Abstract — Discussed in this work is a high level overview of a systematic approach [1] to introducing mathematical reasoning principles into the curriculum, and assessing how well the students have learned them by collecting direct evidence of student learning.

Index Terms — reasoning concept inventory; teaching mathematical reasoning; analytical reasoning principles; learning outcomes; computer science curriculum; RESOLVE

I. INTRODUCTION

ONE THESIS that has driven this work is that mathematical reasoning principles to support formal reasoning about software behavior can be taught, learned, and applied at the undergraduate level. We are not talking about undergraduates being able to prove the correctness of an operating system, for example, but rather being able to reason effectively about the common software written by undergraduates for their software development courses. This includes using design by contract for interactions between modules, using specifications internal to modules to reason about specific inter and intra module behavior, e.g., loop correctness, module level invariants, and correspondences, etc. To gather evidence that supports this thesis we developed a reasoning concept inventory (RCI) that identifies a collection of reasoning concepts that support effective mathematical reasoning about software.

Layered on top of the RCI are specific learning outcomes that are written in such a way as to describe the expected behavior of students who can utilize the concepts of the reasoning inventory.

Dovetailing with the RCI and the learning outcomes are instructional materials needed to support undergraduate instruction of the RCI reasoning concepts.

By utilizing the RCI, learning outcomes, and instructional materials, we were able to collect direct evidence of student learning relative to the concepts of the reasoning inventory. This evidence supports our thesis that students at the undergraduate level can effectively be taught, can learn, and can apply skills for reasoning about software correctness with respect to the common software written in undergraduate software development courses.

II. THE RCI

The RCI [1] captures the fine details of basic reasoning skills for reasoning about software correctness, developing high quality software, and for understanding why software works as specified. It captures five central areas for reasoning about software correctness: Logic, Discrete Math Structures, Precise Specifications, Modular Reasoning, and Correctness Proofs. Each area is broken down, or subdivided into three sublevels. We believe the top three levels of the RCI to contain reasoning concepts that are necessary and sufficient for supporting effective mathematical reasoning about software. Below in Figure 1 is Section 3 from the RCI, Modular Reasoning, with its top two levels expanded. To see the entire RCI, browse to the following website: www.cs.clemson.edu/group/resolve/teaching/inventory.html.

### Figure 1 – Top Two Levels of RCI Section 3

<table>
<thead>
<tr>
<th>3. Precise Specifications</th>
<th>3.1. Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2. Specification structure</td>
</tr>
<tr>
<td></td>
<td>3.3. Abstraction</td>
</tr>
<tr>
<td></td>
<td>3.4. Specifications of operations</td>
</tr>
</tbody>
</table>

III. LEARNING OUTCOMES & DIRECT EVIDENCE OF STUDENT LEARNING

One method for collecting direct evidence of student
learning is to first write learning outcomes, and then create student assessments based on those learning outcomes. A learning outcome in this context describes the behavior that a student is expected to exhibit once she has learned the material related to the learning outcome.

Expanding RCI 3.4 (see Figure 1) two more level reveals the following structure as seen in Figure 2.

<table>
<thead>
<tr>
<th>3.4.1. Initialization and finalization specification</th>
<th>3.4.2.1. Operation Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.2. Operation signature</td>
<td>3.4.2.2. Formal parameters</td>
</tr>
<tr>
<td>3.4.3. Pre- and post-conditions</td>
<td>3.4.2.3. Return Value</td>
</tr>
<tr>
<td></td>
<td>3.4.3.1. Specification parameter modes</td>
</tr>
<tr>
<td></td>
<td>3.4.3.2. Responsibility of the caller</td>
</tr>
<tr>
<td></td>
<td>3.4.3.3. Responsibility of the Implementer</td>
</tr>
<tr>
<td></td>
<td>3.4.3.4. Equivalent specifications</td>
</tr>
<tr>
<td></td>
<td>3.4.3.5. Redundant Specifications</td>
</tr>
<tr>
<td></td>
<td>3.4.3.6. Notation to distinguish an incoming value in the post-condition</td>
</tr>
</tbody>
</table>

Figure 2 – Some of the Second Two Levels of RCI Section 3

In Figure 3 we provide three specific learning outcomes related to RCI 3.4.3 – Pre- and post-conditions, [2].

LO1: State the responsibility of the implementer of an operation with respect to the precondition.
LO2: Examine an operation’s pre- and post-conditions and specify a black-box test case based on those conditions.
LO3: Write the post-conditions for a given operation.

Figure 3 – A Few Learning Outcomes Related to RCI 3.4.3

Note that each of the learning outcomes listed in Figure 3 uses a verb to describe the behavior expected from the student when being assessed.

The learning outcomes in Figure 3 are an example the learning outcomes which have been used to develop student assessments. These assessments have appeared in undergraduate midterm and final exams across at least five semesters. Data from the results of these exams have been aggregated providing us with direct evidence of student learning. None of the direct evidence is based on pre- and post-tests. See [1] for full details.

IV. INSTRUCTIONAL MATERIALS

A number of different types of instructional material were developed to support student learning of various parts of the RCI. These types include: lecture notes, in-class hands-on activities, videos, on-line tutorials, etc.

To support RCI 3.4.3, and specifically LO2 (Figure 3), the Test Case Reasoning Assistant (TCRA) [2] was developed. This is an on-line tool that takes a student through a series of test-case creation exercises and provides the student with immediate feedback. TCRA was developed at Clemson and has been successfully used in an undergraduate software engineering course. It is available free on-line at: www.cs.clemson.edu/resolve/teaching/spec-understanding.html.

V. OBSERVATIONS

Based on direct evidence of student learning collected from our student assessments we were able to make a number of observations [1] about the ability of undergraduate students to learn reasoning principles in the RCI. Figure 4 lists a few of these observations.

- In an undergraduate software engineering course students are capable of learning the RCI reasoning principles taught at advanced levels of difficulty.
- Learning outcomes based on the RCI aid in pinpointing specific areas where students are having difficulties.
- Instructors who have never taught reasoning principles before can teach them well from the outset.

Figure 4 – A Few of Our Observations About RCI-Related Learning

VI. CONCLUSION

Utilizing the RCI, learning outcomes, and instructional materials, we were able to collect direct evidence of student learning relative to the concepts of the reasoning inventory. This evidence supports our thesis that students at the undergraduate level can effectively be taught, can learn, and can apply skills for reasoning about software correctness with respect to the common software written in undergraduate software development courses. Furthermore many of the instructional materials that have been developed (e.g., online tutors [3] and instructional videos) support distance learning in the form of a flipped classroom and hybrid courses.

VII. REFERENCES

