HUMAN TECHNOLOGY

An Interdisciplinary Journal on Humans in ICT Environments ISSN: 1795-6889

www.humantechnology.jyu.fi Volume 6 (1), May 2010, 106–128

USING RATIONALE TO ASSIST STUDENT COGNITIVE AND INTELLECTUAL DEVELOPMENT

Janet E. Burge
Department of Computer Science and Software Engineering
Miami University, Oxford, Ohio
USA

Bo Brinkman
Department of Computer Science and Software Engineering
Miami University, Oxford, Ohio
USA

Abstract: *One of the questions posed at the National Science Foundation (NSF)-sponsored workshop on Creativity and Rationale in Software Design was on the role of rationale in supporting idea generation in the classroom. College students often struggle with problems where more than one possible solution exists. Part of the difficulty lies in the need for students to progress through different levels of development cognitively and intellectually before they can tackle creative problem solving. Argumentation-based rationale provides a natural mechanism for representing problems, candidate solutions, criteria, and arguments relating those criteria to the candidate solutions. Explicitly expressing rationale for their work encourages students to reflect on why they made their choices, and to actively consider multiple alternatives. We report on an experiment performed during a Data Structures course where students captured rationale.*

Keywords: *design rationale, creativity, student cognitive development.*

RATIONALE AS A METHOD FOR BUILDING CREATIVITY AND COGNITIVE MATURITY

One of the orienting questions for the National Science Foundation (NSF)-sponsored workshop on Creativity and Rationale in Software Design in 2009 was, "How can design rationale be used in the classroom to motivate and instruct students about reflection, idea generation, and evaluation?" (Daughtry, Burge, Carroll, & Potts, 2009). At the heart of this question is an implicit claim about creativity, that is, "creativity" in software design seems to involve not just idea generation itself, but also the iterative process that moves the designer to reflect, evaluate, and generate more ideas multiple times before committing to a final design.

Carroll's (2009) workshop manifesto, "The Essential Tension of Creativity and Rationale in Software Design," emphasized this by pointing to liminality as a key aspect of the creative design process. The manifesto described liminality as "Thinking and acting on the border between two contrasting concepts or rules, such as rapid switching between convergent and divergent modes of thinking."

URN:NBN:fi:jyu-20105241909

We see a direct link between a student's cognitive development and the ability to engage in creative processes. Perry (1970) identified nine positions of development starting with duality, where answers exist for everything and where they can be right or wrong, into multiplicity, where all answers are valid, into relativism, where they begin to evaluate solutions based on the context, and continuing through several levels of commitment, where students can begin to integrate knowledge and make their own choices based on that information.

Students in the first two years of college tend to display dualistic and multiplistic tendencies. Though Perry pointed out that most college students are not pure Stage 1 dualists, few students in his study reached even the lowest levels of commitment until their junior year (p. 155). Similarly, Marcia B. Baxter Magolda reported (1992, p. 71) that more than 80% of juniors are "transitional knowers," (those that recognize relativism in some knowledge domains, but are still dualistic in others), and more than 40% of sophomores were still mostly dualistic.

This understanding of the epistemic styles of our students should inform our thinking about teaching design. Dualistic cognition is inherently opposed to the liminal state of mind that is so characteristic of creativity in design. We believe that students who come to a design problem with the attitude that there is a "right answer" to be discovered by analysis will commit to a design without engaging in reflection or iteration. They will commit too early, before they have a chance to be creative. Students in multiplicity or the early stages of relativism may be unable to distinguish between the good designs and poor designs that emerge in their thinking. Some evidence to support these claims can be found in the work of Atman, Cardella, Turns, & Adams (2005), who showed that senior engineering students spend 2 to 3 times more time on a design problem than freshman engineering students. This also correlates highly with the quality of the final solution, though their results do not directly address our claim that these effects are due, in part, to student epistemic styles.

We propose that requiring students to generate design rationale prior to implementing their solutions is a mechanism for encouraging reflection and delaying commitment to their initial design choices. Design rationale, the reasons behind decisions made while designing, is a way to represent design alternatives and the deliberation that produced them. In a sense, the rationale can be considered a language of design (Dym, Agogino, Eris, Frey, & Liefer, 2005), much like sketching (which captures structural aspects of design) or mathematics (which expresses constraints the design must conform to). In the case of rationale, this is a language that captures the design intent and its relationship to the design. The ability to analyze and evaluate design alternatives in terms of their success at achieving design goals (intent) requires higher order thinking skills.

In response to our orienting question, we claim that design rationale help to motivate and instruct students in the creative process by putting off the moment of commitment to a design. The time spent in the liminal phase of design, iterating from idea to evaluation and back, can be lengthened by the use of design rationale. A prospective design rationale (that is, design rationale built before implementation, as a method of exploring possible designs) serves as a way of documenting the designer's process of design.

This lengthening of the time the student spends in ambiguity and reflection should also lead to cognitive development, by forcing the student to experience the kinds of reflection and switching between modes of thought that are characteristic of higher levels of cognition.

In the rest of this paper we explore more fully the following two questions:

1) What are the links between creativity in design and cognitive level, and how can rationale assist in developing a student's capacities for each?
2) How can we assess whether or not use of rationale has had the intended effect?

In the balance of this first section we explore the first question. First we describe the motivation for teaching creativity in software development, and then expand on our proposition that "liminality" links software design with cognitive development. Next we discuss the use of design rationale as a pedagogical tool for encouraging cognitive development through reflection, and describe some prior applications of rationale to education.

Later in the paper we describe an experimental assignment we designed based on our ideas, and provide some initial assessments of our approach. In the final two sections we outline areas for future work and other ways that design rationale may be used to stimulate student cognitive development.

Creativity in Software Development

Software development is, at its core, a creative enterprise. Given a problem, there are many possible solutions. For some practitioners, this is what attracts them to the field—software development as an exercise in creative design. For others, especially as college students, the multitude of solutions, where there is often no clear "right" answer, can be a source of frustration. With the many demands on their time, both curricular and extracurricular, there is significant pressure to find the, or a, correct solution in as little time as possible. The skill of being able to understand just enough about the material to come up with an answer serves them well in some of their earlier courses, where a program is correct if it produces the correct set of outputs given a set of inputs. But they run into difficulty in their later courses, where solutions need to be analyzed on multiple dimensions. These difficulties are exacerbated in courses such as Software Engineering and Human–Computer Interaction, where the system design is influenced not only by the technology available but by how people intend to use it.

It is essential that computer scientists, and computer science students, think creatively in order to successfully develop software. Glass (1995) described several aspects of software development where creativity is critical: determining how to translate the customer/business needs into a problem that the software can solve; resolving stakeholder conflicts; designing solutions to new and complex problems; determining test cases; and enhancing existing systems to meet needs that were not initially anticipated by the customer or the developers. A student convinced of a single right answer is likely to either insist that the stakeholder(s) provide this answer (when the stakeholders may not be approaching the problem with an awareness of what is possible with the technology available) or insist that their solution is the only one, or the best one, even if it may not be acceptable to the client.

Liminality, Creativity, and Cognitive Development

The workshop manifesto (Carroll, 2009) emphasized three major characteristics of creativity in software design: playfulness, empathy, and liminality. We have chosen to focus on the

liminal aspects of creativity because it seems to be the most natural fit for freshman- and sophomore-level courses.

