Repository of Wisdom : Automated Support for Composing Programming Exams

Keith Foster, Daryl D'Souza, Margaret Hamilton, James Harland
School of Computer Science and Information Technology
RMIT University
Melbourne, Australia

{keith.foster, daryl.dsouza, margaret.hamilton, james.harland}@rmit.edu.au

Abstract

This paper presents a macro-level view around exam composition. While previous work known as BABELnot (Lister et. al. 2012 [1]) developed a micro-level classification scheme to consistently categorise individual exam questions, this paper's contribution uses a more holistic intent towards collective exam composition to build on the past research from the BABELnot project. Specifically, it addresses a higher order, cognitive layer of complexity on top of the exam classification work derived from the BABELnot project to categorise foundation level programming exam questions. In preparation for this, we analysed use cases for a programming questions database for the composition of exams and selected two for further analysis and implementation. A database designed for use by both educators and researchers exists, called "The repository of Wisdom" (RoW), however, it needs further enhancements to support the goals of this paper. The RoW was designed and implemented as part of previous work (Hamilton, D'Souza, Harland, Rosalina 2014 [3]) that classified questions in programming exams. The retrieval of these questions can be by various attributes such as topic, concept, aptitude, content, level of the course and benchmarked results, with interesting and innovative retrieval options related to ranked queries. The selection process can also be influenced by difficulty scores, ratings and comments given by the instructors who submitted the questions or others who may have trialled them. We would like the repository to be further evaluated in the "real world" by computer scientists and, in particular, academics assessing novice programming ability or designing entry level exams. We have built ontologies and mechanisms for storing and retrieving exam questions and also discuss these in this paper.

Keywords: Novice programming; Computer Science Education; Computer programming exam question retrieval; Mastery, Assessment; Educational measurement; Examination generation; and Use case analysis.

1 Introduction

Programming is a fundamental skill required by most computing or information technology degrees, but the extent and level of competence required can vary greatly from one context to another. Recent research (Sheard 2013 [7]) has shown a lack of model-based exam construction for introductory programming. It was concluded, that "while most of the interviewees had heard of at least one pedagogical model for assessment, only one or two designed their exams with specific reference to such a model".

This discovery is the main motivation for our paper: to contribute a useful model of exam construction in the use case of setting summative programming exams.

Programming is also a skill that can cause significant angst amongst the student body, and is often taught in a cumulative manner across a number of foundational courses. Designing an exam to assess students in such courses is not only a task that requires attention to detail, but also one that must be repeated at least once a semester (and possibly more often), and which needs to take into account any prior access students have to similar exams. This can mean that the process of designing exams is sometimes ad hoc, and one that does not necessarily follow systematic principles, or make it easy to show that the desired learning outcomes of the course are what is actually assessed by the exam.

The BABELnot project (Lister et. al. 2012 [1]) was a multi-institutional project whose main thrust was to identify and develop methods by which programming courses could be compared, one of whose outcomes was a formalism for describing and linking learning outcomes to programming exam questions used in introductory and later courses. Another outcome was an archive of exam questions (often including performance data) expressed in this formalism. This lead to the RoW project, in which these questions formed the basis of a web-accessible database of such questions, (Hamilton, D'Souza, Harland, Rosalina 2014 [3]). This database is accessible to all interested academics, and is intended as a resource that can be used to develop exams for introductory programming courses.

Having a database of potentially useful questions is one thing; using this resource effectively to design and develop exam papers for a particular introductory programming course is another. In particular, it is often necessary to consider some potentially competing attributes, such as reliability, validity, difficulty and distinguishability (Angoff 1971 [4], de Klerk 2014 [5], D'Souza, Hamilton and Harland 2013 [6]).

The ProGoSs system (Gluga 2012 [2]) has like-minded aims to automate parts of the University assessment process, however, it is limited in scope to the design of curricula and matching topics with external (government) requirements.

In this paper we aim to explore the methods by which a reliable and valid exam, with the appropriate level of difficulty and appropriate distinguishability, can be developed using the material in the RoW database and how far this can be usefully automated. To this end, we have conducted some deeper analyses of the needs of the users and more importantly incorporated this analysis into the development of the RoW tool to provide better-automated support.

This paper is structured as follows. First, we explain and define some terms for important concepts and meta-data that we refer to in the following sections. Then we present an orderly flow of analysis and identification of use cases of the system in Section 3. User goals are presented in Section 4 and mechanisms to support these goals are presented in Section 5. Finally, with a framework for RoW usage developed we highlight how we intend to implement this in RoW (Section 6) some further work that needs to be done (Section 7) and draw some conclusions (Section 8).

