The Decision Pattern: Capturing and Communicating Design Intent

David R. Wright
Computer Science Department
North Carolina State University
Raleigh, NC USA
drwright3@ncsu.edu

ABSTRACT
The design of large software systems requires designers to apply critical thinking skills to resolve the complex design problems these systems typically generate. Cultivating these skills in novice designers involves exposing them to the complexities of these problems, allowing them to critically explore the problem and application domains, and providing a means to communicate the rationales for the design decisions they make. In a classroom learning environment, the processes students use to think about and organize what they know and do not know about the design problems is more important to the students’ learning than the artifacts they generate. Effectively communicating to the instructor why they make their decisions facilitates assessment and feedback on the thinking and organizing process, and enhances the learning experience. This paper presents a pattern or template for capturing and communicating this information in a concise yet information-rich format. The author’s experience using this pattern in a teaching and learning context is discussed to validate the effectiveness of this pattern.

Categories and Subject Descriptors
K. COMPUTING MILIEUX [K.3 COMPUTERS AND EDUCATION]: K.3.2 Computer and Information Science Education—Computer science education, Self-assessment

General Terms
Design, Documentation

Keywords
software system design, communicating design decisions, design patterns, critical thinking, design rationalization

1. INTRODUCTION

Standard curricula in computer science and software engineering guide students through software design and implementation problems of progressively greater complexity. These students rely on their past experiences and classroom examples to help them develop solutions to these problems. Often, however, teaching and assessment methods focus only on the results: does the student’s program generate the correct output for a given input. The assumption is that if the program does what it is expected, the student must understand why (and not just how) the program works.

Open-ended problems are more difficult to construct and grade, particularly when the learning objective of the problem is to assess how the student constructed the solution [15]. Students are usually required to write some type of narrative or expository document to explain how the solution was developed and why it is correct. Students will often write these papers after the solution is complete rather than while they are working on the problem. Connections between the documentation and the solution can be hard to identify, complicating specific and effective feedback to support the student’s learning.

This paper presents a template, in pattern form, of a concise means for students to document their design decisions when working on complex software design problems, and discusses the author’s experiences using the pattern in an advanced undergraduate software design course. This experience demonstrated the effectiveness of this template as a compact and information-dense means for students to collect and communicate the processes they use to create solutions to open-ended design problems. Section 2 discusses the pedagogical background of the pattern and of the assignments given to students in these classes. The Decision Pattern itself is presented in Section 3, followed by a discussion of how it was used in Section 4. Reflections on the use of the pattern, from both student and instructor perspectives, are discussed in Section 5. The final section discusses the effectiveness of this tool, and future work to validate the use of the pattern in other contexts.

2. DESIGN DECISIONS AND CRITICAL THINKING

Critical thinking is a fundamental skill set college students are expected to possess and be able to use when they graduate [4, p.328]. Design in general, and software system design specifically, requires practitioners to apply critical thinking and reasoning abilities. To understand the connection between design and critical thinking, it is necessary to understand what design is in a fundamental way. This connection establishes a rationale for understanding the thought
processes novice software designers use when they approach new problems outside their realm of experience, and lays the foundation for the Decision Pattern described later in this paper.

In software engineering terms, software design is the group of activities that transform requirements and specifications for a software system into artifacts that communicate the structure and behavior of the system to the developers and programmers that actually implement the system. Design is "complete" when a complete model of the system (or some part of it) is ready to be implemented [14], but what this means in practice is dependent upon the development methodology used on a particular project.

Kazmierczak shifts the emphasis away from the objects and artifacts of the design process and onto the meaning that is created and communicated by those artifacts [10]. Rust compliments this definition of design by noting that the artifacts of design must embody the tacit knowledge of the designer in a manner that makes that knowledge accessible to the receiver [18]. Jones [9] states that the process of designing is "the designer’s way of discovering what he knows, and what he does not know, about this new thing he has promised to invent, and to integrate it into the world as it is." Knowledge acquisition, sharing, and integration are critical activities in the design of large software systems [25].