To be sure, many instructors have great luck incorporating playful or empathic approaches in their coursework; many such assignments are presented every year at the SIGCSE (Special Interest Group on Computer Science Education) conference. But, for most students, Data Structures is the first required course that explicitly teaches a set of mathematical tools that can be used to compare one solution to another. Here we are speaking of the use of asymptotic analysis to compare the time and space requirements of data structures. When applied to simple problems, like sorting, such analyses seem definitive: For example, "Randomized Quicksort is more efficient than Insertion Sort." But when designing a data structure for a realistic problem, it is often the case that some operations can only be made fast if other operations are made slow, or if excessive amounts of memory are used, or if auxiliary data structures are used for bookkeeping. This means that, as a data structure is designed, there are many opportunities to shift focus from one operation to another, and to shift from analysis to idea generation and back. The manifesto links this "rapid switching between convergent and divergent modes of thinking" to creativity.

A concrete example will help clarify our point. In the experimental assignment described more fully below, students were asked to design a list-like data structure that needed to support dequeue operations (adding and removing items at the ends) as well as searching by key. One student, in his initial thinking, considered only arrays and linked-lists as possible designs, and selected linked-lists because they support dequeue operations in constant time. Upon evaluation of the designs, however, he discovered that search would be very slow, and so he returned to idea generation and added hash tables as a third design option. Upon evaluating hash tables he discovered that the dequeue operations would be tricky to implement, and returned again to idea generation.

Inspired by this example, we propose that a Data Structures course is a natural place to look for the contrasting concepts that give rise to liminal mental states. In Data Structures we teach the theory of algorithm running times, but also how to actually determine algorithm performance through experiments to confirm (or not!) the theory. We teach the canonical data structures, but we also give students problems for which the canonical data structures are a poor fit. We present the material of the class using diagrams and pseudo-code, but require students to actually write working programs using a real language.

We do not wish to define *creativity* only in terms of liminality, but we feel that much of what is creative about the work of students really arises when they are able to synthesize seemingly incompatible ideas from two apparently opposing or unrelated ways of thinking. In reference to the manifesto (Carroll, 2009), we claim that students are best able to "pursue surprise and unexpected outcomes" when they actively embrace and explore the "border between contrasting ideas."

We believe the ability to embrace liminal states and cognitive development are directly linked. Many useful theories of cognitive development might inform this discussion. We have already described the key aspects of Perry's (1970) model, which undergirds much of our thinking in these early sections. In our final section, we also use Bloom's Taxonomy (Bloom, 1956), which we found helpful in identifying other pedagogical applications of design rationale. The evidence of Perry (1970, p. 55–56) and Baxter Magolda (1992, p. 71) suggests that our Data Structures students (who are mostly sophomore computer science majors and

junior engineering majors) will still be in transition towards relativism. Baxter Magolda's study showed that more than 40% of sophomores were still noticeably absolute in their thinking, and that very few juniors (less than 10%) are independent thinkers. In Perry's study juniors were rated as being in "commitment" (levels 7, 8 or 9) only about 50% of the time, and for sophomores it was less than 10%.

Students stuck in a dualistic way of thinking are unlikely to discover creative solutions, because they will be satisfied as soon as they identify any "correct" solution. The traps for students in multiplicity or naïve relativism are subtler. At this level, the student is aware that there are many viable solutions, but tends to assume that all are equally good. This can again block creativity because the student chooses a solution somewhat arbitrarily. When students are "stuck" at the lower levels of cognitive development, we suspect that the solution chosen is likely to be routine, familiar, or arbitrary, rather than innovative and creative.

So we propose that there is a link between comfort with liminal mental states and cognitive maturity, and that design activities that cause students to experience rapid switching between contrasting ideas help students to build up both cognitive and creative maturity.

Rationale, Reflection, and Liminality

In the experimental assignment sequence presented in the next section, we used prospective design rationale to encourage student creativity in an individual design task. As mentioned above, a prospective design rationale is one that is created before the design is implemented, as part of the design process. Contrast this with retrospective design rationale, which are written after the design is chosen, and may serve only to document the chosen design. Prospective design rationale fosters both creativity and cognitive development by encouraging, and capturing evidence of, reflection.

Reflection serves an important purpose in both education and in practice. In education, many researchers have proposed a link between reflection and cognitive/epistemic level. Dewey (1933, p. 9) defined reflective thinking as "active, persistent, and careful consideration of any belief or supposed form of knowledge in the light of the grounds that support it and the further conclusions to which it tends." Reflection guides the learning process as evidence is examined and conclusions drawn. Dewey's claim that reflective thinking is necessary when it is not possible to come up with "certain solutions" was the reason why King and Kitchener (2002) chose reflection as the basis for their model of student epistemological development. The reflective judgment model (King & Kitchener, 1994) defined seven stages of student epistemological development, broken into three categories: prereflective thinking, quasi-reflective thinking, and reflective thinking.

Schön, in his book *The Reflective Practitioner* (1983), described the need for professionals to move beyond technical rationality, where problem solving is the application of theory, to processes that allow for uncertainty and conflict. He described "knowing-in-action," where practitioners act based on tacit knowledge, and "reflection-in-action," where practitioners reflect on what they are doing as they do it.

Fischer, Lemke, McCall, & Morch (1991) described how design rationale supports reflection by capturing the designer's knowledge about the situation. Similarly, the iteration between idea generation (divergent thinking) and design selection (convergent thinking) is a reflective process. Design rationale supports both the capture of the alternatives and their

exploration by supporting the evaluation of the more promising alternatives and any additional decisions required during their elaboration. In the illustrative example above we saw a student using the process of building design rationale as an opportunity for critical reflection.

We should note, as an aside, that in this study we focus only on the individual design projects, not on teamwork. Though we greatly appreciate the role that rationale can play in capturing and transferring knowledge in a team setting, we believe that the capture of rationale is beneficial even for one individual engaged in an individual design project.

So we claim that design rationale can be used to encourage critical reflection about software design problems. Further, we claim that such critical reflection, if embraced by the student, is likely to lead to greater creativity. Critical reflection and creativity are certainly not the same thing; rather, critical reflection tends to provide grist for creative energies to act upon. Incorrect assumptions tend to act as roadblocks for creativity, but critical reflection helps us to challenge these assumptions. We naturally tend to select designs similar to older successful designs with which we are already comfortable, but critical reflection can cause us to reject familiar solutions that are actually inappropriate.

Prior Work on Rationale in Education

Moran and Carroll's (1996) book included two approaches to using rationale in education. The first was to provide rationale in the form of templates to assist with user interface (UI) design (Casaday, 1996). The templates help designers to "ask the right questions" and assist designers with the process by guiding them toward a solution. Carey, McKerlie, & Wilson (1996, p. 375) built a library of "exemplary user-interface designs" along with their rationale so those examples could be used to teach UI design. Other work using rationale in UI design includes using design space analysis (DSA; MacLean, Young, Bellotti, & Moran, 1991) as part of the FLUID (framework for learning user interface design) interactive media system (van Aalst, van der Mast, & Carey, 1995). The work proposed here uses a more general approach (not one aimed at a specific type of design) and supports additional manipulation and evaluation of design criteria, as well as using rationale to assist with the definition and documentation of new designs.

Several software engineering textbooks either teach rationale (Bruegge & Dutoit, 2004) or use rationale as explanation for design case studies (Fox, 2006). Rationale is also present in the form of "consequences" in the ubiquitous Gang of Four (GoF) design patterns book (Gamma Helm, Johnson, & Vlissides, 1995), used both as a reference and as a supplemental textbook.