2 Definitions

Here we define the ontologies, concepts and measurements used throughout this paper. In our context, "ontology" is simply a structured set of names either ordered in a list or arranged hierarchically giving semantics to meta-data types for examination questions.

The ontologies presented are *skill elements* - which include *topics*, *concepts* and *aptitudes* - and difficulty *Levels*. The elements divide and classify the notion of skill into atomic components (that a student possesses and/or a question requires) and the levels give a standard approximation of difficulty. These aid in the selection of appropriate questions for an exam.

The concept of *mastery discrimination* is about how well questions distinguish between a student's programming ability and *Partitioning* is how we implement that. These aid in achieving a successfully discriminating exam. The *mastery scale* is our standard linear measurement for both a student's ability and a question's requirements to solve, which consists of *mastery levels* from 0 to 100. Lastly, *Discrimination validity* is a standard linear measurement of the validity of a question's discrimination correctness by a human expert, which is used to help the relevance of query results.

2.1 Skill elements

A skill element is an abstraction of three classifications: *topics*, *concepts* and *aptitudes*, each of which has its own context for relevance. These are defined below:

Topics refer to domain specific topics. For example, The Java switch statement, Model-view-controller, Regular expressions, C++ multiple inheritance.

Concepts refer to generic concepts applicable to many programming topics. For example, Nested loops, Recursion, Memory Pointers, Object references, Class Inheritance.

Aptitudes refer to fundamental brain skills that are considered critical for the ability to programming. For example, Boolean logic, Imperative processes, Mathematical functions, Pattern matching, Analysing written English and structuring / re-phrasing logically.

2.2 Mastery levels

A *mastery level* is simply a value from 0 to 100 attributed to a question that represents the relative amount of ability required to answer the question successfully. This level is intended to correlate closely with exam assessment scores out of 100. So, a student who passes a question at level 50 would be expected to get a score of at least 50 in the overall exam. Similarly students who achieve full marks for a question at level 80 would be expected to get at least 80 for an overall exam score.

2.3 Difficulty levels

Difficulty is a classification of questions to indicate the kind of partitioning the question achieves. That is, a question's ability to discriminate between students of different mastery levels. Different questions partition the set of students at different points on the *mastery scale*. Our proposed ordered list of *difficulty levels* is as follows:

2.3.1 At-risk

This means that if a student fails such a question it indicates they will likely fail overall.

2.3.2 Basic

If a student fails such a question it means they have an error in their understanding that indicates they will likely fail many other questions.

2.3.3 Mainstream

Failing these questions indicate a gap in knowledge, understanding or application that is the part of the core competency of introductory programming. They indicate nothing about the likelihood of passing other types of questions.

2.3.4 Advanced

Passing an advanced question indicates they will likely pass all previous types. Failing indicates they will likely fail challenging questions.

2.3.5 Challenging

These questions serve to discriminate the most able students from the majority.

Difficulty levels are not related to mastery levels, however, we can consider that they will discriminate at some imprecise points on the *mastery scale*:

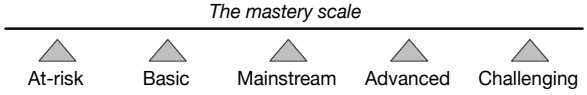

Figure 1: Example Set of Difficulties plotted on the mastery scale.

A secondary use of *difficulty levels* is to aid the exam composer in defining a *partitioning* for each question, which is explained in the following subsection.

2.4 Mastery discrimination and partitioning

Mastery discrimination refers to the ability of a question or an entire exam to distinguish between students of different *mastery levels*. If we test a cohort of students with five questions, each with a different one of the *difficulty levels* above, then we can divide the cohort of students into six subsets based on combined results of all five question as follows:

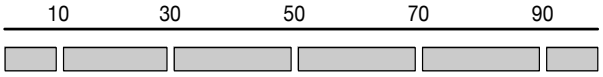

Figure 2: A Five Point Example of Partitioning

Each subset of the cohort contains students that fall into the same range of *mastery levels*. This is assuming, of course, that a student who fails an At-Risk question also fails all higher level questions and the same for Basic and so on. Obviously there will be exceptions to this but statistically this will usually be the case if the questions and their *partitioning* are "good" (i.e. work well for most students).

