

DESIGN PATTERNS GO TO HOLLYWOOD: TEACHING PATTERNS
WITH MULTIMEDIA

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Adam Dukovich

June 2008

AUTHORIZATION FOR REPRODUCTION OF MASTER'S THESIS

I reserve the reproduction rights of this thesis for a period of seven years from the date of submission. I waive reproduction rights after the time span has expired.

Adam Dukovich

Signature

6/9/2008

Date

APPROVAL PAGE

TITLE: Design Patterns Go To Hollywood: Teaching Patterns With Multimedia

AUTHOR: Adam Dukovich

DATE SUBMITTED: June 2008

Dr. David Janzen
Advisor or Committee Chair




Signature

Dr. Gene Fisher
Committee Member



Signature

Dr. Clark Turner
Committee Member



Signature

Abstract

Design Patterns Go To Hollywood: Teaching Patterns With Multimedia

by

Adam Dukovich

Design Patterns have insinuated themselves into the forefront of computer science and software engineering practice. To this end, there has been much scholarship about the proper way to introduce them into the classroom. Studies indicate that understanding the contexts in which design patterns are to be used is one of the most (if not the most) difficult challenge in applying design patterns. However, little research on the topic attempts to solve the problem of better illuminating this context problem, preferring instead to focus on simplification of the patterns and better examples to explain them. This paper discusses a new paradigm through which the teaching of design patterns can be viewed, one which focuses on conceptual examples and contexts as the key elements in teaching design patterns. To better illustrate this new ideology, several short instructional videos, each employing this approach with a different design pattern were created. Their effectiveness was subsequently assessed, relative to traditional lecture that focused more on teaching the structure of the patterns.

Acknowledgements

Thanks to the Cal Poly Computer Science Department for providing funding to create the videos, to Jimmy Hua, Michele Mayorga, and Bobby Kritzer for acting in them, and to Gene Fisher for his cooperation with the experiment. Thanks also to John Dalbey for his cooperation.

Thanks also to Clark Turner, in whose graduate software engineering class this whole idea started to germinate.

This thesis would not have been possible without the assistance of David Janzen, who was an invaluable collaborator in making my thesis as good as possible.

I have also benefited much from the assistance of Heather Smith from the Statistics Department, who was indispensable when it came to breaking down the numbers.

I doubt that I could have completed all this were it not for the support of my family: my parents, Sherell and David; my brother Aaron and my sister Laura.

Contents

List of Tables	viii
List of Figures	ix
1 Introduction	1
2 Related Work	5
2.1 Placement In The Curriculum	5
2.2 Pitfalls to Teaching Design Patterns	6
2.3 Structuralist Notions of Design Patterns Pedagogy	7
2.4 Contextual Pedagogies	8
2.5 Miscellaneous but Notable Esoteric Pedagogies	10
2.6 Analysis of Pedagogies	10
3 Research Approach	13
3.1 Problem Statement	13
3.2 Creation Of Learning Modules	14
3.2.1 Deployability	14
3.2.2 Pattern Selection	15
3.2.3 Video Organization	15
4 Evaluation	20
4.1 Assessment	20
4.1.1 Design Pattern Questions	21
4.1.2 Final Questions	24
4.2 First Classroom Experiment	26
4.3 Second Classroom Experiment	28

4.4	Final Exam Questions for CSC 309 (and 307)	29
4.5	Expected Outcomes	29
5	Results	31
5.1	Analysis of Experimental Results	34
5.2	Results for the Final/Midterm Questions	35
5.3	Student-Reported Results	38
5.4	Comparison to Text-based Tutorials	39
5.5	Threats to Validity	40
5.5.1	Threats to Internal Validity	40
5.5.2	Threats to External Validity	42
6	Conclusion	43
	Bibliography	46
	Appendix: Code and Exercises	52