EXPLORING RATIONALE IN A DATA STRUCTURES COURSE

In the previous section we claimed that careful use of design rationale by dualistic and multiplistic thinkers should lead them to increases in creativity and cognitive maturity. This theory has implications for how one structures "design" projects for lower-level courses. In this section we will present a first attempt at such an assignment for a Data Structures course, and contrast it with the kinds of design assignments we had used in the past.

Our theory also requires some justification through evidence. We have some initial results based on our evaluation of the work produced by students for our experimental assignment

using design rationale. While the experiment was by no means a controlled experiment, nor was it designed to validate our theory (it was, instead, designed to help the students learn), we still are able to report on some tantalizing results that point the direction for future work.

SEURAT and Pugh's *Total Design* in Data Structures

In the Data Structures course, we chose to use two different methods for capturing prospective design rationale. The first was the rationale management system SEURAT and the second was based on examples from Pugh's (1991) *Total Design*. In order for the reader to understand how we think rationale should be used in undergraduate courses, we must first describe what data these two types of design rationale capture, and how they support decision making.

Let us start with some general observations. Problem solving can be broken into four stages: problem definition and analysis, idea generation, idea evaluation and selection, and implementation of the selected idea (VanGundy, 1981). Rationale can support some idea generation techniques, such as brainstorming, by representing alternatives as generated, and attribute listing, a technique developed by Crawford (VanGundy, 1981), where attributes listed would be alternatives. Rationale captured in the form of argumentation is especially useful, however, during the evaluation and selection stage by capturing criteria, their relationship to the alternatives, and supporting evaluation. Some of the techniques described by VanGundy (1981) that could be supported by rationale are (a) the advantage–disadvantage approach, enumerating the advantages/disadvantages of each alternative with respect to a predefined set of criteria; (b) the Battelle method (Hamilton, 1974; VanGundy, 1981), dividing criteria into culling, rating, and scoring in order to narrow the field of alternatives; and (c) reverse brainstorming (VanGundy, 1981; Whiting, 1958), which is brainstorming on the disadvantages of each alternative. Rationale systems that perform evaluation, such as the software engineering using rationale (SEURAT) system (Burge & Brown 2004), can be considered a type of weighting system (VanGundy, 1981), by allowing weights to be assigned to the criteria and using those weights in evaluation.

In this work, we use argumentation-based rationale to capture the idea generation, idea evaluation, and selection stages of problem solving. We used two methods for representing rationale, SEURAT (for one experimental group) and written documents proposed in Pugh's total design methodology (for the other). Both methods require students to list many alternative designs, develop criteria by which to evaluate the designs, perform the evaluation, and select a solution. Both methods, furthermore, require argumentation to back up both the evaluation criteria and the final decision. SEURAT adds the additional capabilities of expressing the rationale in a hierarchical format, showing decisions and subdecisions, as well as providing the capability to calculate a numerical evaluation of the support for each alternative.

Software Engineering Using RATionale (SEURAT)

The SEURAT system (Burge & Brown, 2004) is a rationale management system (RMS) originally developed to assist with software maintenance by providing ways that the rationale could be used beyond just its presentation. SEURAT captures rationale as structured argumentation (decision problems, decision alternatives, and arguments) and uses both the structure of the rationale (syntax) and the content (semantics) to inference over the rationale

to detect incompleteness (of the rationale) and inconsistency (of the design). The arguments in SEURAT can refer to system requirements, desired qualities, assumptions made, and relationships between alternatives. Figure 1 shows some rationale captured in the SEURAT Rationale Explorer. SEURAT stores the rationale in a relational database, allowing the rationales to be shared between multiple users during collaborative decision-making.

Figure 1's example shows three decisions, taken from the rationale for a conference room scheduling system. The decisions are displayed using a diamond-shaped icon containing a double-headed arrow. The second decision, "How do we know what the conference rooms are?" has a warning icon overlaid on it. This is because the alternative selected, "ascii file giving a list of room names," is not as well supported as the other candidate alternative, "serialized vector of room objects." The third decision, "How is the user associated with the meeting," has an error icon because none of the proposed alternatives has been selected yet.

The students who used SEURAT in the experiment were given a tutorial on how rationale are entered into SEURAT and how they could use SEURAT's ability to evaluate alternatives to assist them in their decision-making. The students were instructed to enter the functional and nonfunctional requirements that applied to the problem they were solving and then to enter the decisions, alternatives, and arguments. They were instructed to use their requirements in arguments, rather than utilizing the other types of arguments supported by SEURAT.

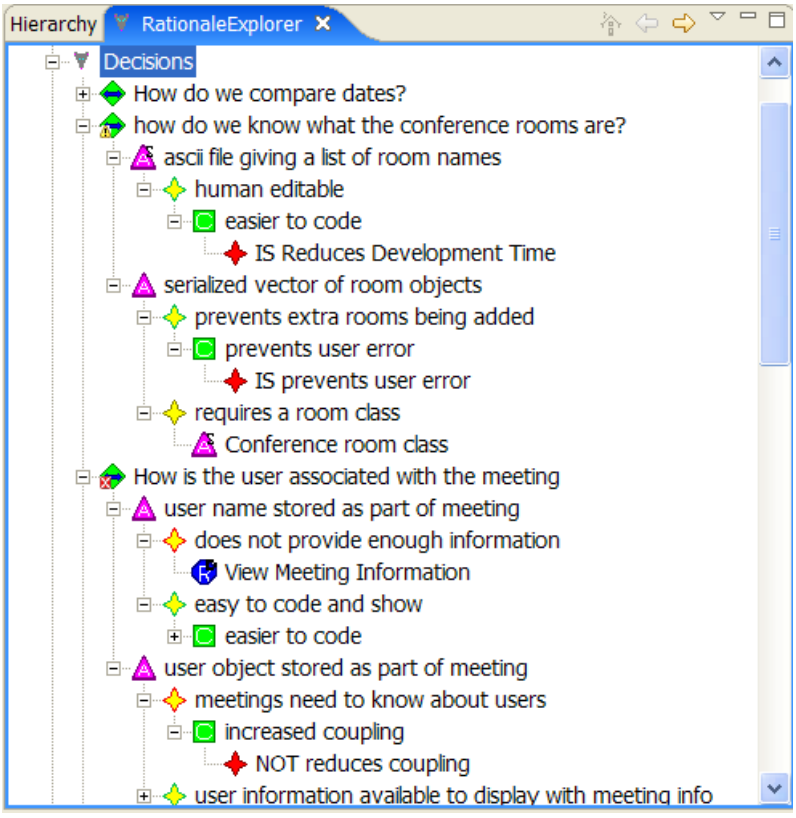

Figure 1. An image of the SEURAT Rationale Explorer, showing the hierarchical view of a structured design rationale.

Pugh's *Total Design*

The other section of the class used a set of design documents based on Stuart Pugh's (1991) book *Total Design*. It is not proper to say that we used "his" documents, because Pugh goes to great lengths to show many different types of documents that might be useful. In particular, we took advantage of three main parts of his approach:

1. A product design specification (PDS) listed the criteria by which designs should be judged. Each student constructed his or her own list of criteria, resulting in a bulleted list with argumentation that was very similar to the SEURAT group, but created in a word processor instead of the SEURAT RMS.
2. A Pugh Matrix was used for idea evaluation. The student created a two-dimensional table in a word processor. Each column corresponded to one of the designs, and each row to one of the criteria from the design specification. The student selected one design to be the "baseline" design, and then each design was compared to the baseline in each criterion. A plus symbol was entered in the table if the design in question was superior to the baseline on that criterion, a minus symbol if it was worse. We also instructed students to enter their arguments in support of their evaluations in each cell.
3. A short essay summarized the idea selection phase and provided arguments in favor of the student's solution, based on the evaluation in the Pugh Matrix. Note that Pugh was very clear that one should not just count up the number of plusses and minuses and then select based on the numerical answer that results. We instructed students to use their evaluation matrix to support their selection process, but to also use common sense.