However, the process of designing is not only logical and rational, it is also creative and intuitive [7, 9, 19]. It is here that the connection with critical thinking begins to emerge. Quellmalz identified six key strategies of critical thinking, employed in a dynamic and metacognitive process that includes planning, monitoring, reviewing, and revising [1]:

- Identifying and defining issues and problems.
- Determining the kind of information that is relevant to solving the problem or resolving the issue.
- Gathering, judging, and connecting the information.
- Generating hypotheses, constructing arguments, making inferences.
- Testing hypotheses, making counter-arguments.
- Evaluating the results, possibly revisiting earlier stages of the process.

Design problems are usually “ill-structured” [20] and even "wicked" [5, 16] because they rarely have precisely prescribed goals or objectives, and the means of solving the (apparent) problem is not obvious [17, p.40-41]. Compounds this with constantly changing technology and user requirements, and software system design becomes a very difficult problem. Software designers must continually refine and reformulate the problem(s) at hand, searching out new knowledge that may be applicable to the situation at hand, proposing alternative solutions based on the new and existing knowledge and experience, and learning from the evaluation of these proposed solutions [11]. The ability to objectively look at a problem from many different perspectives and to formulate and evaluate potential solutions that integrate those different perspectives is a necessary quality of the best software designers [13].

The complementary relationship between design thinking and critical thinking formed part of the basis for senior-level undergraduate course entitled “Perspectives on Software System Design” that was taught by the author in Spring 2007. A key premise of the course was that large software systems should be viewed from many different perspectives as early as possible if the development of the system is to meet the the diverse needs of all of the system’s stakeholders. The course content and graded exercises were designed to cultivate critical thinking skills in the students in the context of completing software design tasks for systems much larger than normally encountered in an undergraduate Computer Science program. The Decision Pattern was developed as a tool to support teaching and learning in this class, inducing students to discover, use, and develop their own critical thinking skills in the context of software system design.

3. THE DECISION PATTERN

Learning exercises were used in the course to develop design skills, and it was critical for the student to be able to effectively communicate the results and the rationale of their design decisions to the instructor. Students were required, as a graded part of the assignments, to explain why they made the choices they did. For the first assignment, they were allowed to document their decisions using a short narrative discussion. It was expected that these descriptions would be fairly vague and difficult to relate to the actual design artifacts generated in response to the questions.

To facilitate clearer communication between the student and the instructor in the grading process, a compact and structured template was needed to capture the student’s rationale for each design decision. One such structure is the argument map that provides a strict convention for diagramming sequences of logical arguments in support of a chain of reasoning [24]. This sequence of arguments allows the instructor to quickly trace the thought processes that the student followed to arrive at the solution. Herman also notes that requiring students to explain both their solution and how they arrived at it as vital to developing critical thinking skills [8]. However, the argument map format would have been awkward to use in the context of the kinds of problems students would be faced with in this class.

Rather than introduce a new and not directly related structure into the exercises, I decided to build upon the students’ past experience. One very familiar concept (based on how the introductory programming courses are often taught) is the Design-by-Contract structure for method or function comments. This structure, as commonly taught, documenting a method with information shown in Figure 1.

| Summary | a concise and complete summary of what the method does from the method user’s view. It only describes the externally evident functionality of the method. Each parameter to the method must be accounted for in this description. |
| Preconditions | Conditions or properties that must be true before the method is called if the method is to successfully complete its computation. |
| Postconditions | Conditions or properties that are true when the method returns, including return values and side effects. |

Figure 1: The Design-by-Contract source code documentation structure

As simple and straightforward as the design-by-contract structure is, I felt that it was not flexible enough to handle
the kinds of design decisions that students were expected
to make. Also, it does not provide an adequate structure
for describing why a particular decision is made, or the
constraints that make that decision the most appropriate reso-


lution. In a discussion of social influences on software archi-
tecture, Cockburn employs a four-element structure (intent,
context, forces, and resolution) for documenting design de-
cisions in the case study he considers [6]. This structure
provided most of the basic elements needed to capture how
and why students made the decisions that led to their solu-
tions. Because it is a compact template, intended to capture
a single decision, it was hoped that students would be more
receptive to documenting their work towards a solution dur-
ging the process rather than after the fact. The template was
presented as the solution element of the pattern shown in
Figure 2.

