. As a result, the working group chose to make the question bank available under the Creative Commons Attribution/Non-commercial/Share-alike 3.0 Unported License (BY-NC-SA 3.0). This license is also the license used by Khan Academy's Exercise Framework for all of its exercises published on github.

To clarify the "share alike" and "attribution" requirements for educators who want to reuse or adapt some of the questions on homework/quizzes/tests/etc., this license includes the following responsibilities:

Attribution If you use any of the questions from the bank in a publicly available resource, you must attribute the work. If you are an individual educator using questions from the bank as part of a homework assignment, test, quiz, or classroom activity that is only available to your students (not publicly available), then attribution is not required.

Noncommercial You may not use this work for commercial purposes.

Share Alike If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one. If you adapt, modify, or enhance any questions for use on a homework assignment, test, quiz, or classroom activity for your course, you can satisfy the "Share Alike" requirement of this license by sending your derived work to questionbank@vt.edu for us to add to the question bank.

6.2 Short term

The most important question in the near term is how to provide access to the questions themselves, to whom, and in what format. While the simple answer seems to be "post them on the web!", the problem is more difficult to address in practice. First, should the question bank be publicly available, along with its answer key? Second, the question of what format to post the questions in is also an issue if the question bank is to be most useful to the intended audience.

6.2.1 Restricting Access to a Qualified Audience?

When the working group began its task, the goal was to provide open access to the resulting question bank so that other educators could use it as freely as possible. However, some educators may feel that posting the question bank complete with answers openly on the web may actually compromise the value of the questions for use on exams, quizzes, or other assessment instruments, even if the questions might still be a good resource for practice or self-study. At the same time, however, requiring educators to obtain a password or otherwise negotiate access to the answer key may pose an obstacle making them less inclined to use the question bank. After discussing this issue, the working group decided that open posting of the complete question bank along with an answer key was preferable. Even if students have access to questions and the answers before an exam, they do not necessarily remember them [22]. In addition, because of the size of the question bank, it would not be practical for students to attempt to memorize answers in advance of a test. Furthermore, if they did actually intensively study the whole set of questions with the hope of preparing for some situation, they might actually learn something!

The QuestionBank is available on-line at: `http://web-cat.org/questionbank`

6.2.2 A Distribution Format (or Six)

While it seems as if the answer to distributing the question bank is just to "post a big file containing all the questions," the real question is what format to use for this information. The questions could be posted in a large text file – but what about questions that include images as part of the question, or even as part of the answer choices? Also, plain text would lose all of the formatting encoded in the questions themselves (fixed-width formatted code blocks, italics and bold, etc.). While questions could instead perhaps be posted in an RTF document or some other form, making this choice involves understanding the primary uses that educators would have for questions from this question bank.

By considering possible uses for these questions, we saw three primary modes that educators would operate in:

1. Searching/browsing: Educators who want to know what questions are available, or want to find questions on a specific topic, or want to find questions they can use as inspiration for their own will all want to read (some of) the questions in the bank.

2. Reusing questions for written instruments: Educators who want to use questions from the bank in writing their own quizzes, tests, or self-study guides that they intend to print will likely want convenient copy-and-paste support into the word processor or text editor they are using.

3. Reusing questions for on-line instruments: Educators who want to use questions from the bank in writing their own quizzes, tests, or self-study guides that they intend to make available on-line will likely want convenient copy-and-paste (or, better yet, electronic import) support into the on-line presentation format or course management system (CMS) they are using.

To address the needs of these groups, and also to best utilize the limited facilities the group has available for translation and presentation of the question bank, we decided to make the question bank available in at least three forms in the short term: as a single PDF, for those who wish to browse or print it that way; as a single HTML document, for those who wish to browse or search it on-line, or who wish to cut-and-paste fully formatted text into a word processor or into an on-line tool; and using a standardized XML-based question format that can be imported into most existing CMS tools or learning management systems (LMS).