Pugh's process has many commonalities with SEURAT. Design criteria are explicitly listed, and require argumentation. Designs are evaluated based on the criteria, and arguments supporting those evaluations are captured. The Pugh process supports a quasi-quantitative approach to selecting the final design.

There are major differences as well. On the negative side, SEURAT requires students to learn a new tool (the Rationale Explorer, which is a plug-in for the Eclipse Integrated Development Environment) instead of using a familiar tool (Microsoft Word). On the positive side, SEURAT forces students to be more careful about linking criteria with design decisions. In the non-SEURAT group, some students used arguments in their Pugh Matrix that had no relationship to their criteria, something that is much harder to do in SEURAT. SEURAT also naturally leads one to represent subdecisions in a hierarchy under the main decisions, much like an outline. The Pugh Matrix places all decisions at the same level.

Design in Data Structures Before Design Rationale

For several years we have had design projects in Data Structures similar to the one described here. In the past, however, students simply submitted a retrospective design justification. These documents took a variety of forms, but none of them were particularly formal, and only in very rare cases did the students compose them before implementing their solution. We found the quality of the resulting programs written by the students to be quite disappointing. In particular, there was some anecdotal evidence that students would not consider all of the important design

criteria at the start of the project, but focus on only a few. Theses students tended to select familiar or canonical data structures because they never discovered the trade-offs involved in the real problem until *after* the solution was implemented.

Our feeling was that by introducing design rationale, and specifically prospective design rationale, into the Data Structures course we could cause the students to delay committing to a solution, and give them more opportunities to fully understand the problem.

Data Structures Class Experiment

Our main goal was to explore whether or not rationale could be of benefit to students in 2nd-year computer science coursework. As noted in the manifesto (Carroll, 2009), it could be that use of rationale would "limit creativity by anchoring thought"; it could also be that rationale would be viewed as busywork, or that time spent building the rationale would take away time from honest reflection and other creative activities. We expected, however, that students would actually spend more time in reflection if they had to build a full rationale than if they simply had to write a brief essay explaining their choice.

We designed a classroom activity in which students needed to design a solution to a data structures problem based on their understanding of the performance characteristics of various common data structures. The assignment was broken into 5 steps (see Table 1), each of which had its own delivered artifact.

The fifth step of the process is also meant to test whether students overcommit to their chosen solution, and refuse to change when criteria change.

The experimental subjects were 38 students (34 male, 4 female) in an undergraduate course on data structures and data abstraction (most students were in their 2nd or 3rd year of college). The difference between the two experimental groups was in Step 3. The first experimental group (henceforth the "SEURAT group") constructed design rationale using the SEURAT system. The second experimental group (henceforth the "Matrix group") used a version of Pugh's total design methodology (Pugh, 1991, Section 4.8).

Table 1. Description of the Main Problem and Stages of the Assignment Used in the Experiment.

Problem:	Design a list-like data structure that supports the following operations: Adding and deleting at the head and tail of the list, searching to find the index of the first data item matching a search term, and retrieving an item based on its index in the list.
Step 1:	List the criteria that you want your solution to adhere to. For example, do you want to have constant time searching? Do you want to try to minimize time spent coding?
Step 2:	Make a list of possible alternative implementation strategies. For example, a linked list would support all the operations, though not very efficiently. A hash table, on the other hand, can be made to be very efficient, but most students would find implementing it to be too challenging.
Step 3:	Create a design rationale expressing the tradeoffs between various alternatives in terms of how well they meet your criteria.
Step 4:	Select one of your alternatives, and implement it.
Step 5:	Write a paragraph explaining which alternative you would have selected if the "most important" criterion was removed.

We collected three artifacts from each participant: Their rationale (generated in Steps 1–3), their computer program (generated in Step 4), and their paragraph explaining their response to changing criteria (generated in Step 5). Note that, for the SEURAT group, the rationale could be fully captured in SEURAT, but the rationale for the Matrix group consisted of a list of evaluation criteria (with argumentation), a list of possible designs, and an evaluative matrix (henceforth the "Pugh Matrix"). Table 2 provides the metrics used to evaluate both sets of rationale in terms of rationale quality, and Table 3 provides the metrics used to evaluate the ideation skills demonstrated.

This first set of metrics, R1–R4, is meant to judge student success on the assignment in terms of their mastery of course objectives, as defined by the instructor. A score of 3 points or 2 points indicates that the student met instructor expectations, 1 or 0 indicates failing to meet expectations.

R1, R2, R3, R5 and R6 evaluate the rationale. R4 evaluates the response to changing criteria, and R7 evaluates the computer program code.

Examples of Student Artifacts and Reflections

In order to make this discussion more concrete, we present some small examples of student work. We will show some examples of creative designs from the experiment, as well as some examples of student argumentation.

What Kind of Creativity is Expected/Possible in Data Structures?

First, we wanted to provide some examples of creative solutions to the design problem. Recall that the student needed to design a list-like data structure that supports adding to the head and tail, looking up items by index, and searching for the first occurrence of a particular item.

Table 2. Data Structures Assignment Learning Metrics.

Metric	Excellent/High (3 pts)	Good/Medium (2 pts)	Poor/Low (1 pt)
R1: Are all relevant alternatives identified and provided?	The student provides all the relevant alternatives	The student provides most of the relevant alternatives	The student only produces one alternative
R2: Are the criteria appropriately mapped to the alternatives?	The student maps all the criteria to the correct alternatives	The student maps most of the criteria to the correct alternatives	The student does not successfully map criteria to alternatives
R3: Did the student select an alternative based on the rationale?	The student selects an alternative based on the level of support	The student selects some alternatives based on the level of support	The student did not appear to have reasons for making the selection.
R4: Did the student change the decision after the criteria change?	The student looks at differences in support levels and changes the decisions	The student sometimes fails to change the decision but instead stays with the initial plan	The student did not acknowledge the effect of changing criteria

Note: Each student received a score between 3 (for excellent) and 0 (for incomplete).

Table 3. Data Structures Assignment Ideation Metrics.

R5: Completeness	For each alternative in the following list, the student receives 1 point: Array, Linked List, Vector (or Array-List), Skip List, Hash Table, and Binary Search Tree. These are all of the data structures studied in the class (to that point) that would have been reasonable alternatives for the assignment. This scale is meant to measure the *quantity* of a student's candidate solutions (Shah, Smith, & Vargas-Hernandez, 2003). The instructor made a list of all canonical data structures that would have been useful in the assignment, and awarded one point for each. Students did not receive multiple points for minor variations on each data structure, so this scale does not count absolute quantity, but the quantity of "different enough" design candidates.
R6: Creativity	For each alternative in the following list, the student receives either 1 point or 0.5 points: Skip List (1 point), Binary Search Tree (1 point), Linked Lists with multi-item nodes (1 point), Extra pointers to speed up list traversal in a linked list (0.5 points), Pre-allocation of nodes for a linked list (0.5 points). These are all of the ideas that students came up with that did not come directly from lecture. Significant ideas received 1 point, and less useful ideas 0.5 points. This scale is meant to measure the *novelty* of student solutions, and our approach is very similar to that of Shah et al. (2003). In this case the instructor took a list of all design alternatives submitted by students, and eliminated those that appeared in most or all student submissions. The instructor then assigned point values to the remaining novel solutions based on how different the solution approach was from the non-novel approaches.
R7: Contest rank	Student solutions were ranked based on three speed tests. These three tests were given to students as part of the assignment description. As part of their analysis, they had to decide how heavily to weight these speed criteria, compared to other criteria such as ease of coding. This scale is meant to measure the *quality* of student solutions. Students received an ordinal ranking in each speed test, and then final rankings were based on a standard sum of ordinals. So, for example, a student that received 1st in two tests and 3rd in the last (sum of ordinals is 5) would beat a student that placed 2nd in all three tests (sum of ordinals would be 6).