<table>
<thead>
<tr>
<th>Pattern Name: Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem:</strong> How can novice software designers capture and communicate their motivation and justification for decisions they make.</td>
</tr>
<tr>
<td><strong>Context:</strong> Open-ended software system design tasks incorporated into homework assignments and in-class exercises.</td>
</tr>
<tr>
<td><strong>Forces:</strong></td>
</tr>
<tr>
<td>- Narrative descriptions can be vague.</td>
</tr>
<tr>
<td>- Templates provide cues for necessary information.</td>
</tr>
<tr>
<td>- Documenting individual decisions supports targeted feedback at problem points.</td>
</tr>
<tr>
<td>- Structured documentation supports self-reflection over multiple decisions.</td>
</tr>
<tr>
<td><strong>Solution:</strong> Use the following template to record design decisions:</td>
</tr>
<tr>
<td>- Decision number - Chronological ordering of decisions.</td>
</tr>
<tr>
<td>- Intent - The reason why this decision must be made.</td>
</tr>
<tr>
<td>- Context - The question or problem this decision answers or resolves, its connection to the world/domain, and why it is important.</td>
</tr>
<tr>
<td>- Forces and Counterforces - Information that defines the problem and limits the solution. Whenever possible, both sides (pro and con) should be given.</td>
</tr>
<tr>
<td>- Resolution - How the problem is resolved, taking into account all of the forces and their relationships to and within the context of the problem.</td>
</tr>
<tr>
<td>- Predecessor(s) - The decision(s) made prior to this one that resulted in the need to make this decision.</td>
</tr>
</tbody>
</table>

**Figure 2: The Decision Pattern**

There are two significant differences between this pattern and Cockburn’s. First, the **Forces** section explicitly requires listing counterforces. This was done to push students to actively consider both sides of each piece of information, e.g., the positive and negative influences of that force on the problem. Second, a **Predecessor** section was added to the pattern to capture the ordering and dependency relationships in a set of design decisions. This information was not necessary for Cockburn’s analysis. However, to assess the processes students use to solve design problems, this information may help teachers (and researchers) understand how earlier decisions influence later ones by studying decisions as parts of a sequence rather than as isolated events.

In my experiences as a teaching assistant and instructor, I have often asked students why they made a certain decision. This question was often raised when I suspected that their choice was more of a guess than a justifiable response, or because the choice did not fit with the other choices that had been made towards completing the task or problem. The usual responses to this questioning were statements like “I saw a similar example in the textbook (web page, handout, etc.)” or “This just seemed like the right thing to do here” or, in the case of bolder students, “I thought we were expected to use this (technique, idiom, etc.) somewhere in the assignment since it was recently covered in class.” Requiring students to document their solutions with instances of the decision pattern would support the critical thinking process: they would become more accustomed to justifying their decisions, and they would also begin to develop the habit of reflecting on past decisions to help keep them focused on the problem at hand.

As van Gelder noted [24], requiring students to use a strict format for documenting their decisions can provide insights into how the student arrived at a particular solution. This is similar to showing each step in a mathematical proof or derivation: a step represents the application of some operation or transformation intended to move the problem closer to a successful resolution. The teacher can pinpoint errors in judgement more easily and precisely, and provide feedback that addresses the specific mistake. Such direct and targeted feedback helps the student learn from their mistakes more effectively than general comments (e.g., often the teacher’s or grader’s best guess as to where the student’s reasoning toward the solution went off track) about the incorrectness of a response or problem solution.

The **Decision Pattern** abstracts and structures the key elements necessary to make a critically-informed decision. This structure supports student learning by providing a template for documenting their progress towards completing difficult problem-solving and design tasks. By effectively communicating these thought processes to the instructor, feedback can address specific errors in judgement or logic that resulted in incorrect or inappropriate design artifacts, enhancing the instructor’s ability to use written assignments as learning tools as well as assessment instruments.