List of Tables

4.1	Experiment design for CSC 309	27
4.2	Experiment design for CSC 307	29
4.3	Differences between the different kinds of videos.	29
5.1	Experiment Results for Short Questions. The (2) Denotes the Long Video shown in the Second Experiment (CSC 307).	34
5.2	Experiment Results for Long Questions. The (2) Denotes the Long Video shown in the Second Experiment (CSC 307).	34
5.3	309 Final Exam Questions	36
5.4	307 Midterm Questions	37
5.5	Results from subjective questions in both experiments.	39

List of Figures

3.1	Still image from the Strategy video skit.	16
3.2	Slide from the context portion of the Strategy video.	17
3.3	Slide from the structure portion of the Adapter video.	18
3.4	Slide introducing the exercise in the Observer video.	19
4.1	This is the UML diagram provided to the 309 students on the final exam.	25
5.1	Results of Various Methods of Teaching the Strategy Design Pattern.	32
5.2	Results of Various Methods of Teaching the Adapter Design Pattern.	33
5.3	Results of Various Methods of Teaching the Observer Design Pattern.	33

Chapter 1

Introduction

A design pattern is, fundamentally, a pairing between a common problem in software development and a proven solution for that problem [32]. The concepts contained in the seminal book *Design Patterns: Elements of Reusable Object-Oriented Software* (commonly referred to as the Gang of Four, or GoF book) [15] engendered a great deal of recognition from industry, scholarship, and controversy, often in equal measures. The GoF book proposed twenty-three software design patterns, whose implications are the focus of this thesis. This book did not mark the beginning of the concept of design patterns, as the idea of design patterns originated in the field of architecture decades ago [6] and its emergence into software development has long been in the making. Nor does the GoF book mark the be-all and end-all of design patterns

There has been enduring criticism of the patterns in some circles, which includes the patterns' utility and potential for misapplication [16]. However, it is safe to say that such voices are on the fringes of this debate, and that the use of design patterns is definitely mainstream. Tao [32] notes the pervasiveness of design patterns in modern commercial software, something which is also at-

tested to by many other sources, such as Astrachan [6]. It should be entirely uncontroversial to say that design patterns have become a major force in professional software engineering when one further considers several surveys [20] [21] of software professionals which indicate that a knowledge of design patterns is an essential skill in the field. Other sources (like Beck [7]) note that the penetration of design patterns into industrial practice is so profound that the patterns are effectively a shorthand way of referring to complicated design concepts. A knowledge of these patterns is therefore of paramount importance. Those who might still be concerned about the potential downsides of the patterns should take heart in the study by Prechelt [28]. Despite the previously mentioned controversy over the use of design patterns, it would appear that their use is beneficial to the practice of programming. Cooper states that, "Design patterns are a powerful way to structure the interaction between classes in an OO programming language like Java" [11]. Design patterns have successfully insinuated themselves into the professional practice of software development and engineering.

We have discussed how deep design patterns have burrowed themselves into the world of professional software development, which places a mandate upon educators to introduce patterns in the classroom. The popularity of the patterns ultimately comes down to their power in helping to reduce software complexity a little bit by providing proven solutions for common recurrent problems in programming, not unlike how standardized parts aided the Industrial Revolution. Papers on design patterns range in approach from practical, industrial uses of the patterns to how they ought to be taught in an undergraduate curriculum. The study of design patterns pedagogy has proceeded slowly since the publication of the Gang of Four book—with over 13,000 citations in scholarly papers at the time of this writing, there is no denying that the Gang of Four book has spurred an

enormous amount of research on design patterns, though little empirical research on teaching the patterns has been carried out so far.

The popularity and widespread use of design patterns has led some educators to speculate about and experiment on different ways to instruct students about design patterns. The imperative to know and use the patterns is spelled out by Sterkin [30], who argues that design patterns are simply a better way to think about software, and they mix well with the object-oriented paradigm because design patterns encourage modularity and reuse. This line of argument is further advanced by Lang [19], who argues that the idea that design patterns are a part of OO ideology is precisely backward, and that the two concepts are ultimately inseparable. And, finally, there is some evidence that design patterns have an incidental relation to how programming expertise is physically stored within the brain [10]. He makes the assertion that pattern-oriented thinking is therefore rooted in the brain and thus ought to have a prominent role in computer science education. There is no denying that there is a sort of synchronicity between design patterns and OO methodology, and these source serve to underline the necessity of teaching design patterns.

