Patterns as Topics in Computer Science Courses

Dwight Deugo
deugo@scs.carleton.ca, 103163.2657@compuserve.com
http://www.scs.carleon.ca/~deugo
School of Computer Science,
Carleton University
1125 Colonel By Drive,
Ottawa, Ontario, Canada, K1S 5B6

Introduction

I strongly believe that writing patterns is an excellent way to describe well-known, proven software engineering principles and programming concepts. I also believe that reading, discussing and applying patterns are great ways to learn about these elements, and that these efforts should begin in a student's very first undergraduate computer science course – SC1. However, I do not believe that instructors need to discuss patterns at the same level of detail in a first-year course as they would in a graduate course. If you consider patterns as trees in a forest of principles and concepts, there is no need to make the forest so dense that students are afraid to enter and make discoveries. Some controlled cultivation and growth is required to make it easy for students to navigate.

I have two goals for this position paper. First, I want to present my views on how to introduce patterns into different levels of computer science courses. I do this in order to initiate discussions with others on how they have done the same and then come to some agreement on the best approach. Second, I want to solidify an outline for a complementary book to CS1 based on patterns.

To meet these goals, I begin by describing my approach to introducing patterns into CS1 (Introduction to Object-Oriented Programming), its continuation course (Design and Implementation of Computer Applications) and a graduate course called Software Patterns. For each course, I describe why patterns are important for it and the approach I took to introduce them. I also include an initial draft of a complementary book to CS1 based on patterns, identifying existing patterns that fit the subject matter and new ones needing development. I conclude with some thoughts on using patterns in computer science courses.

Introducing Patterns in CS1

Over the past few years, those teaching introductory courses on object-oriented programming (specifically Java) have had a difficult time finding books to use for their courses. The problem was that, although there were numerous books to choose from, most were syntactic descriptions of Java with little emphasis on general object-oriented concepts and programming techniques. When a book did focus on techniques, it often omitted the reasons as to why the techniques were important and when it was appropriate to use them. One example of this situation is in Deitel & Deitel's book [Deitel & Deitel 98]. The book's syntactic discussion on for loops continues for seven and one half pages. The first page's first line states, 'the For repetition structure handles all the details of counter-controlled repetition.' The remaining pages describe numerous examples and syntactic rules to remember when using for loops. The learning paradigm used here, one popular with artificial intelligence research, is learning by example. However, I believe we can do better than forcing our students to learn by example all of the time. In this case, I believe we should start with a discussion of the concepts of looping and then the problems the control structure can solve for us before we ever describe the syntax and implementations of the solutions. My comment about these types of books is that they are missing the foundations for real-world programming – the software engineering approach.

Fortunately, there is another bread of book making its way into the market for CS1 courses on object-oriented programming. Take for example [Holmes 98; Savitch 99; and Wu 99]. Written with the ACM-recommended curriculum for CS1 in mind, these books emphasize a gradual introduction to the fundamentals of programming along with becoming familiar with and using the pre-written Java classes

and methods. These are not just books on syntax! In Holmes' book, for example, different contexts dealing with looping: loops controlled by a counter and ones controlled by data, are made explicit. No longer does the reader have to infer the context for himself from the examples, as with the previous types of books, although the examples are still there to help with his understanding. However, all is still not well. Although the contexts, solutions and implementations are present, little discussion is devoted as to why one solution, approach, technique or class is preferred over another. Again, this information is left to the reader as an exercise.

The point I am trying to make here is that with every new book, we get closer and closer to having resources written in simple pattern form: name, problem, context, forces, solution and implementation. It is not surprising then to me that many of my CS1 lectures turn out to pattern lectures. I am forced to fill in the information the books leave out.

Why are Patterns important for CS1?

One question I ask myself is the following: am I constantly pushing patterns because I am a 'Pattern' person, or is there a greater benefit other than my own personal satisfaction in getting someone else hooked on them? In other words, would I be discussing patterns in CS1, even if I was not someone who enjoyed writing and reading them? I believe it is a good question to ask, since in order to introduce patterns into CS1 it requires the support of non-pattern people. Many instructors will never write a single pattern or attend one of the many PLoP conferences. These people will also need to be convinced of the merits of patterns before they ever introduce them into CS1.

My argument to these people is based on features I find missing from the CS1 books they use for object-oriented programming and in many of their lectures. First, neither describes the forces of a given solution to a problem in a context. Second, the relationship between the tuples is for the most part missing, e.g. which loop construct to use in a situation or which collection class is best. Pattern and pattern languages are a good way to ensure that all programming elements are in place, discussed and related to one another. What not add them to the course?