This process of dividing a cohort into subsets of students based on the student's *mastery level* we call *partitioning* and the subsets we call *partitions*. There is still an ordering of the partitions maintained. A *partitioning point* is a single value in the *mastery level* range that separates two adjacent *partitions*. For example, 10, 30, 50, 70 and 90 (from figure 3).

A **single** question can partition up to a maximum of the number of marks allocated (using one partition point for each mark) plus one.

However, typically and practically, only one partition point is useful (when treating the question as pass/fail for example). So typically, you'll get only two partitions from a single question, even though it may have many more marks allocated to it.

An **exam**, however, can combine the multiple partition points from all of the questions to effectively create a huge partitioning with scores of partition points. This can then partition a cohort into scores of partitions.

In any given exam, the number of partitions is the sum of all the partition points of all questions.

2.5 Discrimination validity

Validity refers to the human element to validate that a given question is in fact a good discriminator based on expertise and past experience. This means an "expert" in teaching programming is needed to review questions and assign a validity score. The Validity score can be used to rank questions from a search result. Ranking is covered in more detail in section 5.2.

3 Use cases

We identify two use cases for composing programming exams with the support of our tool and present each one in the following subsections. In analysing the use cases we draw out a number of goals, some of which are shared between use cases. There were many other use cases that we considered. However, in the interests of focus, we reduced them to these two. Some other use cases are presented in section 8 on further work.

3.1 Composing summative assessment exams

An exam composer is a teaching staff member responsible for writing new exams. In our context, they use the RoW to source questions for a new exam and the metadata incorporated in RoW to guide them in selecting the best or desired questions to include in their exam. We have identified a few goals that an exam composer would have in mind when composing an exam. They include the following.

(Goal 1) Full coverage of the *topics*.

(Goal 2) Full coverage of the *concepts*.

(Goal 3) Full coverage of the *aptitudes*.

Another goal is to have "good discrimination" in an exam meaning the aim is to have the results of a cohort of students to be more distributed across the "mastery scale".

Note that to enable reasoning about student abilities in a standard way, we reuse the hypothetical concept of "mastery level" used in (Gluga 2012 [2]) and defined it as a range from 0 to 100 to be analogous with an assessment result.

For example, a poorly balanced exam may have "clusters" of students with close scores and large gaps between clusters on the result scale. This is undesirable as assessors aim to discriminate between students at many points on the result scale.

We are not claiming that good exams should only display a normal distribution of scores; rather this is an interesting metric that highlights gaps and clusters in the distribution that may indicate a problem with a candidate exam. It would then be the composer's decision whether and how to address these anomalies.

We have identified two different methods by which a composer could improve an exam in this regard:

(Goal 4) Balancing the types of questions in an exam by difficulty.

(Goal 5) Evenly distributing the marks of an exam across all *mastery levels*.

The first is coarse-grained and the second fine-grained. Both are goals for exam composition and are explained further in section 4. The last goal (above) can be satisfied by selecting a number of questions each requiring varying *mastery levels* to pass, such as, including a question to identify the high achievers in the exam, or including a question that bare passing students should be able to do well in. We can expand this last goal to include an additional goal:

(Goal 6) Identifying good *mastery-partitioning* questions

This last goal is really a means to and end, the end being distributing an exam's marks across *mastery levels* using the identified partitioning questions.

3.2 Composing university entrance exams

An entrance exam is one where an educational institution asks prospective students to sit a formal exam prior to making an offer for them to be enrolled into the program. These entrance exams have two main aims. Firstly, to rank the students by their abilities and thereby enable the school to make offers to the top students. Secondly, to partition the students based on aptitude and thereby enable the school to advise the student which program would suit them best (or indeed, if no program suits their aptitudes, they may be advised to seek a different field of study, or undertake some preliminary course of study).

Note that we are not positing nor refuting that the "Geek Gene" (Lister 2010 [8]) determines aptitudes, rather, the measurement of a prospective student's current aptitudes in combination with the University program's aptitude preferences provides relevant information with which to influence the decision. It may well be the opposite case of the "Geek Gene" theory where a University has a program designed to build programming related aptitudes through targeted intellectual exercises and decides to deliberately make an offer to a student who is currently lacking in an important aptitude but who shows great potential on other measures. In both cases, the aptitude assessments are very useful.

In order to generate such an exam there are a few goals we think an exam composer would have in mind:

(Goal 6) Identifying good *mastery-partitioning* questions (for *aptitudes*).