4. APPLICATION AND EXPERIENCE

The importance of understanding how to design software systems is reflected in the National Science Foundation’s (NSF) creation of a Science of Design Program [22] directed at funding research into the design of software-intensive systems, drawing a distinction between design and implementation (the realm of software engineering and other related research areas) [21]. Brooks states that while not a “silver bullet” by itself, software designers need to be cultivated and nurtured to build the skills needed to successfully design high-quality software systems [2]. The design of large systems involves more than programming: the system designer must take into account human and social factors, environmental issues, rapid technological advancement, and other concerns outside the traditional topics of computer science and software engineering [12].

In response to this need at the undergraduate level, the “Perspectives on Software System Design” course was de-
developed to expose advanced undergraduate students to the complexities of large software systems. The course took an architectural approach to software system design, in the sense that architecture is concerned with building structures within an existing environment, generating a new environment for social and cognitive experience within its boundary.

While software systems do not have the physical impact that buildings do, they do create very similar conditions for human users [3]. In addition to the complexities of the software itself, the cognitive and social influences on the design and development processes, as well as those generated when the system is installed, induce complexities that are difficult (if not impossible) to resolve unless the designer is cognizant of these perspectives and influences. This course offered students insight into these complexities and challenged them to build their own awareness of the many different perspectives that affect the development of large software systems.

The design tasks used in this course were taken from both textbook examples and problems as well as actual production systems. The tasks were intentionally open-ended, mimicking real-world design problems where decisions made early in the design process constrained (sometimes severely) what could be done later. Homework assignments contained two to four tasks, with the later tasks depending partially upon the results generated for the earlier tasks. In-class exercises focused on a single task or activity. The Decision Pattern was introduced in conjunction with the second homework assignment, and its use was required in that and all subsequent homeworks.

The first and second assignments presented students with similar problem domains and required them to respond to the same set of tasks. One-page documents described the proposed systems in non-technical language that omitted key pieces of information and while including extraneous details. Both assignments required documentation and justification of the responses: the first assignment using any form the student desired, as long as the rationale was tied to specific elements of the response; the second assignment required students to use the template given in the Decision Pattern and directly associate a particular decision with response elements. The objective of these two assignments, as a unit, was to introduce the Decision Pattern format for documenting design decisions in conjunction with a set of design principles [27] to be used as triggers and guides for their decisions.

Student responses for the first assignment were vague, shallow, and full of unjustified assumptions, although they were not stated as such. While most of the students had been exposed to interactive web-based systems (the focus of the assignment), they had never experienced the responsibility for translating a customer's description of such a system into a high-level architectural design. Justifications for their decisions were equally vague in most cases, including statements such as:

- “These elements are enough to meet the requirements and they have the potential to provide a complex and efficient solution when implemented well.”
- “I feel with these 4 user types would cover any circumstance and would not leave any part of the system to hang in a certain state.”
- “I only have the states above and no more because I feel as though a system state should represent a single small set of related actions and all such actions desired by the system are covered in the seven states above.”

As noted above, the second assignment required the use of the Decision Pattern to document the decisions students made while completing the assignment. The graded submissions for the first assignment were not returned to the students until after the second assignment. This was done to prevent students from adapting their responses for the first assignment to the second assignment. As expected, some (approximately 4 - 6 of 15) of the students did “reuse” their earlier work, based on finding the same kinds of errors and assumptions in their responses. The structured documentation for these decisions was also shallow and probably done after the solution was written up. About one-half of the students' submissions had a mixture of concurrent and fill-in documentation, while 3 students made a definite effort to use the Decision Pattern while they were working out their solutions. In these 3 cases, the documentation clearly described and justified their decisions, even when those decisions did not directly result in an element of the solution (i.e., a decision was made to partition the problem domain in a certain way). Furthermore, in several instances, an early decision clearly foreshadowed later decisions that would be made, although the students did not make the connection in their submissions.

By the middle of the semester, all of the students were correctly using the pattern in their written work, clearly documenting their decisions as they worked rather than after the fact. Their responses to assigned tasks became more complete and well thought-out. They also began asking more penetrating questions to clarify their understanding of the given problems, and often justifying their questions based on the problem write-ups and external sources.