It is not hard to argue that there is growing acceptance of patterns as being a valuable resource for documenting the time-honored practices, designs, and software elements of the past generations of software architects, designers, developers and language specialists. This statement forces one to the realization that the new bread of software engineers at some time in their careers will be either forced to learn specific patterns or wind up discovering ones for themselves. Since software development is expensive and there is no benefit to having future developers learn everything from scratch all the time, there is no better time than in their first computer science course to introduce them to patterns.

How were Patterns Introduced?

In CS1, I wanted to lecture about the details of specific patterns, rather than on what patterns were. This meant that many of my lectures tended to focus on problems, contexts, forces, solutions and then the associated implementation details with Java, without ever mentioning the word pattern. The reason for this approach was that the moment I mentioned the word pattern, students began thinking of patterns as another sub-topic of the course. They began asking questions about pattern formats, history, books, and templates, and this was not what I was trying to emphasize in the course. Rather, I wanted to take a pattern approach to imparting the material because I believe it to be the best way, both structurally and pedagogically, to get the material across. I never discussed the background of the pattern movement. This level of detail I left for the continuation course and for my graduate software patterns course.

My approach to any topic, which generally coincided with a chapter in book, was the following:

1) Review the chapter for the high-level concepts I wished to discuss. For example, a chapter about the class hierarchy would break down into the concepts of Poloymorphism, Delegation, Substitution, and Inheritance.
2) Develop an overall context where all concepts would apply.
3) Develop problems that gave rise to solutions involving the concepts.

4) Refine the global context for each problem and add a discussion of the corresponding forces.
5) Describe the solution to each problem in terms of the appropriate concept and the specific language constructs or classes. At this point, smaller patterns dealing with language trade-off would often be needed. For example, in Java you can use interfaces or abstract classes and subclasses to implement the substitution of one object for another. Which is better to use? When should one approach be used over another? These were easily explained in pattern terms.
6) Show the relationships between the various patterns. In effect, every topic would give rise to a pattern language. Some patterns naturally followed others, while some were in tension with one other. It was important for students to see and understand these relationships.
7) As the term went on, I would start to give each pattern a specific name, again without every saying that I had described a pattern. This helped both in class discussions when referring back to material and discussion with students during office hours.

A typical CS1 textbook has the following sections: Introduction, Data Types, Program Design, Selection, Repetition, Arrays, Classes and Methods, Encapsulation, Class Hierarchy, Exception Handling, Graphical User Interfaces, Applets, and Algorithm and Data Structures. One section I would also add to this was on Collections. It was not useful to take a pattern approach for every topic. For example, I presented the early course material: Introduction, Data Types and Program Design, as is. However, once I was in a section where different choices could be made, I found the pattern approach worked well. Even in the sections on Classes and Methods, I could always present discussions on public, private and protected access in terms of what problems were solved by using these features of the language. Once students could understand the problem, I felt they could also better understand the solutions. The reverse was not the case. For example, if I presented how to use these modifiers first and then presented a problem, many could not answer which modifier was best to use. The path towards their understanding seemed to be blocked.

It has been my experience that students can understand material better when you go beyond what most textbooks provide (problem and solutions pairs), and answer the simple question of why a solution is appropriate for the problem? Since patterns answer this question explicitly with their contexts and forces, it makes logical sense to include them in any CS1 course.

Outline for First Year Pattern book

Not meant to be a replacement to any existing CS1 book, I believe we can develop a book that complements the existing material with one solely based on patterns. The good news is that some of the chapters have already been written. The idea is to take the same sections as in the CS1 book and develop chapters that provide pattern languages for each. The following is a proposed outline for such a book including existing patterns suitable for its contents.

Title: A Pattern Approach to The Introduction of Computer Science & Object-Oriented Programming

Chapters:

1. Introduction
2. Why the need for the book?
3. Background on Patterns
4. Pattern Format
5. Classes, Objects and Methods
   - Reusability Through Self-Encapsulation, Ken Auer, [Coplien and Schmidt, 95]
   - Strategy Pattern, [Gamma et al., 95]
   - Singleton Pattern, [Gamma et al., 95]
6. Programming Concepts
   - Foundations Patterns: Delegation and Substitution, Dwight Deugo, PLoP ‘98
7. Flow of Control
   - Loop Patterns, Owen Astrachan and Eugene Wallingford, PLoP ’98.
8. Collections