(Goal 4) Balancing the types of questions in an exam by difficulty.

(Goal 3) Full coverage of the *aptitudes*.

Note, we do not include goal G5 - evenly distributing the marks of an exam across all mastery levels - because we think that people don't have a **degree** of mastery in an aptitude, rather, we think the intention of the definition of aptitudes is that people either have an aptitude or they don't.

Concepts and topics are not included in these goals because this is what the students learn, not what is relevant to entrance. A high school student will not have been exposed to many of the concepts usually contained in an introductory programming course at a tertiary education level. Therefore, a specific set of secondary education mathematical and problem-solving *aptitudes,* designed to assess programming *aptitudes* would be more appropriate in this use case.

The set of *aptitudes* for programming should be specific and well known. Identifying a standard set of *aptitudes* for this purpose is another project in itself for further work (Section 7). In the following section we will explain each goal identified in this section (G1 through G6) in more detail.

4 Goals for exam composition

In this section, we briefly explain all the goals required by the use cases identified in the previous section. We explain the expected benefits to the exam composer of each goal in order to highlight what the proposed support mechanisms (in the following Section 5) will need to achieve.

4.1 Full coverage of assessable skill elements

A creator of formative assessments desires an exam that fully covers a set of *topics* and *concepts* that she has decided needs to be assessed. Similarly, a creator of University entrance assessments desires an exam that fully covers a set of *aptitudes* that she has decided needs to be assessed.

The goal is for the set of candidate exam questions to approach 100% coverage of all the specified *skill elements* to be assessed. Effectively there are three goals, one for each sub classification of *skill element*:

(Goal 1) Full coverage of the specified *topics.*

(Goal 2) Full coverage of the specified *concepts.*

(Goal 3) Full coverage of the specified *aptitudes.*

4.2 Optimising mastery discrimination in an exam

In our analysis we identified two approaches to optimising the effectiveness of an exam to discriminate between students with different mastery levels. The first is coarse-grained and uses the difficulty levels defined in section 2. The second is fine-grained and uses *partitionings*, also defined in section 2. Both approaches aim to spread the marks allocated to questions across either difficulty or mastery levels, according to what the exam composer specifies.

(Goal 4) Balancing marks across ***difficulty levels …***

When a composer is using the coarse-grained *difficulty balancing* method to compose a new exam, they desire their exam to contain questions at many different *difficulty levels* and different amounts of marks at each *difficulty level*. To this end, we propose to capture this desire in what we call a "*difficulty signature*". This signature simply defines the proportion (out of 100%) of each *difficulty level* desired. For example: 10% at-risk, 20% basic, 40% mainstream, 20% advanced and 10% challenging. The composer should define the desired *difficulty signature* first and then the system can calculate how closely a candidate exam approximates the desired *difficulty signature*. It is important to note that the percentages above would be weighted on the **marks allocated** to a question, and not on the number of questions.

(Goal 5) *Distributing marks across* **mastery levels …**

When a composer is using the fine-grained *partitioning point distribution* method to compose a new exam they aim to have the graph of marks allocated to

questions by *partitioning point* to approach some specified pattern (or graph shape).

Normal distribution is an example of a graph shape, however, a normal distribution is not appropriate for exam discrimination; rather, an even distribution (horizontal line) starting at 30 on the mastery scale may be more appropriate. In a "normal" world, the normal distribution would show itself when plotting the number of students attaining a given score, not the marks requiring that *mastery level*. Therefore, the composer needs to specify the desired graph shape for *mastery level* distribution before applying the metric to a candidate exam.

4.3 (Goal 6) Identifying mastery-partitioning questions

In order to create a candidate exam the exam composer needs to select candidate questions from the database. The RoW tool provides a search mechanism for this and then ranks them according to some pre-defined weighting formula.

This goal is specifically about finding questions that make good mastery discriminators and we use *partitioning* to accomplish this. *Partitioning* refers to the mathematical concept of partitioning a set into two or more subsets. In our context, we want to partition a cohort of students into subsets ordered by *mastery level*. This means every student in a higher subset has a higher *mastery level* than every student in the next lower subset and so on.

Once a set of candidate *partitioning questions* are identified for a particular *skill element* and *partitioning point*, we would like to rank them according to their *discrimination correlation*, popularity, validity and other qualities. The composer uses the ranking to aid in the selection of questions for their exam (higher ranked questions being preferred). We explain how we calculate *partitioning question* ranking in Section 5.2.