We have already given the start of an example in the Liminality subsection above. Our problem allows for a very wide variety of valid approaches. The most comfortable approaches would have been to use an array (the main data structure used in previous classes) or a linked list (which they had used on the previous assignment). Students had also seen the approach of leaving spare space at both ends of an array to cut down on the time needed for adding and removing items, and hash tables. None of these approaches were optimal for all operations: Linked lists are slow for searching and indexing, arrays are slow for searching, and a naïve use of hash tables would result in either fast searching or indexing (depending on whether one uses the value or the array index as the hash key), but not both. Also, many students considered their own programming abilities when selecting a design, and so leaned toward array- or linked-list-based solutions because these solutions tend to be easier to read, easier to program, and easier to debug.

The most creative students found synergistic combinations of the canonical approaches.

- One student combined arrays with linked-lists to get a solution that had faster indexing, but which was similar enough to the linked-list he had previously written

that he felt confident he could complete it correctly. He changed his linked-list nodes to contain arrays of length 1000, which made his index-based lookups several hundred times faster than students with a regular linked list.

- Several students discovered that they could achieve better performance by keeping two separate data structures, one for searching by value and another for index-based lookups. One student had an array and a hash table, and another had two hash tables. One student attempted to combine a binary search tree (something he learned in high school) with an array.
- Two students kept auxiliary pointers to the middle of their linked-lists to speed up index-based lookup. They (and others) had considered skip-lists as a potential design, but eliminated them as an option for being too complicated.

On what grounds do we call these creative solutions? These students all found ways to combine apparently contrasting approaches. This is a form of liminal thinking, and also suggests that they returned more than once to idea generation. There is plenty of room for creativity in data structures classes, because even relatively small problems tend to fit the canonical data structures poorly.

Excerpts from Student Rationale and Argumentation

We also want to present a few concrete examples of student use of rationale, and reflections on rationale in our Data Structures course. Our goal here is to briefly indicate to the reader the type of argumentation and rationale that students produced. One should not try to make any general conclusions from the three anecdotes presented here, but instead we feel this should provide a bit of clarity in our discussion of assessment below.

Several students explicitly commented on the way that using rationale affected their performance on the assignment. One student from the SEURAT group said,

> *… I kind of went in biased towards a Doubly Linked list with a Hash Table … [but] the Hash-backed Array list still came out on top. This is because, while Doubly Linked list has a faster add/remove time, the Array-list has a much faster lookup by index time. … The design rational helped me visualize. Without this tool, I might not have fully realized that problem until it was too late.*

Interestingly, this student's Rationale (see Figure 2) did not take advantage of SEURAT's hierarchical decision-making capabilities, but did make use of its evaluation affordances.

A student in the Matrix group similarly noted that,

> *The analysis part of this report helped me pick this option. Doing the analysis allowed me to compare different options with each other to see the advantages and disadvantages of each. In the end the requirements that I found most important to deciding which option to go with included having a very fast way to search through the data structure, and having am* [sic] *option that was relatively easy to code.*

Though the student's argumentation is very brief (see Figure 3), it captures key differences between the various options. The instructor is able to see that the student thought through all the criteria, and had reasons (even if incorrect or naïve) behind the choices made.

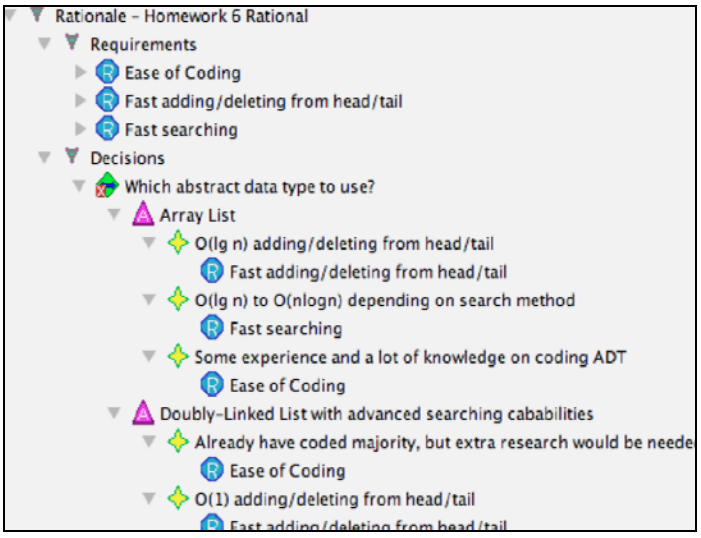

Figure 2. Excerpt from a student's SEURAT rationale.

Of course, there were some students critical of the use of design rationale. One student in particular commented about SEURAT that,

> *My design rational* [sic] *helped me … but in the end I don't think I will agree with it. … I could have figured this process easier by just writing this all out on paper, … I looked at my decisions and realized what was most important to me in this project, learning about the data structures. (I didn't put this in the calculations, so maybe they would be different...).*

What the student is saying here is that he chose not to include his most important decision criteria in his rationale and, as a result, the design rationale did not support his eventual decision. Several students struggled because they believed that only technical criteria should be present in design rationale. For a homework project in an undergraduate course, however, the controlling criteria may be completely nontechnical, just as in industry. Most students in the course seemed to understand this, and were willing to include nontechnical concerns (like time available to code or educational goals) in their list of criteria.

Assessing the Results of the Data Structure Design Assignment

The main purpose of this assignment was to stimulate student creativity and critical reflection through the use of design rationale. It will be quite clear to the reader that this was not an experiment designed to validate our theory, but rather a first attempt to put our ideas into practice.

Requirements	Option 4 *Array-backed list with doubling approach*	Option 5 *Array-backed list with double and dictionary ADT*
The program is simple to code	Harder than the linked lists	The hardest option to code
The program is commented well	Easy to comment	Easy to comment
addToHead is fast	addToHead is nLog(n) time	addToHead is nlog(n) time
addToTail is fast	addToTail is nlog(n) time	addToTail is nlog(n) time
removeFromHead is fast	removeFromHead is nlog(n)	removeFromHead is nlog(n)
removeFromTail is fast	removeFromTail is nlog(n)	removeFromTail is nlog(n)
Search is very fast	Search is linear	search is constant
The program is robust	Harder than linked lists	Hardest to make robust
The list scales well	Methods do not scale as well as option 3	Search scales well, others methods don't
Finding nth node	time constant	time constant

Figure 3. Excerpt from a student's Pugh Matrix.

Nevertheless, we provide some assessment of the results of the assignment. We provide some evidence that using a RMS (as opposed to simply written rationale) did not negatively impact student performance. We also tried to gauge the amount of creativity exhibited by students in the project.

Experimental Results: Student Success and Rationale

As described above, we evaluated student designs using seven rubrics. The first four rubrics (R1–R4) assessed student success in assignment tasks, and were initially rated on a scale from 0 to 3 independently by the two authors. In cases of disagreement, the authors consulted and reached a consensus rating. Table 4 shows the results for each of the standard scale metrics.