9. Inheritance
10. Event-Driven Programming
    - State Patterns, Paul Dyson and Bruce Anderson, [Martin et al., 98]
    - A Pattern Language For Developing Form Style Windows, Marc Bradac and Becky Fletcher, [Martin et al., 98]
    - React-Update Pattern Language, Dwight Deugo and Dorin Sandu, EuroPLoP '99
    - Observer Pattern, [Gamma et al., 95].
    - Model-View-Controller Pattern, [Buschmann et al., 96]
11. Exception Handling
12. Recursion
    - The Object Recursion Pattern, Bobby Woolf, PLoP '98.
13. Stream Processing
14. Streams: A Pattern Language for 'Pull-Driven" Processing, Stephen H. Edwards, [Coplien and Schmidt, 95]
15. Techniques
    - Lazy Optimization: Patterns for Efficient Smalltalk Programming, Ken Auer and Kent Beck, [Vlissides et al., 96]
    - Null Object, Bobby Woolf, [Martin et al., 98]
    - Type Object, Ralph Johnson and Bobby Woolf, [Martin et al., 98]

The outline is a first draft at indicating what has and what needs to be done. Now all we have to do is agree on the contents and fill in the pieces.

Reintroducing Patterns in the Continuation of CS1

The second core course for students in computer science at Carleton is called Design and Implementation of Computer Applications. This is a continuation of CS1 focusing on the design and implementation of complete applications including the user interface, the software structure, and the interacting domain objects. Applications make use of the techniques such as text processing, symbolic and graphical manipulation and file processing. The assumptions made about students taking this course are that they are familiar with its programming language – Java – and that they understand the basic principles of object-oriented programming. This course is not promoted as a pattern course, but pattern discussions occur often in its lectures.

Why are Patterns Important for the Continuation of CS1?

Patterns are important to this course for two reasons. First, since the course has a design focus, design patterns fit the topic well, and do not require a great deal of motivation, since the students can already see the problems in their applications. I tend to focus on the Gang-of-Four patterns [Gamma et al., 95] because they are applicable to most applications. In general, I like to provide students with information that I know is useful to them not only for the class, but also for their future work.

Second, I feel it is very important to stress to students that many of the design problems they will face, both in the course and later in industry, have been solved already. Too often, I see higher-level students and graduate students reinventing software solutions. It not only costs them time to do this; it also costs the companies they work for in terms of money and time spend reconstructing the solutions. All they need to do is pick up a pattern book once and a while and read a new pattern. To get them to do this, I feel it is important for them to make the connection with patterns early. Therefore, in addition to teaching students about Java programming techniques and Java classes they will use often to develop their applications, every opportunity I can find to apply a know pattern, I incorporate a discussion on it into a lecture.

How were Patterns Reintroduced?

I always introduce patterns in the same way into this second course. Without naming the pattern, I set up a design problem, describing the context and forces. Given time, I will go through one or two different solutions to the problem and discuss their advantages and disadvantages. The point that I try to make to

them is that the proposed solutions have not balanced all of the forces. I try to give the students a feeling that there must be a better way.

When I reach this point, I identify a pattern that has the same problem, context and set of forces to the ones I presented, and discuss its solution. As I heard Jim Coplien and Ken Auer mention once, this is the part where I get that sense of release, or relief as the case may be, from the students. I find that they are able to grasp what is significant about the pattern and then patterns in general.

The following example is one that I find exemplifies this approach. Recently I built a tree object in order to discuss recursion. The first implementation looked as follows:

```
public class Tree extends Object {

    int value;
    Tree rightTree, leftTree;

    public Tree(int nodeValue, Tree leftChild, Tree rightChild) {
            value = nodeValue;
            rightTree = rightChild;
            leftTree = leftChild;}

    public Tree() {
            value = 0;
            rightTree = null;
            leftTree = null;}

    public int height(){

            if ((rightTree == null) && (leftTree == null))
                    return 0;
            if (rightTree == null)
                    return 1 + leftTree.height();
            if (leftTree == null)
                    return 1 + rightTree.height();
            return 1 + Math.max(leftTree.height(),rightTree.height());}


    public static void main (String args[]){
            Tree root;

            root = new Tree(5,
                            new Tree(4,
                                    new Tree(3, null, new Tree(2, null, null)),
                                    new Tree(1, null, null)),
                            new Tree(8,
                                    new Tree(5, null, null),
                                    new Tree(7, null, new Tree(6, null, null))));

            System.out.println(root.height());}
}
```

The problem I want to point out was that the code was checking for null everywhere in the height method - recursion is another good scenario for introducing patterns but I will not discuss it here. I asked the students for different variations and we explored what was good and bad about each. One of the best answers was not to use recursion for height. When I ask the student to give me an alternative solution, the only one he could come up with was recursive. In the end, it appeared to the students that there was no hope for a good solution, and then I mentioned the Null Object pattern.