We also explain how we can map from a *partition* in any particular question to the standard *mastery level*. A given question may have a score out of five, which results in six partitions. Each partition needs to be mapped to one *mastery level*. How this mapping is achieved is addressed in Section 6.1.

5 Mechanisms to support goals

Having identified the goals that exam composers have, we now describe the mechanics of acheiveing these goals with the RoW tool in mind. We have divided the mecahanisms into four broad categories: searching and ranking questions; then coverage and discrimination analysis of whole exams.

5.1 Searching for candidate questions

In order to create a candidate exam the exam composer needs to search for candidate questions. Most of this functionality is already in the RoW tool allowing her to search by topic, content or level of the course. To this, we add the ability to search by concept and aptitude.

5.2 Ranking of questions

Search results may return many questions – too many to manually review in the selection process – so a ranking of the questions is desirable to reduce the selection time.

One way to rank questions is to compare the ordering of students based on the question's partitioning with the ordering of students based on their overall exam score (we call this *discrimination correlation)*. If the question ordering closely correlates with their overall exam ordering then this could be considered a "good partitioning". If not, then perhaps this question's partitioning is not reliable. However, if the discrimination of a question does closely correlate with the overall exam discrimination, this would not necessarily mean that it is correct, but rather it means that the question's *partitioning point* is consistent with the average scores of the aggregation of questions in past exams.

When ranking *partitioning* questions, we can mix in a number of factors, *discrimination correlation* being just one of them. We have identified four additional factors that could be used in the ranking process:

- **Popularity** - the percentage of "likes" on a question.
- **Validity** - the discrimination validity score assigned by a human expert.
- **Commentary** - a full text analysis of the comments on a question to derive how positive the comments are.
- **Recency** – how recently was the question used in an exam.

In order to rank questions then, an exam composer needs to define a *ranking formula* (or let it default to one) such as this:

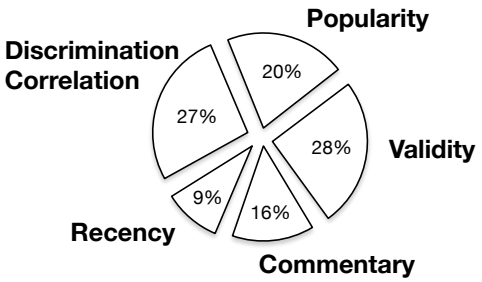

Figure 3: Example *Partitioning Question* Ranking Formula.

Each of these factors will have a relative weight assigned to it for calculating an overall ranking.

5.3 Coverage analysis

Once a candidate exam is established, the exam composer wishes analyse how well it covers all the skill elements desired. Firstly, the composer needs to select a subset of skill elements applicable to this exam (for entrance exams this would likely be aptitudes only). Subsequently the system can then calculate the percentage of the desired elements covered by the set of candidate questions.

The RoW tool's coverage analysis feature should also graph the relative coverage for each element so the composer can visually identify "gaps", "peaks" or "dips"

not aligned with the specified coverage graph in order to address them.

5.4 Discrimination analysis

Once a candidate exam is established, the exam composer also wishes to analyse how well it will discriminate between students with varying abilities (measured using mastery levels). As described in Section 4.2, there are two ways to do this: *difficulty balancing* and *partitioning point distribution*. The RoW tool needs a way for composers to pre-define both *difficulty signatures* and *partitioning* graph shapes.

With these created the tool can then calculate the coverage percentage and graph correlation for every question according to any given signature and shape. These calculations are implemented with derived attributes defined in Section 6.2.

6 Tool Support

6.1 Meta-data support

In order to implement the meta-data required by the use cases we propose to expand the "topic" attribute into three classifications (either as three attributes or one attribute of some type with three subtypes): *Aptitude*, *Concept* and *Topic*. For each of these three entities we define the following attributes: *Name* and *Description*. We require each question to have "sets" of aptitudes, concepts and topics associated with them which requires three attributes: Aptitudes required, *Concepts required* and *Topics required*.

As a guide for designing good discriminator questions, it is recommended that questions have very few associated skill elements (ideally one) so the assessment scores and analytics reflect the ability a student has with one specific skill element. Conversely, a question that covers too many skill elements will be ineffective as a discriminator because the assessment scores will be reflecting the student's aggregated ability with many elements, which cannot be deconstructed back to the elemental level.