While both PDF and HTML formats are understandable alternatives, some working group members considered supporting import to current LMS tools to be an absolute necessity for our group, if we want others to be able to use the question bank. Most LMSes support on-line quizzing with import/export capabilities, and if our MCQs couldn't be used without re-entering the questions by hand, that would seriously limit the value of our group's work to the rest of the community. Unfortunately, most LMS tools appear to have their own individualized export formats, in order to support the full set of features of their own individual online quizzing modules. Not all systems support the same feature set – or even exactly the same styles of questions or question answering mechanisms. That is one reason why there are so many different tool-specific formats for exporting quiz-style questions.

Fortunately, the questions collected by the working group fall within the "least common denominator" of MCQ use among quizzing systems. All are multiple choice questions with 2-5 choices, and all have one designated right answer. (We anticipated that some of the peer instruction questions would have more than one correct answer, but in fact there were very few. Even there, PeerWise requires the author to designate a single correct answer and indicate the other possibilities in a comment.)

For our work to be useful to the widest segment of our target community, the format chosen should work on as many systems as possible. To this end, we chose to provide the question bank in QTI v1.2 format, the IMS Question & Test Interoperability Specification. QTI is a standard XML-based format for question interchange. There are multiple versions of the standard that have been developed over the past 10 years, but the newest versions are not widely supported – in part, because the newest standard covers a superset of all features, and few tools actually provide for everything that is covered. Fortunately, the slightly older v1.2 of the QTI standard appears to be the most commonly supported format among LMS tools, with Blackboard, WebCT, Moodle (via an add-on plug-in), Sakai (aka, Scholar), Desire2Learn, Canvas, eCollege, and Angel Learning Management Suite.

Further, QTI v1.2 is well-supported by Respondus, one of the more widely used commercial tools for preparing and publishing on-line MCQ and quiz/test questions. By providing the question bank in this format, educators will be able to import to most LMS tools, and at the same time they will also be able to use commercial tools like Respondus as the "front end" for viewing/editing/adapting questions and uploading them to quizzing systems. While QTI may not be generally accepted as the standard format for MCQ interchange, it does appear to be the de facto choice that is most widespread today.

After pursuing this course and using Respondus to generate a QTI version of the question bank, it was easy to use Respondus' export capabilities to generate a number of other formats. As a result, the question bank is being provided in 6 formats: Word .doc, PDF, HTML, plain text, QTI, and CSV format.

6.3 Long Term

Over time, the working group will seek out community contributions of questions to increase the coverage of the question bank. Educators who have questions they wish to contribute can e-mail them to questionbank@vt.edu. Questions in any readily accessible format are welcome, although contributors must be willing for their work to be publicly available for others to use under the QuestionBank's BY-NC-SA 3.0 license.

In addition to soliciting contributions, additional future work to provide more innovative services is also possible. For example, it is possible to provide more tool-oriented (rather than human-oriented) question bank access, for example through a RESTful web service that provides structured access to questions in JSON or XML format, including built-in search/filter capabilities.

If you are interested in working on enhancing the collection, contributing questions, or providing different modes of access, please contact the working group.

7. DISCUSSION AND CONCLUSIONS

We have achieved the primary goal of the project: developing a substantial body of MCQs on CS1 and CS2 topics. The questions were written and reviewed by knowledgeable, experienced instructors. While there is undoubtedly room to polish them further (see, for example, the guidelines for writing MCQs given by Piatek in [27]), we are confident that they are good enough to provide a useful resource. Further, the QuestionBank was made available to alpha testers at the end of the summer and (if all goes according to plan) will be released more broadly as a beta version by the end of the year.

Our second question was whether we could reliably add metadata to the questions, and there our answer is mixed. We experimented with a wide variety of metadata for these questions, and in general, could not reliably assign metadata values to the questions. Even after studying the taxonomies closely and (in the case of Bloom) completing training, for almost all the tags we examined, inter-rater reliability was "fair" or worse. For the schemes to work as search indices, contributors would have to tag the questions consistently and users – likely less familiar with the metadata – would have to use them effectively as well.

