Evaluation of Formal Reasoning Abilities Using a Concept Inventory

Joseph E. Hollingsworth[1] and Murali Sitaraman[2]

[1] Department of Computer Science
Indiana University Southeast, New Albany, IN 47150, USA
jholly@ius.edu

[2] School of Computing
Clemson University, Clemson, SC 29634, USA
murali@clemson.edu

Abstract. To understand, assess, and improve student abilities to perform analytical reasoning about the correctness of object-based software they build, we have developed a two-part reasoning concept inventory. The inventory is a collection of multiple choice questions. The inventory makes use of a minimal set of formal notations to model and present operations on objects. The inventory has been administered in two required courses for CS majors at Clemson, and will be offered again this semester. An analysis of results helps clarify what concepts were well understood and where instructional improvements are needed. Furthermore, since the questions are multiple choice, wrong answers also provide useful information.

Keywords: Education, specification, reasoning, and software engineering

1 Introduction

A central goal of all Computer Science education is to teach students how to reason analytically about the code they develop. Analytical reasoning, when introduced as an alternative to testing for finding and fixing errors, helps students understand better not only that the software they build works but also why it works correctly. To reason analytically about engineering software involving objects, students also need to understand formal specifications of objects that describe the behavior of operations in mathematical terms. To assess student learning, we have developed an initial reasoning concept inventory (RCI) focusing on skills needed to reason analytically about software correctness. The inventory can provide the CS education community a common tool for assessment of formal reasoning and eventually facilitate a comparison of alternative instructional approaches and lead to better instructional practices.

Like other concept inventories, the RCI is a collection of multiple choice questions. Goldman [4] points out that "CIs assess students' conceptual understanding, not their problem solving, design, or interpersonal skills." A focus on analytical reasoning

distinguishes the RCI from [10], where the emphasis is on language-independent concepts typically taught in first year software development courses. The RCI shares some goals with [5], where a time-intensive interview process using a "think aloud" approach has been used to identify common misconceptions with propositional logic and student ability to translate English to Boolean expressions. The RCI includes questions to assess student abilities on higher levels of Bloom's taxonomy [12] (e.g., at the application/analysis level), because it is at these higher levels that a software developer must succeed in order to reason about software correctness. Therefore, we have taken a performance-based learning outcome approach to develop the RCI, a starting point that we hope will evolve into an accepted standard.

2 An Overview of the Reasoning Concept Inventory

The inventory of reasoning principles underlying the RCI is a result of several years of education research and assessment on teaching formal reasoning at Clemson and other institutions. The process used to develop it along with its technical basis can be found in [2, 3] where the reasoning principles are organized into 5 topic areas: Boolean Logic, Discrete Structures, Precise Specifications, Modular Reasoning, and Correctness Proofs. Whereas the first two areas provide the basics for reasoning, the last three topic areas focus on software engineering aspects of understanding contract specifications, code correctness reasoning using those specifications, and proofs, and they are the focus of our prior research and analysis in [3] on which the RCI is based.

The RCI pre/post tests have two parts, each consisting of 10 multiple choice questions. The number of questions has been kept to a minimum because one goal is for instructors to be able to administer each part within 15-20 minutes and for students to be focused. Part 1 is aimed at software development foundations (typically covered in a second or third course in CS) and is more elementary than the other. It contains 3 questions focusing on precise specification understanding, 5 questions concerning the role of specification contracts in modular software development and reasoning, and 2 questions emphasizing basic proofs. The more advanced second part is aimed at a subsequent software engineering course and it has 4 questions targeted at specification and modular reasoning aspects with the other 6 devoted to elements of establishing correctness proofs (e.g., loops and invariants).

It has been a challenge to minimize the knowledge base needed to answer the questions in the RCI while at the same time introducing questions that involve object-based specifications and reasoning. For questions involving objects we almost exclusively use queues because it is an everyday concept familiar to most students. Our formal descriptions of queues, where they are explicitly needed in the questioning, use mathematical strings (similar to sequences except that no positions are involved) and notations for string concatenation and length that are straightforward to understand. One other notation is used to distinguish input (#x) and output (x) values of parameters to operations. These notions come from RESOLVE [8,9] the formal specification language used in our classrooms.

While the inventory has to use a particular syntax for the notations that are involved, it is easy to see that the specification notations can be changed to suit the target audience as long as the same reasoning concepts are tested. In other words, standardizing understanding of analytical reasoning principles does not require standardizing the use of any particular notation. Any well-known formal specification language such as VDM or Z (or others summarized in [7]) can be used for the inventory.