I discussed the pattern with them and then reimplemented the code using the suggested approach. All agreed that this was a better solution, even at the overhead of creating an interface and a NullTree class. The solution’s simplicity and elegance was visible to the students. They could also see where the pattern could be applied to other problems. This was the point that I wanted to make with them. Good patterns exist for their design problems. In addition, a good pattern is applicable to many problems.

```
public interface TreeInterface {
```

```
    public int height();
}

public class NullTree implements TreeInterface{
    public int height(){return -1;}
}

public class Tree extends Object implements TreeInterface  {

    int value;
    TreeInterface rightTree, leftTree;

    public Tree(int nodeValue, TreeInterface leftChild, TreeInterface rightChild) {
           value = nodeValue;
           rightTree = rightChild;
           leftTree = leftChild;}

    public Tree() {
           value = 0;
           rightTree = new NullTree();
           leftTree = new NullTree();}

    public int height(){
           return 1 + (Math.max(leftTree.height(),rightTree.height()));}


    public static void main (String args[]){
           Tree root;
           NullTree nullTree = new NullTree();

           root =
              new Tree(5,
                   new Tree(4,
                          new Tree(3, nullTree, new Tree(2, nullTree , nullTree )),
                          new Tree(1, new NullTree(), new NullTree())),
                   new Tree(8,
                          new Tree(5, nullTree , nullTree ),
                          new Tree(7, nullTree , new Tree(6, nullTree , nullTree))));

           System.out.println(root.height());}

}
```

One student noticed from my code that it was a good idea to use the same instance for the null tree and wondered how that could be handled globally. This lead nicely into a discussion on the Singleton pattern.

In the past, I have tried two different ways to introduce patterns into this continuation course. The first approach described a design pattern to the students and then showed them how to apply it. The second approach discussed a real design problem or issue, as I did above, and then worked towards the appropriate design pattern(s). I found that students appreciated the latter. The tension and release was there for them to see and feel. The first approach just felt like yet another lecture.

A Graduate Course on Software Patterns

The number of books on patterns and the breadth of the topic areas discussed by them have increased significantly over the last few years. On my bookshelf, I count twenty different books ranging in focus from language specific patterns to those on high-level design, and everything in between. Initially developers could not find any patterns to help them with their design problems, now there are so many of them, developers still can't find the ones they need. The current situation is that the people who know about patterns are the ones that do not need them: the writers and the experts. To bring patterns to main stream developers, we need to educate them about the philosophy of patterns, the pattern community, and the patterns themselves.

My graduate course surveys current developments in software patterns (three-part rules, expressing relations between software contexts, problems and solutions). Pattern categories discussed include

architectural, design, distribution, process, general-purpose, analysis, anti-patterns, and idioms. Students are required to apply existing patterns and to develop and defend new ones.

Why are Patterns Important for a Graduate Course?

Graduates students at Carleton University usually have some object-oriented background but most often have never heard about patterns. The students that have treat them as only problem solution pairs, and none has every written one.

I feel it is important for graduate students to consider themselves as leaders in bringing sound software engineering principles to the software development projects they become involved with. In my mind, patterns are an excellent means to this end. Therefore, a base level knowledge of existing patterns and where patterns can be found is important for them to acquire. However, knowing about existing patterns is not enough. They must develop the ability to recognize their own patterns and be able to write them. This ability not only benefits the companies they work for, but will also add to the growing body of work in the pattern community. In order to do this, student must develop an appreciation for what a pattern is, what it is for, and how it is structured. This level of detail, combined with the corresponding assignments, is impossible to provide in an undergraduate course. Therefore, a graduate course on patterns is, in one sense, the only place for such a course. Moreover, student much reach a level of maturity combined with some real-world experience to both appreciate and write there own patterns, so a graduate course is the suitable place to learn about patterns in detail.

How were Patterns Introduced?