However, this thesis sought to do more than just make a few observations. One of the key insights that this thesis proposes is an enhanced focus on teaching the contexts in which design patterns are to be used, an approach that has generally not been favored by other researchers who have looked at introducing design patterns into the classroom. Section 2 will examine prior attempts to teach design patterns and their underlying motivations. The thesis will then discuss the shortcomings of the current state-of-the-art in Section 3, which will lead naturally into a discussion of how the learning modules that became the lynchpin of this project were devised. Section 4 will discuss the plans for assessing the

learning modules created in this research and the metrics developed to ascertain the efficacy of the modules. After this, the thesis will discuss the design of the experiment to see how effective the modules were. The paper will then segue into a look at the results of the experiment, followed by a conclusion in section 6.

Chapter 2

Related Work

This section discusses prior approaches to teaching design patterns. The section covers published, scholarly, freely available work. There are approaches to teaching design patterns that do not fit these categories, from the reputable (such as corporate training videos) to some that are less so (e.g. YouTube videos). Such things might be useful but are not research.

2.1 Placement In The Curriculum

Despite the general agreement on the importance of design patterns, there is still much disagreement about the particulars of how to teach them. Some researchers, such as Rudolf Pecinovsky, insist on teaching the patterns as early as possible in the curriculum. Pecinovsky's paper "Let's modify the objects-first approach into design-patterns-first" gives some indication about his views of this subject. Other researchers propose putting the patterns much later into an undergraduate curriculum. One of these is Johnson [18], who taught the patterns in an intermediate-level software engineering course. The arguments for the re-

spective approaches follow naturally from their proponents' positions: on the one hand, advocates of "design patterns early" insist that patterns, like objects, are of such tremendous importance to industry that teaching them as early as possible, and getting students to think in terms of patterns, is an important and worthy goal. Waiting too long, they argue, will only calcify bad programming habits in students [26]. On the other hand, opponents note that CS1 is a notoriously difficult class to teach, and that objects early has not been such a smashing success in and of itself [5]. Adding new and sophisticated material to the mix, then, might not be a good idea [34], especially when one considers that Clancy [10] and Dewan [13] both state that the point in the curriculum where design patterns are used does not seem dispositive in determining the success of students' use of them. And there have been some suggestions outside of these categories (such as a proposal for a graduate-level course exclusively about teaching design patterns) whose underlying assumptions seem flawed [31]—this proposal does not even discuss how undergraduate students are to learn patterns.

2.2 Pitfalls to Teaching Design Patterns

So, putting aside the preceding discussion of curriculum placement, how does one go about teaching design patterns? A good starting place would probably be the literature on past attempts to teach design patterns, which tends to be less than voluminous. Perhaps this dearth is due to the novelty of the subject, but also perhaps it is due to the difficulties of setting up effective experiments to assess the effectiveness of teaching the patterns [27]. One of the most significant papers on this subject, which tackles this very question, is the one by Lew Della and David Clark [12]. This paper provided, in order, the two biggest stumbling

blocks to teaching design patterns:

1. The contexts in which the patterns are to be used are difficult for students to understand.
2. The examples used to help learn the patterns are overly complicated.

The ordering is significant here. Design patterns are unusual in that one can understand the structure of a pattern, its constituent parts and their uses, and how they all fit together and still not be able to apply it properly. The proper context of a pattern is, as Della and Clark maintain, the most difficult part of a design pattern to master because it is often incumbent on having experience to know when to apply the pattern. Sterkin [30] agrees on this point, that the ability to understand the context in which a pattern is to be used is the key to success in using design patterns. An interesting experiment is described by Clancy [10], which showed that students who were just given text and diagram descriptions of design patterns struggled to use them effectively, contributing further to the evidence against the notion that giving such descriptions of design patterns to students will be enough to actually get the students to learn the patterns.