The students showed success as measured by metrics R1, R3, and R4, but were less successful with mapping criteria to alternatives. Lacking a control group, we cannot prove that using design rationale helped students develop their criteria and alternatives, but we can observe that almost all met instructor expectations on these tasks, indicating that the design rationale were not an impediment to the intended learning. From the weak scores on R2 (mapping criteria to alternatives), it appears that students were weakest in the analysis phase of the assignment, which is not surprising for students at this level.

Experimental Results: Comparison of SEURAT to Matrix Results

The use of Pugh-style matrices is well established in engineering design. We wished to evaluate whether or not using SEURAT in undergraduate classes was a supportable approach. In order to do this, we considered seven hypotheses that compare student performance when

Table 4. Results for Student Success Metrics.

		SEURAT	Matrix	Average of both groups
R1: Are all relevant alternatives identified and provided?	%Excellent/High	**68**	**53**	**61**
	%Good/Medium	32	37	34
	%Poor/Low	0	10	5
R2: Are the criteria appropriately mapped to the alternatives?	%Excellent/High	47	29	38
	%Good/Medium	41	35	38
	%Poor/Low	12	35	24
R3: Do the students select an alternative based on their rationale?	%Excellent/High	**83**	**79**	**81**
	%Good/Medium	6	11	8
	%Poor/Low	11	11	11
R4: Do the students change their decision after the criteria change?	%Excellent/High	**83**	**74**	**78**
	%Good/Medium	6	5	5
	%Poor/Low	11	21	16

Note: $N = 38$, and some columns sum to 99% or 101% due to rounding.

using SEURAT to student performance when using the Pugh Matrix method. If students using Pugh matrices did substantially better than students using SEURAT, we might conclude that SEURAT should not be used with younger students. However, this turned out not to be the case: Students using SEURAT performed as well as, or better than, students using Pugh matrices in most tasks.

We used a two-sided Mann-Whitney U test to compare experimental groups (see Table 5). Because we have no reason to believe that any of our rubrics would correspond to a normal distribution, we felt that it would be unsound to use, for example, a *t*-test, because it requires the sampled data to be independent and normally distributed. The Mann-Whitney test does not suffer from this defect because it works by first ranking all samples, and then evaluating the likelihood of there being a marked difference in rank sum between the two experimental groups. More information about Mann-Whitney U tests (also known as Wilcoxon rank sum tests) may be found in statistics textbooks, such as Rice (1995, pp. 402-410). We set our threshold for significance at the $\alpha = 0.1$ level.

Table 5. Comparison of SEURAT Users to Matrix Users.

Alternative hypothesis	N_S	N_M	S_S	S_M	Test result
Students using SEURAT are more likely to present all the relevant alternatives than those using the Pugh method. (R1)	19	19	336	405	Null hypothesis accepted, $\alpha \approx 0.32$
Students using SEURAT are more likely to correctly map criteria to alternatives than those using the Pugh method. (R2)	19	19	322	419	Null hypothesis accepted, $\alpha \approx 0.16$
Students using SEURAT are more likely to select an alternative based on their rationale than those using the Pugh method. (R3)	19	19	369.5	371.5	Null hypothesis accepted, $\alpha \approx 0.98$
Students using SEURAT are more likely to change their selected alterative after criteria change than those using the Pugh method. (R4)	18	19	343.5	359.5	Null hypothesis accepted, $\alpha \approx 0.60$
Students using SEURAT will have a more complete set of alternatives than those using the Pugh method. (R5)	19	19	322.5	418.5	Null hypothesis accepted, $\alpha \approx 0.16$
Students using SEURAT will have a more creative set of alternatives than those using the Pugh method. (R6)	19	19	386	355	Null hypothesis accepted. (Note : this shows a negative correlation) $\alpha \approx 0.66$
Students using SEURAT will do better on instructor-defined performance criteria than those using the Pugh method. (R7)	17	18	246.5	383.5	Null hypothesis rejected, $\alpha \approx 0.05$*

Notes: N_s is the number of samples in the SEURAT group, and S_S is their scaled rank sum. N_M is the number in the Matrix group, and S_M is their scaled rank sum. α is a numerical approximation of the probability of rejecting the null hypothesis when it should be accepted, based on a two-sided Mann-Whitney U test. Note that since the lowest rank is best (1st place is better than 38th place), the *smaller* scaled rank sum indicates better performance.
* Statistically significant

We must take some care in interpreting these results. In particular, look at our result for R7. This measures the speed of the student's solution: The instructor gave students some speed-related criteria at the start. Students were free, however, to reject these criteria and instead focus on criteria such as ease of coding, ease of debugging, re-use of code, and other similar criteria that are contrary to high scores in R7.

One major threat to the validity of this assessment is due to the way the experimental groups were assigned; the SEURAT group comprised all students from one section of the course, while the Matrix group composed the other section. The SEURAT group was stronger than the Matrix group as measured by homework grades on assignments other than the experimental assignment. It is possible that higher ability levels of the SEURAT group masked difficulties with using SEURAT that would have been identified if experimental groups were allocated in a more careful way. Furthermore, some have suggested (e.g., Amabile, 1983) that technical expertise is a key factor that enables creativity. This would mean that the SEURAT group might be expected to be more creative than the Matrix group, on these sorts of tasks, simply due to their increased technical proficiency.

To try to correct for this problem we computed a best-fit line between each experimental variable (R1-R7) and final homework grade, and then analyzed the residual values that result when subtracting the predicted values from the actual values. The residual values essentially tell us how much the student was over- or under-performing on this assignment compared to his or her usual performance in the class. When comparing the groups using these residuals,

- The groups were still not significantly different for rubrics R1, R2 and R4,
- The differences in R5 and R7 were no longer significant, and
- The Matrix group outperformed the SEURAT group in R3 and R6, with levels of $\alpha \approx 0.07$ in both cases

Again, we are not making any strong claims about the validity of this approach, but we felt that in the interest of completeness, as well as fairness to the Pugh method, we should present a grade-corrected version of the results.

Experimental Result: Rationale and Creativity

One other interesting trend is the very strong negative correlation, -0.4906, between R1 (Consider all relevant alternatives) and R6 (Consider a more creative set of alternatives). While we cannot make any claims about statistical significance, this relationship seems an intriguing topic to explore in future research.

We hypothesize that this results from some preconceptions among the study participants about the number of alternatives that the instructor expected them to generate. In particular, the instructor indicated that each student should have "at least 4-5 alternatives."

Some commentators have suggested that rationale might be inherently contrary to creativity (see Carroll, 2009). We do not view our results as supporting that claim, because we believe the key problem was the process by which students decided whether or not they had considered "a sufficient variety of alternatives." Students may have been rushing to escape the liminal/undecided state, and so simply stopped when they had four alternatives. We believe that this is not a problem inherent in rationale, but instead a mistake in the instructor's design of the grading rubric for the assignment.

Summary of Claims

Because of the nature of the experiment, we want to be very careful to precisely state what we think our experiment shows. First, our experiment gives some evidence that using a RMS instead of a more traditional (writing) assignment does not negatively impact student learning. Though it now seems obvious that this would be the case, the course instructor initially had serious reservations about using SEURAT in class.

Second, the somewhat weaker scores in evaluating alternatives seem to support our claim that students in the course have not reached the "commitment" stages (levels 7 to 9) of Perry's (1970) scheme. On the other hand, we would be able to obtain much better data on this topic by giving the subjects appropriate critical thinking inventories, and we should do this in the future.