The course has three sections: the background and structure of patterns, existing patterns, and the construction of new patterns. In the first section, I reviewed the work of Alexander, the philosophy behind patterns, and the different formats, making sure students understood the purpose of each heading in the various formats and the relationship between the different headings. Since patterns come in many formats, I felt it was important for students to grasp the core concept of a pattern. As an assignment, I selected two small patterns (one in GOF format and another in the basic Coplien/Alexanderian form [name, problem, context, forces, solution, rationale, and resulting context]) and asked students how they would translate each to the other format. The students all agreed that this exercise forced them to get into the structure of patterns and understand what is required, what is optional, and what is missing from the different formats.

In the second part of the course, we focused on selected elementary, design, architecture, idioms and anti-patterns. Since some students had already reviewed patterns from the Gang-of-Four book, I deliberately chose not to focus on material in this book. Rather, I selected patterns out of the PLoPD books (1-3), the PLoP ’97, ’98 proceedings, the gang-of-five book [Buschmann 96] and the anit-pattern book. Occasionally, I needed to review one of the GOF patterns in order to discuss one of the ‘new’ patterns, but the GOF patterns were never the focus.

As one of the assignments for the second part, I gave the students a small project and asked them to apply as many of the patterns we had discuss in class that were **appropriate** for their design. I also asked them to justify the application of the pattern. This was a good exercise for the students. It not only forced them to go through a design exercise, they also had to think about what patterns they were using and why.

The course’s project assignment had students write a new pattern, have it reviewed, make final modifications to it, and then submit it to me for grading. To keep them on track, in the middle of the course I asked students to provide me with their ‘new’ pattern proposals. After reviewing them, I gave students the ok to write their proposed patterns. The course ended with a student workshop. Similar to PLoP, students broke into groups of eight and reviewed each another’s patterns. Each student had the opportunity to be the lead moderator, session moderator, author and workshop participant, and comments were made and recorded.

The course ended with a one-day take-home exam. This exam focused on concepts rather than on particular patterns. For example, I asked them to discuss the advantages and disadvantages of the different pattern formats. I also gave them a maximum word count so I could read their answers in my lifetime.

Based on the reviews, student found this course both the most interesting and most difficult course they had taken. On one side they complained about the amount of work – the development and writing of their patterns took a lot of time. However, they also said that the workshop was the best part of the course. They felt that the course gave them the graduate experience: learn, write, and publish. There final comment was don't change a thing.

Conclusion

It has been my experience that we can not provide the same pattern treatment in all levels of computer science courses. Although it is important for students to understand the contexts, forces and solutions to software engineering problems and programming concepts that are described in CS1, they do not require a formal introduction to patterns. It is too soon. This is the stage where students are acquiring their own patterns. However, it is worth noting that most CS1 textbooks fall short at providing good discussions on the context and forces of the problems and concepts. Therefore, a complementary CS1 book with patterns would be a welcomed addition.

Once students take a course on building applications, applying what they learned in CS1, this is the time to begin introducing named patterns and having students apply them to their design and programming problems. This is the stage where students become patterns users.

As a forth-year or graduate student, each has reached a stage where they can be pattern writers. They have had some industrial experience and can now write about these experiences. Therefore, it is time for them to be introduced formally to patterns, including their background, format, philosophy and types. To complete the stage, students must suffer and write their first pattern. At first glance, this task seems simple, but as many will agree it is one of the most difficult pieces of writing to do.

What is the next stage? It happens to be recursive, repeat the stages of eval, apply and write. For those of you that are familiar with Scheme you will know about eval and apply. The only difference here is that once you understand how to eval and apply, it is now time to record!

References

F. Buschmann, Meunier, R., Rohnert, H., Sommerlad P., and Stal M., "Pattern-Oriented Software Architecture", John Wiley and Sons, New York, NY, 1996,

J. Coplien, and Schmidt, D. C., eds., "Pattern Languages of Program Design", Addison-Wesley, Reading, MA, 1995.

H.M. Deitel and P. J. Deitel, "Java: How To Program", Prentice-Hall, 1998.

E. Gamma, Helm, R., Johnson, R., and Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, 1995.

B. Holmes, "Programming with Java", Jones and Bartlett, 1998.

R. Martin, Riehle D., and Buschmann F., "Pattern Languages of Program Design", Vol. III, Addison-Wesley, Reading, MA, 1998.

W. Savitch, "Java: An Introduction to Computer Science and Programming, Prentice-Hall, 1999.

J. Vlissides, Coplien J., and Keith, N., eds., Pattern Languages of Program Design, Vol. II, Addison-Wesley, Reading, MA., 1996.

T. Wu, "An Introduction to Object-Oriented Programming with Java", McGraw-Hill, 1999.