2.3 Structuralist Notions of Design Patterns Pedagogy

Based on the preceding sources, a method of teaching design patterns that focused primarily on illuminating the contexts would stand a better method of success. And, yet, approaches of this sort are quite rare. Perhaps this is because most researchers in this area are interested in introducing the patterns in a CS1-

like course. Scholarship in this area has tended to focus on efforts to simplify the structure of design patterns and to simplify the conceptual examples used to teach the patterns - in essence, to make them more user-friendly. These papers (which include, among others, [26, 3, 1, 2, 29, 36, 37, 22]) tend to unfold along predictable lines: simplification, a stress on better examples as the key element to improving performance with design patterns, and largely anecdotal evidence to back up the researchers' claims. It is safe to say that, in terms of volume, published articles on design patterns pedagogy that focus more on structure and examples are more prevalent than those that focus on contexts, as a perusal of the ACM Digital Library will confirm.

2.4 Contextual Pedagogies

Despite the prevalence of what I have dubbed structuralist notions of design pattern pedagogy, there have been some attempts that (in most cases, unwittingly) have employed a context-centered approach. In the textbook realm, the book *Head First Design Patterns* [14] is an example of a way of teaching design patterns that gives special emphasis on a conceptual understanding of design patterns and, in particular, their contexts. However, as was previously mentioned, this is not the prevailing paradigm for teaching patterns.

A method employed by Weiss [35] seems a bit more promising than some of the structuralist pedagogies. Weiss's idea is to teach patterns "by stealth", by which he means without the knowledge of the students. He proposes a multi-stage project in which design patterns are added in successive stages. In this way, patterns are introduced in a context with which students will be familiar. One problem with this approach is that the creation and deployment of learning

modules will necessarily be quite difficult to pull off—structuring such an expansive project for the purpose of using many design patterns makes the idea a bit cumbersome—especially when it comes to creating actual learning modules. Nevertheless, it is an idea that has a couple of devotees [4] [9]. Nevison [25] proposes another approach, which functions similarly to the “stealth” method as previously described, but in which design patterns are taught post hoc (i.e. after a project) as an alternative way of having done the project. Once again, I see an attempt to attack the context problem by introducing the patterns in conjunction with a project that students will already intuitively understand. However, this method does not seem a sufficient way of teaching the patterns. The idea that students could learn all the subtleties of design patterns and their contexts by a quick, after-the-fact retrospective like this seems inferior to having the students actually write some code in conjunction with the patterns. Johnson’s [18] experiment seems to have some similarity with this approach: Johnson introduced some light, written homework on design patterns to the curriculum of an intermediate-level computer science course and saw little change in the pass rate of his course (it had been 56% before the addition of design pattern homeworks, it was 55% afterward). This experiment proves that just a little bit of background on patterns does not seem to make much of an impact on students, although Johnson’s objectives were different and the patterns were just one of many different changes he tried with the class.

2.5 Miscellaneous but Notable Esoteric Pedagogies

It is also worth noting that there have been several attempts to teach design patterns that do not fall neatly into either the explain-and-simplify camp or the focus-on-contexts camp. For example, there is the paper by Callahan [8] that utilizes the Java3D package, coupled with hypertext, to facilitate interactive visualizations of design patterns. In a similar vein, there has also been some research into the idea of teaching design patterns through musical composition by Hamer [17]. These approaches bring a great deal of novelty to the table, in terms of their ambition to try to teach design patterns in ways that, to use a cliché, can aptly be described as outside of the box. However, these papers present little in terms of follow through or concrete results to assess them, and they are somewhat obtuse with regard to some of the important details to the approaches they commend. It is difficult, for example, to tell if Hamer's approach even involves having students write any code.

2.6 Analysis of Pedagogies

To return to the question at hand, why is there such focus on simplifying design patterns and coming up with better examples of their structure? Why is there less of an emphasis, relatively speaking, on focusing on the contexts in which patterns are supposed to be used, as the existing research seems to indicate is the most difficult part of teaching design patterns? One reason might be that trying to simplify design patterns is not very hard, relatively speaking. Coming up with effective conceptual examples that capture the subtleties of the patterns

is substantially more difficult.