Third, our results suggest future experiments on rationale and creativity in education must be more careful about the instructions given to students about how to assess their own idea generation process. It appears that the use of rationale did indeed inhibit creativity, but probably primarily as a result of a poorly designed grading rubric, which focused on quantity instead of variety. If our theory, that increasing time spent in a liminal state increases creativity, is correct, then it might make more sense to require students to spend a predetermined amount of time on idea generation, instead of aiming for a predetermined number of ideas.

REFLECTIONS ON VERIFYING THE CONTRIBUTION OF RATIONALE TO CREATIVITY IN THE CLASSROOM

In this paper we proposed a theory that use of design rationale with 2nd-year computer science students should lead to improved creativity as well as cognitive development. We reported on an experimental assignment that put these ideas into practice in a Data Structures course, and we reported on our assessment of the impact of this intervention on student learning. It is clear, however, that there are many lessons to learn from the first attempt about *how* one ought to try to verify the effects of such an assignment.

First, we wished to make claims about student cognitive levels or epistemologies. At the most basic level, we assumed that students in our class would exhibit some dualistic tendencies, and that few would be contextual or committed knowers. We are particularly interested in how students at different levels of cognitive development respond to the challenge of creating design rationale, but there was no way for us to assess this because we did not collect this data. In the future we are considering applying epistemic inventories, such as Baxter Magolda's (1992) measure of epistemological reflection, as well as attempting to develop an inventory that specifically measures the student's epistemology of knowledge as it relates to software design. We hypothesize that students in transition or multiplicity may view relativism as normal in humanities classes, but not in engineering classes.

It is unlikely, however, that such inventories can directly assess the impact of our intervention on students, even if given as both pretest and posttest. Because students take many courses, most of which aim to increase student cognitive level, it seems unlikely that the effect of our intervention can be teased out from such data. In the future we should directly ask the students questions about why they did what they did and how they made decisions on our particular assignment. The examples of student reflection presented in

earlier in this paper were suggestive, but we did not systematically ask students to comment on their processes. So we only have such data for a very small number of students.

Similarly, general-purpose measures of ideation similar to those used by Shah et al. (2003) should be adopted to make our results on creativity more comparable with results presented by other researchers. This would still leave us, however, with the problem of discovering *why* the student produced the set of design ideas that he/she produced. This again requires some qualitative methods that we did not use in our initial assessment. We need to question students, either on paper or through interviews, as to why and when they stopped generating ideas. If, for example, they did not start building their design rationale until the hour before it was due, we should not be surprised that the alternatives discussed were canonical or familiar. An approach similar to the verbal protocol analysis used by Atman et al. (2005) would be most useful.

FUTURE WORK—RATIONALE ACROSS THE CURRICULUM

The experiment described in this paper focused on one group of students, those taking the Data Structures course (typically sophomore computer science majors and junior engineering majors). If the goal is to aid student cognitive development, as shown by their progression through the levels of the Perry (1970) scale, appropriate exercises and evaluation measures need to be applied at multiple points throughout the curriculum and, ideally, cognitive development evaluated both as an aggregate over all students and for individual students as they pass through the program.

This would require that the exercises be targeted to specific stages of development. At the earlier stages of their education, students can be presented with the problems, candidate solutions, and the rationale. For example, in a Data Structures course, the students learn many different ways to represent collections of objects. The student focus is often on how to implement these collections. The implementation is certainly important, but since many of these constructs are often supplied with the programming language (and do not require implementation), it is often more important that the students understand why they might choose a particular data structure for a problem, that is, to *analyze* the possible solutions by determining which criteria are relevant to the specific problem. The students' emphasis on the implementation rather than the selection becomes apparent in later classes, where they tend to stick to one or two favorite structures that may or may not be the best choices for the problem at hand. Providing rationale in a form that can be easily understood and manipulated may be a more effective way to teach the students the tradeoffs involved in selecting between data structures. The ability to manipulate the argument criteria can also help the students to explore how changing priorities result in different preferred solutions. Rationale can be presented in a form where the criteria can be manipulated by modifying their relative importance in order to demonstrate how as criteria change, so should the recommended solution.

When the students are comfortable with the idea of multiple alternative solutions, the next step would be to involve them in exercises where the problems and criteria are provided but where the students need to identify (*synthesize*) the candidate alternatives based on what they have learned in class and on their own experience. An example of this would be if

students were asked to provide alternative methods for data entry or visualization based on usability criteria that they have learned in an HCI course.

The ability to identify the problems themselves, propose solutions, and define criteria requires *evaluation*—identifying what aspects of the solutions are important to the problem and its context. This is an essential skill in both software requirements analysis and in design. The requirements elicitation process is one of defining the problem and the criteria under which the solution will be evaluated, while the design process involves identifying and selecting solutions to that problem.

The movement from dualism through multiplicity and into relativism and commitment is more of a challenge. Kloss (1994) recommends several strategies to move students from dualism towards relativism that stress the importance of analyzing and structuring different points of view. This requires looking at the alternatives and evidence, including understanding the role of assumptions. As students move from working with the rationale of others to producing rationale themselves, the rationale can serve as both an instructional tool and as a means of assessing their intellectual development.

Table 6 lists the levels of the Bloom Taxonomy (1956), how the reflection and rationale approach should support those levels, and how students at different levels of development, as measured by the Perry (1970) scale, would perform on the rationale-supported tasks.

Three courses in our department curriculum now contain explicit course outcomes regarding analysis of multiple alternatives: CS2 (1st year), Data Structures (2nd year), and Senior Design Project (4th year). Using rationale within these courses will provide an opportunity for studying how rationale can assist in the students' progression from dualism toward the higher levels of development.

SUMMARY AND CONCLUSIONS

Students progress through several stages as they gain knowledge and experience. They run into difficulty when they need to operate at higher cognitive levels than they are accustomed to. The ability to make decisions when confronted with uncertainty and ambiguity is important since the problems they will tackle become more realistic and beyond the point where, if they perform the right sequence of actions, they can produce a single correct answer. The ability to synthesize and evaluate solutions becomes critical for problem solving and creativity. Related to this is the need for students to move beyond duality, where there are always right and wrong answers, towards higher levels of thinking where they can begin to analyze the evidence and understand that not all criteria are equally valid in every context.

Our experiment with using two rationale representations, the SEURAT RMS and Pugh's (1991) total design methodology (as part of a writing assignment), indicated that students at the college sophomore level do indeed start out at a fairly low level. The experiment suggests that using the RMS does not inhibit creativity when compared to results using the more traditional writing assignment. Using rationale, however, did not result in a wider variety of ideas. This could be because the students were told how many ideas were required and some may have stopped searching once they achieved their "quota."

While not giving definite answers on rationale's impact on creativity, there were insights to be gained from the use of rationale. The rationale provided insight into student thinking for

Table 6. Relating Bloom's Taxonomy, Reflection and Rationale, and the Perry Scale.