At Clemson, analytical reasoning principles are taught in two required courses: a second-year course on software development foundations (CP SC 2150) and a subsequent course on software engineering (CP SC 3720) which has CP SC 2150 as its prerequisite. RCI Part 1 was administered before and after the first course and RCI Part 2 was administered before and after the second course.

3 Elementary Reasoning Concept Inventory

Specific performance-based learning outcomes are summarized in [2] and they guide the questions for the inventory. Due to space limitations, only some questions are shown here. What CS educators should note about questions, such as #9 below (and others not shown) is that the essence of the question is what is important, not so much the specific data types or mathematical notions. Instead of queues, for example, arrays or lists, possibly mathematically conceptualized with functions or sequences, respectively, could be used. Same comments apply to the control construct questions in the next section.

Two questions in the inventory (shown below) concern the idea of modular reasoning with the specific learning outcome being students "understand the design-by-contract principle" [11]. In design-by-contract, the implementer of an operation can assume that the precondition holds when the operation is called which leads to a more efficient implementation. Additionally, the implementer must be able to confirm that the implementation does indeed guarantee the results specified by the post condition. So the answer expected for question #5 below is "I and IV" (answer choice b). A related question (#6 below) applies to the calling client. In design-by-contract, the client is responsible for guaranteeing that the precondition holds prior to calling an operation with a precondition and then reaps the benefit of being able to assume the postcondition holds after the call. The correct answer is II and III (answer choice c).

5. In verifying the correctness of code that *implements* an operation with a requires clause *pre* and an ensures clause *post*, the verifier:
   - I. may assume that pre is true in the initial (or the first) state of the code
   - II. must prove that pre is true in the initial (or the first) state of the code
   - III. may assume that post is true in the final (or the last) state of the code
   - IV. must prove that post is true in the final (or the last) state of the code

   a. I and III
   b. I and IV
   c. II and III

d. II and IV
e. None of the above

6. In verifying the correctness of an implementation that *calls* an operation with a requires clause *pre* and an ensures clause *post*, the verifier:

I. may assume that pre is true in the state before the call
II. must prove that pre is true in the state before the call
III. may assume that post is true in the state after the call
IV. must prove that post is true in the state after the call

a. I and III
b. I and IV
c. II and III
d. II and IV
e. None of the above

Question #9 (below) assesses understanding of object-based reasoning that utilizes mathematical strings to model queue variables Q0, Q1, and Q2. String theory as well as other concepts are integrated into our curriculum and provide the context for many of the questions in the inventory. The correct answer for #9 is e.

9. Suppose that in verifying a piece of code the following assertion needs to be proved (goal): Q2 = empty_string. The assumptions available to prove this goal are the following, where Q0 and Q1 are values of type Queue and E0 and E1 are values of an Entry E in some other states.

I. |Q0| = 2
II. Q0 = <E1> o Q1
III. Q1 = <E2> o Q2

a. Goal is not provable from the assumptions
b. I only
c. I and II only
d. II and III only
e. I, II, and III

Part 1 of the RCI pre/post test was administered in CP SC 2150 (Software Development Foundations) course at Clemson and it contains the three questions above along with seven others. Detailed contents of this second-year required course (fourth course in our sequence for CS majors) that introduces Java and covers software engineering ideas and programming language concepts may be found in [6]. Only two to three weeks of the course are devoted to basic elements of formal specification and analytical reasoning.

Figure 1 shows the results of administering Part 1 of the RCI in one section of CP SC 2150 at Clemson during the first semester of 2014. (Though the test was given in

both sections of the course, only a few self-selected students completed the post test for the second section; while the student performance—in terms of percentages—was indeed better than the results for the section reported here, self-selection possibly biased those results. Therefore, we do not report them here.)

In the chart, each of the 10 questions is represented by two bars, the left hand bar represents percentage of students that answered the question correctly at the outset of the semester (pre-test), and the right hand column for the end of the semester (post-test). The improvements range from 21% for question #2 to 77% for question #1, with most others in the 50-70 range. While such improvements may be anticipated, they cannot be taken for granted just because the materials are presented. What is perhaps more interesting from a pedagogical perspective is the percentage of students who answered design-by-contract question #5 correctly (70%) versus #6 (50%). This is the kind of insight a reasoning concept inventory can provide educators.