For every question, two attributes are defined; *Partitioning* and *Discrimination correlation*. *Partitioning* is a mapping from the marks allocated to a question to the *mastery level*. *Discrimination correlation* is calculated statistically (across all students that sat any exams that included this question). A higher *discrimination correlation* indicates the question performs better as a mastery discriminator.

Partitionings are defined by people adding them to the database and so it requires some further explanation and examples. For each possible assessment, score of the question define the *mastery level* (0 to 100) required to attain that score. For example, a question assessed out of five could have the following partitioning:

Score	Mastery-Level
1	30
2	50
3	60
4	70
5	80

Figure 4: Example Partitioning with six partitions.

A score of zero implies a mastery level less than thirty and a score of one has a mastery level from 30 to less than 50. A different question assessed out of five might have the following partitioning:

Score	Mastery-Level
4	60
5	80

Figure 5: Example Partitioning with three partitions.

A score of zero to three implies a mastery level less than sixty (giving the third partition).

Finally, we need a *Difficulty Level* attribute defined for all questions.

6.2 Analysis support

When creating candidate exams for coverage and discrimination, composers need some information to aid them in the analysis process. Two derived attributes are required on exams: *Skill Coverage* and *Discrimination Correlation.*

Both of these have calculations that involve external data including: desired skill elements, difficulty signature or partitioning graph shape. These three external data items must be defined by the composer prior to running queries involving these attributes.

6.3 Validation support

To validate the correctness of the *discrimination correlation* we need historical data on past question scores by the same students. To support this we define the *Discrimination Validity* attribute for all questions. The value of the *discrimination validity* can be just a number between 0 and 10 which indicates its success rate in discrimination compared to the truth as judged by the expert who enters these validation scores.

6.4 Query support

The two queries described in this section both depend on ranking to be useful. To support this, the RoW tool needs to implement a ranking formula feature that allows composers to alter the weighting of each factor to what they desire. For the purpose of ranking questions for their partitioning qualities, the tool needs to pre-define a derived attribute with a default formula like the one described in Section 5.2. In addition, we need to define the *Discrimination Ranking* attribute for all questions.

Once all the supporting attributes and features are in place, the relevance of useful queries should improve. We have described two useful queries that the tool can perform to realise the use cases identified in Section 3:

6.4.1 Query to fill a gap in coverage

- Filter on Subsets of *topics*, *concepts* or *aptitudes*.
- Group by *Difficulty Level.*
- Order by *Discrimination Ranking.*

The query above is useful to find good discriminator question for an exam. The search can be focused on finding questions for specific skill elements for which the exam currently under construction is lacking. These

queries can be iterated and questions collected until full coverage of the desired skill elements is achieved.

6.4.2 Query to fill a gap in difficulty level

- Filter on *Difficulty Level.*
- Group by *Topic* or *concept* or *aptitude.*
- Order by *Discrimination Ranking.*

This query is useful to improve the "balance" of an exam (which in turn aims to get a normal distribution of results for a cohort). The search can be focused on finding question with specific discrimination types for which the exam currently under construction is lacking. These queries can be iterated and questions collected until the desired "discrimination signature" is achieved.

7 Further Work and Reflection

The next phase of work that needs to be performed is creating meta-data about existing exam questions in the repository. We desire to have a significant and useful portion of the questions fully attributed using the above mechanisms and ontologies. To that end, we need experts in teaching introductory programming to create this meta-data and define some ontologies.

Some ontologies might include definitions of standard sets of *aptitudes*, *concepts* and *topics*. In the latter case the set is specific to individual courses so a common set of topics that can be built upon by each educational institution is desirable.

For each question, definitions are required for the *skill elements* and difficulty levels that apply to this question, one partitioning and optionally a discrimination validity for the question. The standard ontologies need to be defined in a focus group consisting of experts. To that end, a forum like an international conference workshop is recommended to complete the job.

Adding values for the above attributes to each question and the subsequent building up of institution/course specific topics can be done incrementally by teachers on their own schedule and in their own places of employment.