This change in attitude and thinking also spilled over into the classroom. I had worked hard from the start of the semester to cultivate an atmosphere that welcomed questions from students and encouraged them to critique each other’s questions and answers. The Decision Pattern gave them a framework to structure these critiques, and by the middle of the semester, almost the entire class had internalized that structure and used it naturally and effectively during class discussions. Questions were framed using the intent, context, and forces elements of the template in expectation of an appropriate resolution. If the response (either from me or another student) did not adequately address and resolve the forces, students (not always the one who originally asked the question) would immediately recognize and point out the imbalance. While the environment supporting student questions was actively nurtured in the classroom, the students’ adoption of the Decision Pattern as a cognitive structure for framing their questions and responses was an unexpected surprise.

5. REFLECTIONS

The Decision Pattern was developed as a teaching and learning tool rather than as a research instrument. The intent was to provide a means for students to record their thinking processes in a form that would easily communicate those processes to the instructor, and allow them to reflect back over their own decision sequences. The information structure of the pattern was designed to promote critical thinking based on accepted principles [8,23,24] and extends
the contract programming paradigm students were already familiar with. Based on students’ classroom participation and dialogue, performance on written assignments, and private comments and questions outside of the classroom environment, the Decision Pattern achieved this intent.

In my teaching experiences I have recognized the desire many students, particularly undergraduates, have for structured presentation of information, both from the instructor as well as in the work they turn in for grading. The open-ended nature of this course was uncomfortable at first for nearly every member of the class. This was reflected in one student’s comment in the course evaluation:

I remember thinking while working on the first design exercise that it would have been helpful to have some kind of basic template or example to follow. I had become accustomed to classes providing strict guidelines for completing assignments. This [open-ended assignments] however required the student to think about problems instead of just following instructions.

This comment mirrors Tsui’s findings that asking learners to construct responses or answers to questions, problems, etc. is more effective at promoting critical thinking than just memorizing and regurgitating correct answers [23]. The Decision Pattern provided students with a way to bring some structure to the course content while allowing themselves to think more openly than most of their other courses would allow. With familiarity and practice, the critical thinking process embodied in the pattern became integrated into their natural thought processes. Another student wrote:

The decision pattern is just a tool of the critical thinking process, but this pattern and this course in general has helped me to enhance my critical thinking by analyzing the reasons why I do or do not do things — be that software design or just things in general.

A common theme that ran through the students’ reflections on their own learning was the need to effectively communicate with others. Many noted that standard design notations (such as UML) communicate the design itself, but do not easily incorporate the assumptions, forces, and sequences of decisions that result in those notational artifacts. They also noted that this tacit information is necessary to construct a solid understanding of the artifacts and their relationships, which in turn leads to an understanding of the system as a whole. Several students noted the value of the Decision Pattern for communicating the intent and rationale supporting design decisions.

From a teaching perspective, the Decision Pattern was valuable in several ways, all related to communicating with students in different ways. I first began using the pattern as an aid for constructing lesson plans. As a relatively inexperienced teacher, two of the most difficult aspects of preparing a lecture are succinctly stating the objective for the class session and connecting the new material with the students’ prior learning experiences. The intent section of the pattern forced me to capture the lesson objective in a single sentence, concise, but rich with meaning. The context was defined by my expectations of the students’ previous coursework (based on prerequisites and knowledge of the curriculum) and modified by my own impressions of the students’ current understanding of topics related to the objective. The forces section allowed me to characterize the relationship between the new material and their existing understanding. This part of the pattern also pushed me to think about questions and objections students might have to the new material. The resolution then became the actual lesson plan, constructed with all of the other information clearly documented with it. I did not explicitly expose the use of the Decision Pattern in the lectures or materials, although in retrospect, it probably would have been valuable for the students to see the pattern used in another context.