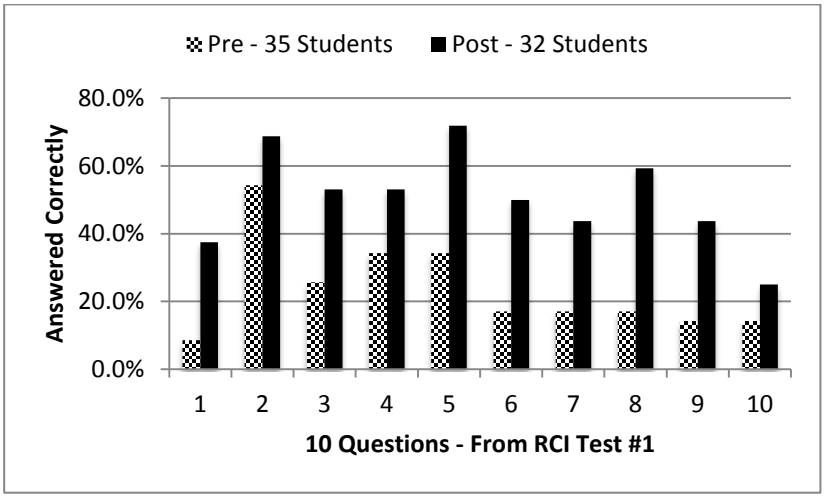

Fig. 1. CP SC 2150 Results

One important aspect of concept inventory tests is that the distractors (the inferior or incorrect choices in a multiple choice question) need to be based on common misconceptions held by students for a particular topic. We have been able to gather some of these distractors through observation in the classroom. Prior to developing the two-part RCI, we have frequently administered collaborative in-class activities where students work in pairs to solve problems at the analysis/application level. Some of these in-class activities are based on the reasoning topics covered by RCI Part 1 & Part 2. During the activity the instructor moves about the class helping students to overcome difficulties with the application of a reasoning principle. At this time the instructor sees firsthand the various misconceptions students have about the reasoning topics. We have developed many of the distractors found in the inventory test from these first hand observations.

4 Advanced Reasoning Concept Inventory

The questions in Part 2 of the RCI are at a higher level of Bloom's taxonomy and often involve application of the reasoning principles to analyze given instances. Some of these involve imperative-style code using objects, others involve assertions and proofs, and yet others a combination. We present a few illustrative questions.

2. Consider the following code, where I and J are integers and ":=" denotes assignment.

```
Max := I + J;
If I > J then
    Max := Max – J;
end;
If J > I then
    Max := Max – I
end;
```

   a. The code is correct and finds the maximum of I and J.
   b. The code has an error that can be fixed by changing the first if-then statement.
   c. The code has an error that can be fixed by changing the second if-then statement.
   d. The code has multiple errors.
   e. None of the above

6. Consider the following piece of code. Assume that Queue Q2 is empty initially.

```
While (not Is_Empty(Q1)) do
   Dequeue(E, Q1); Enqueue(E, Q2); Dequeue(E, Q1); Enqueue(E, Q2);
end;
```

   a. The code moves the contents of Q1 to Q2 in the same order.
   b. The code moves the contents of Q1 to Q2 in the reverse order.
   c. The code is wrong if Q1 is empty.
   d. The code is erroneous.
   e. None of the above.

7. Consider the following piece of code. Assume that the initial value of Queue Q1 is #Q1 and that Q2 is empty initially. Which of the given invariants is maintained by this loop, where o denotes concatenation?

```
While (not Is_Empty(Q1)) do
        Dequeue(E, Q1); Enqueue(E, Q2);
end;
```

a. Q1 = Q2.
b. Q1 = #Q1.
c. Q2 is empty.
d. Q1 o Q2 = #Q1;
e. Q2 o Q1 = #Q1.

The correct answer choices for the three questions are (d), (d), and (e), respectively.

Figure 2 shows the combined results of administering Part 2 of the RCI pre/post tests in two sections of a software engineering course in the second semester of 2014.