For now, the QuestionBank is being made availabe in a searchable format, without indices. This may prove to be the best solution – after all, it is the way most of us find things on the Web. But there are some reasons for optimism with regard to the metadata. First, some of the categories were more reliable than others, including the two that (in informal conversations) users indicated they would be most interested in: Skill and Topic. The reliability for Skill was moderate, and the level of agreement for Topic (although difficult to measure) also seemed promising. A shorter, more focused list of topics might improve the level of reliability even further. Second, it is possible that future researchers who focused on a single one of these taxonomies,

might achieve better results. Writing 100 questions and assigning each one a value in 15 different categories may have led to "tagging fatigue."

PeerWise, the software we used to support our collaboration, made it possible for a group located on four different continents to enter and review each other's MCQs. Not surprisingly, the large number of questions entered by each person and the extremely large number of metadata categories and possible tag values pushed the system to its limit. There were a number of features we would like to see added, including the ability to upload multiple questions from a file, support for diagrams, the ability to designate multiple correct answers, and the ability to select and view subsets of the data. Nevertheless, it was a remarkable feat.

The existence of this substantial body of MCQs also opens up the possibility of a variety of research projects. For example, we can use the QuestionBank as a testbed for exam classification schemes. We outlined above preliminary investigations of questions such as whether the ATT Taxonomy can be applied to CS1 and CS2 as well as CS0, and the extent to which the Block Model can be used for MCQs, as well as for understanding code.

We can also ask questions about the questions. Our first attempt at this is described in Section 5, where we present a variety of MCQ patterns that we found in the QuestionBank. Other possible questions include:

- Which tags (if any) are unsuitable for MCQs?
- Which tags (perhaps like How questions) retrieve few questions, but the ones that are there are particularly interesting? And is there some way to generate more of that type of question?
- Which tags correlate with each other, and what can we say about the questions that fall into the intersection of those two tags?

These are all intriguing directions for future work.

Finally, the most important question is whether the repository will be of value to computing instructors and researchers. We will investigate that, and continue to seek ways to make it more useful, in the future.

Acknowledgments

Thanks to Paul Denny for his considerable help with PeerWise, including writing software to assist the working group. We also thank the people outside the working group who contributed questions: Donna Teague, Andrew Luxton Reilly, Phil Robbins, and colleagues at Auckland University of Technology. Support for this research was provided by the Office for Learning and Teaching of the Australian Government Department of Industry, Innovation, Science, Research and Tertiary Education.

8. REFERENCES

[1] AlgoViz.org; the Algorithm Visualization Portal. `http://http://algoviz.org`.

[2] The Ensemble Computing Portal. `http://www.computingportal.org/`. Retrieved August 7, 2013.

[3] D. Buck and D. J. Stucki. Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, SIGCSE-00, pages 75–79, 2000.

[4] T. Clear. The hermeneutics of program comprehension: a 'holey quilt' theory. *ACM Inroads*, 3(2):6–7, 2012.

[5] Q. Cutts, S. Esper, M. Fecho, S. R. Foster, and B. Simon. The Abstraction Transition Taxonomy: developing desired learning outcomes through the lens of situated cognition. In *Proceedings of the Eighth Annual International Conference on International Computing Education Research*, ICER '12, pages 63–70, 2012.

[6] P. Denny, J. Hamer, A. Luxton-Reilly, and H. Purchase. Peerwise: students sharing their multiple choice questions. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 51–58, 2008.

[7] S. H. Edwards, J. Börstler, L. N. Cassel, M. S. Hall, and J. Hollingsworth. Developing a common format for sharing programming assignments. *SIGCSE Bull.*, 40(4):167–182, 2008.

[8] S. Fincher, M. Kölling, I. Utting, N. Brown, and P. Stevens. Repositories of teaching material and communities of use: Nifty Assignments and the Greenroom. In *Proceedings of the Sixth International Workshop on Computing Education Research*, pages 107–114, 2010.