Beyond this, we foresee a number of potential directions for the project to progress. We briefly introduce four potential use cases that were not addressed in this paper:

1. Identifying questions that have been used in the past. For example, a computer science educator might find it interesting to compare how their cohort of students is performing in relation to past cohorts of students, or against a cohort from another campus, university or country. This could involve selecting a question or questions for which benchmarked data has already been provided in the RoW, and including this in their own exam, for research purposes.
2. Automatically generating "good" exams given a set of *skill elements* plus a *difficulty signature* and/or a *partitioning distribution graph shape*. The RoW tool could automate the search for the optimal exam that most closely approximates the given *learning attributes*, *difficulty signature* and/or *graph shape*.
3. Analytics on the *discrimination correlation* of individual questions in relation to overall exam scores in order to iteratively refine the question in order to progressively achieve better and better correlation.
4. Collecting data on practical assignments and formative assessments in addition to summative exams for the purpose of analytics. For example, predicting summative results from formative results or at least discovering unknown relationships between them.

In this paper, the mechanisms for improving the mastery discrimination of exams rests on the assumption that questions in an exam are "targeted" (i.e. that require one or only a few skill elements). The reason being that fine-grained allocation of marks to a few specific skill elements greatly aids the accuracy of the partitioning process. If, however, exams contain large questions and marks are allocated broadly across a large number of skill elements then this process breaks down.

To mitigate this flaw in the future, we suggest composers break down large exam questions and allocate smaller amounts of marks to each skill element associated with the question. Commonly, exam markers will have a marking scheme that resembles this already, so the next logical step is to organise these finer-grained marks in a marking scheme based on skill elements and capture results in the RoW. In this way, the discrimination processes benefit and opens the potential for more detailed analytics on individual questions that can improve discrimination at the question level as well.

8 Conclusions

In this paper, we have presented two use case scenarios for developing examination papers based on questions available in the repository of wisdom. We have discussed how mechanisms, ontologies and metrics can be defined for the exam composer to select questions for an exam that satisfy the goals of these different use cases. The *mastery levels* presented in this paper can be mapped to Bloom's taxonomy, or whatever mastery scales the composer requires giving their exam papers the required educational measurement.

It is hoped that the analytical processes presented will result in the dissemination of better questions and exams that discriminate well. As more data is collected about how students have fared using the various questions in the repository, it will become possible to analyse more deeply the relationships between aptitudes, concepts and topics and gain further insights into detecting and overcoming obstacles in learning programming.

9 References

[1] Lister, R., Corney, M., Curran, J., D'Souza, D., Fidge, C., Gluga, R., Hamilton, M., Harland, J., Hogan, J., Kay, J., Murphy, T., Roggenkamp, M., Sheard, J., Simon and Teague, D., Toward a shared understanding of competency in programming: An invitation to the BABELnot project. In proceedings of the 14th Australasian Computing Education Conference (ACE2012), Melbourne, Australia, 2012.

[2] Gluga, R., Kay, J., Lister, R., "ProGoSs: Mastering the Curriculum", Proceedings of the Australian

Conference on Science and Mathematics Education. (ACSME2012). Sydney, Australia. September 2012. in Australian Conference on Science and Mathematics Education (ACSME2012). ed M. Sharma and A. Yeung. UniServe Science, Sydney, Australia, pp.92-98 http://ojs-prod.library.usyd.edu.au/index.php/IISME/article/view/5914 last accessed: 13.12.13.

[3] Hamilton, M., D'Souza,D.,Harland, H., Rosalina,E., Repository of Wisdom: A database for storing and retrieving classified and benchmarked exam questions for introductory programming courses, Proceedings of the 23rd Australasian Software Engineering Conference (ASWEC), Sydney, April, 2014.

[4] Angoff, W., Scales, norms, and equivalent scores, in R.L. Thomdike (Ed.), Educational measurement (2nd ed., pp. 508-600). Washington, DC: American Council on Education, 1971.

[5] de Klerk, G., Classical test theory (CTT), in M. Born, C.D. Foxcroft & R. Butter (eds.), Online Readings in Testing and Assessment, International Test Commission, http://www.intestcom.org /Publications /ORTA.php, last accessed 1/09/2014.

[6] D'Souza,D., Hamilton, M., and Harland,J., A Comparative Analysis of Results on Programming Exams, Proceedings of the Fifteenth Australasian Computing Education Conference (ACE2013) 117-126, Adelaide, January,2013

[7] Sheard, J.,Simon, Carbone, A., D'Souza, D., Hamilton, H., Assessment of Programming: Pedagogical foundations of exams, (2013).

[8] Lister, R., Computing Education Research, ACM Inroads, Vol 1, No.3,2010,http://203.144.248.23/ACM.FT/1840000/1835434/p16-lister.pdf, last accessed 1/09/2011