A second way that the Decision Pattern was valuable in the classroom was as a common structure for critical dialogue. I encouraged students to ask questions during class, as well as to answer each other’s questions. I took this a step further by challenging them to critically evaluate any answer, whether it was from another student or from me. The Decision Pattern provided a basis for that evaluation. To stimulate discussion, I would often ask the questioner to state the context of their question clearly, and to list the forces constraining possible answers as they understood the question. I would then pose the question to the class, first asking them to consider whether or not the set of forces was complete enough to resolve the question. Very often, the process of analyzing the question would lead the student who asked the question to answer it themselves.

As the semester progressed, the tenor of the questions moved away from simple clarification to deeper and more insightful questions about how to apply the lecture and reading material in new ways and domains. This occurred in parallel with the growth in the students’ ability to use the Decision Pattern in the homework exercises. Part of this may have been a result of my own development and use of the Decision Pattern in my lecture preparation, resulting in clearer presentations that preemptively answered the simpler questions. However, the insights many students exhibited were clearly a result of their own critical thought about the material and how it could be used.

The third way the Decision Pattern was valuable to me as a teacher was in the construction of exercises, problems, and test questions. As I continued to use the pattern as a guide for my lessons, I found that I was using it in the background as I developed various problems for students to work out. Once I realized this, I started explicitly using the pattern in much the same way as I did to construct lesson plans. The result was that the problems I created were much more on track with what I wanted students to learn from them. They were also crafted in ways that made it easier for me (as the grader) to assess what they had learned and where their weaknesses were.

Reflecting back on the initial presentation of the pattern, it would have been better to have introduced it as part of an in-class exercise rather than as part of a homework assignment. This would have allowed their initial use of the structure to occur in a supervised environment, and probably would have sped up the acceptance and correct usage of the pattern. Ideally, this initial presentation would have given students the opportunity to identify and document the pattern on their own. This would have given the students a sense of personal ownership in the structure, and its impact would have been more immediate and pervasive.
6. CONCLUSIONS AND FUTURE WORK

This paper has reported on the development and use of a template structure for capturing and communicating decision students make while working on software system design problems, and on the author’s experiences using this template in a classroom environment. The Decision Pattern presented in this paper was derived from accepted principles for teaching and assessing critical thinking and leverages Computer Science students’ prior experience with the contract programming paradigm and design patterns. It was designed to enable students to record how and why they made their design decisions as they worked on homework and in-class exercises, and to enable a teacher to easily trace and evaluate a student’s reasoning processes.

The Decision Pattern was a valuable tool to support teaching critical thinking in the context of software system design at the advanced undergraduate level. It provided students with a touchstone of structure in an otherwise open-ended and abstract course, allowing them to communicate their design decisions and corresponding justifications. As they became comfortable with the construct, it was used to support their verbal communication as well as their written work. As a tool for the teacher, the Decision Pattern was also very useful as a framework for designing and constructing lessons and problems. The structure of the pattern can serve as a template for creating comprehensive teaching and learning entities intended to communicate specific objectives to students.

As a research tool, the Decision Pattern can serve as an alternative or supplemental data collection method in studies of subjects’ decision-making processes. Self-reporting does have limitations, particularly the problem of common method variance, but there are techniques such as using multiple subject types or longitudinal studies that can help minimize these effects [26]. A grounded theory analysis is currently under way using the written homework submissions from two groups of students: those in the class described in this paper, and another group in a graduate-level software engineering course. The graduate class completed the same two assignments as the undergraduate class did, including the use of the Decision Pattern as a documentation tool for their design decisions. From this analysis we hope to identify common types and sequences of decisions that will lead to a deeper understanding of how novice designers reason about design. Additional studies are planned with lower-level undergraduate students as well as with experienced software professionals, with the goal of identifying different design reasoning strategies. These results could be used to guide curricular modifications to better prepare students for real-world software system design activities.

This pattern could be easily adapted and used in many other disciplines where the interest is in how and why subjects construct responses. While most studies of critical thinking involve observation of subjects involved in critical thinking activities or interviews before and after such activities, the Decision Pattern provides a structure for subjects to report their own reasoning processes. As noted earlier, undergraduate students expect a high degree of structure in their work, and the Decision Pattern provides a structured vehicle for communicating their thinking and problem-solving processes that is easier to assess than traditional narratives.

7. REFERENCES