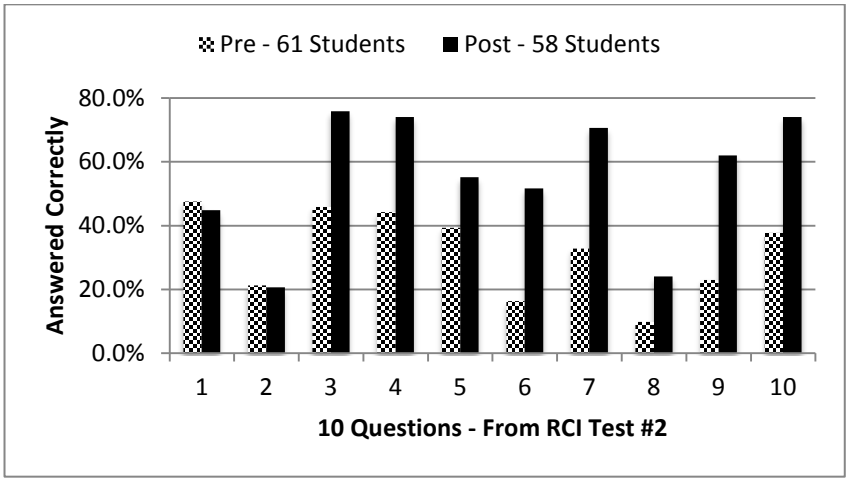

Fig. 2. CP SC 3720 Combined Results of Two Sections

Whereas one section of the course covered the reasoning materials minimally (over 3 weeks) and involved only a simple reasoning assignment, the other section covered the topics over a 5-week period [1]. The results were indeed better for the section with the extended coverage. An analysis of the post-test is revealing. For example, question #2 involves noticing the potential computational Integer overflow/underflow problem in the first line as well as noticing that the code fails when I equals J; more students noticed an error in post-test, though they failed to notice multiple errors. Also, nearly the same high percentage (over 70%) of students answered questions concerning invariants correctly, involving integers (not shown) or queues (#7).

5 Summary

This paper presents an inventory of questions to assess the conceptual understanding of analytical reasoning principles. The questions have been guided by performance-based learning outcomes developed through extensive educational research and assessment over multiple years. Employing the inventory in two courses has made it possible to pinpoint what works and where improvements are needed. The inventory

can form a basis for continuous improvement and to facilitate exchange of different formal software engineering methods among educators.

Acknowledgments. We thank members of the RESOLVE/Reusable Software Research Groups and the course instructors Blair Durkee, Cathy Hochrine, and Stephen Schaub. This research is funded in part by the US NSF grants CCR-0113181, DUE-1022191, and DUE-1022941.

References

1. Cook, C.T., Drachova-Strang, S.V., Sun, Y.-S., Sitaraman, M., Carver, J.C., and Hollingsworth, J.E., 2013. Specification and Reasoning in SE Projects using a Web IDE. In *26th Conference on Software Engineering Education and Training (CSEE&T)* IEEE Computer Society, San Francisco, California, United States, 229 - 238.
2. Drachova, S.V., 2013. Teaching and Assessment of Mathematical Principles for Software Correctness Using a Reasoning Concept Inventory, Ph.D. Dissertation, Clemson University.
3. Drachova-Strang, S., Hallstrom, J. O., Sitaraman, M., Hollingsworth, J. E., Krone, J., and Pak, R., "Teaching Mathematical Reasoning Principles for Software Correctness and Its Assessment," *ACM Transactions on Computing Education*, 2015, accepted to appear.
4. Goldman, K., Gross, P., Heeren, C., Herman, G.L., Kaczmarczyk, L., Loui, M.C., and Zilles, C.. 2010. Setting the Scope of Concept Inventories for Introductory Computing Subjects. *Trans. Comput. Educ.* 10, 2, Article 5 (June 2010), 29 pages.
5. Herman, G.L, Loui, M.C., Kaczmarczyk, L., and Zilles, C. 2012. Describing the What and Why of Students' Difficulties in Boolean Logic. *Trans. Comput. Educ.* 12, 1, Article 3 March 2012, 28 pages.
6. Hallstrom, J. O., Hochrine, C., Sorber, J., and Sitaraman, M. 2014. An ACM 2013 exemplar course integrating fundamentals, languages, and software engineering. In *Proceedings of the 45th ACM technical symposium on Computer science education* (SIGCSE '14). ACM, New York, NY, USA, 211-216.
7. Hatcliff, J., Leavens, G. T., Leino, K. R. M., Müller, P., Parkinson, M., Behavioral Interface Specification Languages. *ACM Computing Surveys 44*, 3, June 2012.
8. Sitaraman, M. and Weide, B.W. 1994. Component-based software using RESOLVE. *SIGSOFT Softw. Eng. Notes* 19, 4 (October 1994), 21-22.
9. Sitaraman, M., Adcock, B., Avigad, J., Bronish, D., Bucci, P., Frazier, D., Friedman, H.M., Harton, H., Heym, W., Kirschenbaum, J., Krone, J., Smith, H., and Weide, B.W., 2011. Building a push-button RESOLVE verifier: Progress and challenges. *Formal Aspects of Computing 23*, 5, 607-626.
10. Tew, A.E. and Guzdial, M. 2011. The FCS1: a language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (SIGCSE '11). ACM, New York, NY, USA, 111-116.
11. Meyer, B. 2000. Design by contract and the component revolution. *International Conference on Technology of Object-Oriented Languages* (July 2000), 515-515. IEEE Computer Society.
12. Bloom, B. S., and Krathwohl, D. R. (1956). Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain.