Bloom	Reflection and Rationale	Perry
Knowledge	Given a general decision problem, list and define the alternative solutions, as described in class. Example: What data structures can be used to store lists of items?	Students at all levels should be able to do this since it could be directly recalled from their lecture notes.
Comprehension	Given a general decision problem, and a set of criteria for making a selection, explain why these criteria are important. Example: Why is it important to be able to efficiently remove elements from a list of items? Given a set of alternatives for a general decision problem, differentiate between them. (This may require giving the students the criteria). Example: What is the difference between two data structures that store lists of items?	Students at all levels should be able to explain the criteria, but students still in the dualism stage may show biases toward certain alternatives (the "right" one) when differentiating between options and may not explore them in detail.
Application	Given a specific decision problem, give a list of possible solutions. Example: Given a design that requires sorting and searching a list of items, which data structures could be used to solve it?	Students in dualism may have difficulty providing more than one solution or more than one valid solution.
Analysis	Given a specific design problem, provide a list of possible solutions and map those to a set of design criteria. Example: Given a design problem that requires sorting and searching a list of items, list the appropriate data structures for storing the items and how they relate to criteria, such as time required to search, time required to add new items, etc.	Students in dualism may have difficulty providing more than one solution or more than one valid solution. If multiple solutions are produced, they may have trouble proposing arguments opposing solutions they have already deemed "correct" or identifying arguments supporting solutions other than the "correct" one.
Synthesis	Given a specific design problem, define the criteria that should be used in order to make a decision. Example: Given a design problem that requires sorting and searching a list of items, what criteria should be used to evaluate candidate data structures? Which criteria are more important to the specific problem?	This is not clear. Will students in dualism only come up with criteria that apply to their chosen alternative, discarding any criteria that do not support their beliefs? Will students in the multiplicity stage have issues identifying some criteria as being more important than others or will they consider all criteria equally valid?
Evaluation	Given a specific design problem, define alternatives and the key criteria, and use this information to select a solution. Example: A design problem that requires sorting and searching a list of items, what are the candidate data structures, what criteria apply in evaluating the appropriateness of each data structure to solving the problem, and given those criteria, which solution is the best choice?	Students in dualism are likely to have the same issues listed above, and are likely to have difficulty getting to the evaluation stage. Students in multiplicity may have difficulty in making a selection, even after identifying alternatives and criteria.

the instructor to use to assess both the student's understanding of the problem at hand and where they are likely to be in their development along the Perry (1970) scale. Understanding where the students are developmentally, relative to where we want them to go, is important in deciding how to help them progress. Rationale can be a valuable tool in both aiding and assessing that progression.

Our experiment demonstrated that rationale provides a mechanism for students to express the results of the analysis, synthesis, and evaluation required to design solutions to problems and provides assistance during the process. Explicitly expressing rationale for their work encouraged both reflection on why they made their choices and the active consideration of multiple alternatives. This experiment demonstrated that students using rationale considered all reasonable alternatives and were able to select criteria and evaluate alternatives in a way that indicated they were progressing in their intellectual development.

REFERENCES

Amabile, T. M. (1983). The social psychology of creativity: A componential conceptualization. *Journal of Personality and Social Psychology*, *45,* 357–377.

Atman, C. J., Cardella, M. E., Turns, J., & Adams, R. S. (2005). Comparing freshmen and senior engineering design processes: An in-depth follow-up study. *Design Studies*, *26*, 325-357.

Baxter Magolda, M. B. (1992). *Knowing and reasoning in college: Gender-related patterns in students' intellectual development*. San Francisco: Jossey-Bass Inc.

Bloom, B. S. (Ed.). (1956). *Taxonomy of educational objectives: The classification of educational goals*. New York: David McKay Company, Inc.

Bruegge, B., & Dutoit, A. H. (2004). *Object-oriented software engineering: Using UML, patterns, and Java* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall.

Burge, J. E., & Brown, D. C. (2004). An integrated approach for software design checking using rationale. In J. Gero (Ed.), *Design Computing and Cognition '04* (pp. 557–576). Cambridge, MA, USA: Kluwer Academic Publishers.

Carey, T., McKerlie, D., & Wilson, J. (1996). HCI design rationales as a learning resource. In T. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use* (pp. 373–392). Mahwah, NJ, USA: Lawrence Erlbaum Associates.

Carroll, J. M. (2009, June). *The essential tension of creativity and rationale in software design*. Manifesto from the workshop on Creativity and Rationale in Software Design, State College, PA, USA.

Casaday, G. (1996). Rationale in practice: Templates for capturing and applying design experience. In T. Moran & J. M. Carroll (Eds.), *Design rationale: Concepts, techniques, and use* (pp. 351–372). Mahwah, NJ, USA: Lawrence Erlbaum Associates.

Daughtry, J., Burge, J., Carroll, J. M., & Potts, C. (2009). Creativity and rationale in software design. *SIGSOFT Software Engineering Notes, 34*(1), 27–29.

Dewey, J. (1933). *How we think*. New York: D.C. Heath and Company.

Dym, C. L., Agogino, A. M., Eris, O., Frey, D. D., & Liefer, L. J. (2005). Engineering design thinking, teaching, and learning. *Journal of Engineering Education*, *94*, 103–120.

Fischer, G., Lemke, A. C., McCall, R., & Morch, A. I. (1991). Making argumentation serve design. *Human–Computer Interaction*, *6*, 393–419.

Fox, C. (2006). *Introduction to software engineering design*. Boston, MA, USA: Addison Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, MA, USA: Addison-Wesley.

Glass, R. S. (1995). *Software creativity*. Englewood Cliffs, NJ, USA: Prentice Hall.

Hamilton, H. R. (1974). *Screening business development opportunities*. *Business Horizons*, *7*(4), 13–24.

King, P. M., & Kitchener, K. (1994). *Developing reflective judgment*: *Understanding and promoting growth and critical thinking*. San Francisco: Jossey-Bass Publishers.

King, P. M., & Kitchener, K. (2002). The reflective judgment model: Twenty years of research on epistemic cognition. In B. K. Hofer & P. R. Pintrich (Eds.), *Personal epistemology: The psychology of beliefs about knowledge and knowing* (pp. 37–61). Mahway, NJ, USA: Lawrence Erlbaum.

Kloss, R. J. (1994). A nudge is best: Helping students through the Perry scheme of intellectual development. *College Teaching*, *42*, 151–158.

MacLean, A., Young, R. M., Bellotti, V. M., & Moran, T. P. (1991). Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, *6*, 201–250.

Moran, T. P., & Carroll, J. M. (Eds.). (1996). *Design rationale: Concepts, techniques, and use*. Mahwah, NJ, USA: Lawrence Erlbaum Associates.

Perry, W. G. (1970). *Forms of intellectual and ethical development in the college years: A scheme*. New York: Holt, Rinehart, and Winston.

Pugh, S. (1991). *Total design*. Wokingham, UK: Addison-Wesley Publishing Company.

Rice, J. A. (1995). Mathematical statistics and data analysis (2nd ed.). Belmont, CA, USA: Duxbury Press.

Schön, D. A. (1983). *The reflective practitioner*. New York: Basic Books, Inc.

Shah, J. J., Smith, S. M., & Vargas-Hernandez, N. (2003). Metrics for measuring ideation effectiveness. *Design Studies, 24*, 111–134.

van Aalst, J. W., van der Mast, C. A. P. G., & Carey, T. T. (1995). An interactive multimedia tutorial for user interface design. *Computers and Education*, *25*, 227–233.

VanGundy, A. B. (1981). *Techniques of structured problem solving*. New York: Van Nostrand Reinhold Company.

Whiting, C. S. (1958). *Creative thinking*. New York: Reinhold.

Authors' Note

This work was supported by NSF CAREER Award CCF-0844638 (Burge). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

We would like to thank the students taking the Data Structures course for their willingness to participate in this experiment, and fellow participants in the workshop on Creativity and Rationale in Software Design.

Correspondence should be addressed to:
Janet Burge
Miami University
205V Benton Hall
Oxford, Ohio, USA
burgeje@muohio.edu