[9] R. Gluga, J. Kay, R. Lister, S. Kleitman, and T. Lever. Over-confidence and confusion in using Bloom for programming fundamentals assessment. In *Proceedings of the Forty-Third ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 147–152, 2012.

[10] M. Goldweber. Proposal for an on-line computer science courseware review. In *Proceedings of the First Conference on Integrating Technology into Computer Science Education*, ITiCSE-96, page 230, 1996.

[11] S. Grissom, D. Knox, E. Copperman, W. Dann, M. Goldweber, J. Hartman, M. Kuittinen, D. Mutchler, and N. Parlante. Developing a digital library of computer science teaching resources. In *Working Group Reports of the Third Annual SIGCSE/SIGCUE ITiCSE Conference on Integrating Technology Into Computer Science Education*, ITiCSE-WGR '98, pages 1–13, 1998.

[12] J. Hamer, Q. Cutts, J. Jackova, A. Luxton-Reilly, R. McCartney, H. Purchase, C. Riedesel, M. Saeli, K. Sanders, and J. Sheard. Contributing student pedagogy. *SIGCSE Bull.*, 40(4):194–212, 2008.

[13] O. Hazzan, T. Lapidot, and N. Ragonis. *Guide to Teaching Computer Science*. Springer, 2011.

[14] D. Joyce, D. Knox, J. Gerhardt-Powals, E. Koffman, W. Kreuzer, C. Laxer, K. Loose, E. Sutinen, and R. A. Whitehurst. Developing laboratories for the SIGCSE Computing Laboratory Repository: guidelines, recommendations, and sample labs. In *The Supplemental Proceedings of the Conference on Integrating Technology Into Computer Science Education: Working Group Reports and Supplemental Proceedings*, ITiCSE-WGR '97, pages 1–12, 1997.

[15] N. Kasto and J. Whalley. Measuring the difficulty of code comprehension tasks using software metrics. In *Proceedings of the Fifteenth Australasian Computing Education Conference*, ACE2013, 2013.

[16] J. Kilpatrick and J. Swafford. *Helping Children Learn*. National Academies Press, 2002.

[17] D. L. Knox. On-line publication of CS laboratories. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97)*, 1997.

[18] D. L. Knox. The Computer Science Teaching Center. *SIGCSE Bull.*, 31(2):22–23, 1999.

[19] D. L. Knox. CITIDEL: making resources available. In *Proceedings of the Seventh Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '02, pages 225–225, 2002.

[20] J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1), March 1977.

[21] R. Lister. Objectives and objective assessment in CS1. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, SIGCSE-01, pages 292–296, 2001.

[22] R. Lister. The Neglected Middle Novice Programmer: reading and writing without abstracting. In *Proceedings of the Twentieth Annual Conference of the National Advisory Committee on Computing Qualifications*, NACCQ-07, pages 133–140, 2007.

[23] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bull.*, 36(4):119–150, 2004.

[24] S. M. Mitchell and W. G. Lutters. Assessing the value of computer science course material repositories. In *Proceedings of the Nineteenth Conference on Software Engineering Education and Training Workshops*, CSEETW '06, pages 2–5, Washington, DC, USA, 2006.

[25] B. B. Morrison, M. Clancy, R. McCartney, B. Richards, and K. Sanders. Applying data structures in exams. In *Proceedings of the Forty-Second ACM Technical Symposium on Computer Science Education*, pages 353–358, 2011.

[26] E. Patitsas, M. Craig, and S. Easterbrook. Comparing and contrasting different algorithms leads to increased student learning. In *Proceedings of the Ninth Annual Conference on International Computing Education Research*, ICER '13, 2013.

[27] M. E. Piontek. Best Practices for Designing and Grading Exams. `http://www.crlt.umich.edu/publinks/CRLT_no24.pdf`.

[28] K. Sanders, B. Richards, J. E. Moström, V. Almstrum, S. Edwards, S. Fincher, K. Gunion, M. Hall, B. Hanks, S. Lonergan, R. McCartney, B. Morrison, J. Spacco, and L. Thomas. DCER: sharing empirical computer science education data. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 137–148, 2008.

[29] C. Schulte. Block model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 149–160, 2008.

[30] Simon, J. Sheard, A. Carbone, D. Chinn, and M.-J. Laakso. A guide to classifying programming examination questions (Working Paper No.2), 2013. `http://hdl.handle.net/1959.13/1036148`. Retrieved August 6, 2013.

[31] Simon, J. Sheard, A. Carbone, D. Chinn, M. Laalso, T. Clear, M. de Raadt, D. D'Souza, R. Lister, A. Philpott, J. Skene, and G. Warburton. Introductory programming: examining the exams. In *Fourteenth Australasian Computing Education Conference*, ACE2012, pages 61–70, 2012. `http://www.crpit.com/confpapers/CRPITV123Simon.pdf`.

[32] B. Simon, M. Clancy, R. McCartney, B. Morrison, B. Richards, and K. Sanders. Making sense of data structures exams. In *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER-10, pages 97–106, 2010.

[33] C. A. Thompson, J. Smarr, H. Nguyen, and C. Manning. Finding educational resources on the web: exploiting automatic extraction of metadata. In *Proceedings of the ECML Workshop on Adaptive Text Extraction and Mining*, 2003.

[34] M. Tungare, X. Yu, W. Cameron, G. Teng, M. A. Perez-Quinones, L. Cassel, W. Fan, and E. A. Fox. Towards a syllabus repository for computer science courses. *SIGCSE Bull.*, 39(1), 2007.

[35] J. Whalley, T. Clear, P. Robbins, and E. Thompson. Salient elements in novice solutions to code writing problems. In *Proceedings of the Thirteenth Australasian Computing Education Conference*, ACE '11, pages 37–46, 2011.

[36] J. Whalley and N. Kasto. Revisiting models of human conceptualisation in the context of a programming examination. In *ACE 2013*, pages 67–76, 2013.

APPENDIX

Possible values used for Topics

ADT-Dict-DefInterfaceUse
ADT-Dict-Implementations
ADT-List-DefInterfaceUse
ADT-List-Implementations
ADT-Map-DefInterfaceUse
ADT-Map-Implementations
ADT-PriorityQ-DefInterUse
ADT-PriorityQ-Implementations
ADT-Queue-DefInterfaceUse
ADT-Queue-Implementations
ADT-Set-DefInterfaceUse
ADT-Set-Implementations
ADT-Stack-DefInterfaceUse
ADT-Stack-Implementations
Algorithm Complexity-Big-Oh
Arithmetic Operators
Arrays
Assignment
Character-ASCII
Character-Representations
Character-Unicode
Choosing Appropriate DS
Class Libraries
Collections Except Array
Constants
Data Types And Variables
Events
Exception Handling
File I/O
Graphs
GUI-Design-Implementations
Hashing-HashTables
Heaps
I/O
Java Interface
Lifetime
Linked Lists
Logical Operators
Loops Subsumes Operators
Methods Funcs Procs
Notional Machine
Numeric-Float-Precision
Numeric-Float-Range-Overf
Numeric-Float-Representations
Numeric-Float-Rounding
Numeric-Integ-Range
Numeric-Int-Range
Numeric-Int-Representation
Numeric-Int-Truncation
OO concepts
Operator Overloading
Params-Subsumes Methods
Pointers-But Not References
Program Design
Programming Standards
Recs-Structs-HeteroAggs
Recursion
Relational Operators
Runtime-Storage Management
Scope-Visibility
Searching
Searching-Binary
Searching-Linear
Selection Subsumes Operators
Sorting-NlogN
Sorting-Other
Sorting-Quadratic
Strings
Testing
Trees-Other
Trees-Search-Balanced
Trees-Search-Not Balanced