Another reason for the surfeit of structuralist pedagogies is that much of the scholarship in this field, as has been previously discussed, has been focused on introducing design patterns into CS1, toward the principle of teaching design patterns alongside objects as early as possible, and at that level such measures are needed. This assumes that teaching design patterns as early as possible is a desirable practice, which would depend on whether teaching the patterns as early as possible facilitates the benefits that its supporters claim. The proponents of “design patterns-early” generally cite two reasons for teaching patterns early in the curriculum: first, they assert that placing design patterns early in the curriculum will foster better coding skills [26] [30], and second, that teaching design patterns and the idea of reuse will better prepare students for industry [36] [37]. In their minds, the earlier design patterns are taught, the better. In any event, such efforts to introduce design patterns as early as possible require not only simplification but teaching of contextual material as well. From looking at prior work in the field, it would seem that targeting intermediate students would be a better course of action, as they would undoubtedly have more comfort with the technical detail that these patterns present. Additionally, targeting the contexts of design patterns—the most difficult part of understanding the patterns according to Della and Clark [12]—is a better logical starting point than focusing on the structure. In a greater sense, though, what is stunning about the literature about teaching design patterns is just how few of the articles and conference proceedings cited in this section have any statistical weight behind them. Much of the validation, such as it is, is anecdotal.

One important work to consider going forward is the framework, proposed by Muller et al [23], which is entitled “Pattern-oriented instruction (POI)” and

focuses particularly on how to teach coding patterns to find solutions to problems. Among other things, POI seeks to introduce individual examples and abstract the pattern from those examples, as well as to compare the results of using one pattern with another and, finally, focusing explicitly on how the contexts of these particular patterns. The results of their study showed that students who employed the POI method were more easily able to identify subtasks and apply solutions than were students who did not use their paradigm. Unlike other researchers who focus solely on the structural aspects of design patterns, Muller actually acknowledges the importance of contexts, and the projects he discusses in the paper are more centered on helping students understand where and how to apply the patterns. While not all of his ideas are used in this paper, Muller's central insight—that through abstraction, conceptual examples, and a context-first focus we can teach design patterns better than with other methodologies—underlies the research in this thesis.

Chapter 3

Research Approach

This chapter will discuss the context-oriented approach to teaching design patterns.

3.1 Problem Statement

The goal of this project was to create an easily-deployable set of context-oriented design pattern learning modules that would be just as effective, if not more so, at teaching design patterns than would a method with a primary focus on structure. Such a way of teaching design patterns has never before been formalized or attempted, and such a method of delivery of design pattern material is not known to be tried.

These modules would be targeted at intermediate-level undergraduate students. This was thought to be superior to efforts targeted at introductory CS students, as students who have a more comprehensive background in concepts such as the object-oriented paradigm would be more comfortable with the level

of technical detail associated with design patterns than students who have only recently learned the function of a for-loop. This base of knowledge would allow more focus on teaching the pattern contexts, which as Dewan noted is among the hardest (if not the hardest) elements of teaching design patterns.

Ideally, these modules will prove to be an effective way of teaching design patterns, and could be useful to an instructor looking to teach design patterns to his or her students; to a manager in industry looking for some quick training for employees; and to researchers looking to enhance the selection of modules available.

3.2 Creation Of Learning Modules

This section discusses some of the key considerations that went into the creation of the learning modules that this research hinged upon—the design patterns videos.

3.2.1 Deployability

One of the most important metrics in creating these learning modules was deployability. I wanted to create learning modules that could easily be used by instructors as part of an in-class lesson, a lab, or as homework. I ultimately decided to create several short instructional videos that would be distributed over the internet. I felt that this setup would most easily facilitate the flexibility with respect to deployment that I sought for this project.

3.2.2 Pattern Selection

We chose to create three videos which covered the Adapter, Observer, and Strategy design patterns, as defined in [15]. These patterns were chosen because of their potential utility to students and because each one lent itself fairly naturally to a conceptual example.

3.2.3 Video Organization

These videos were intended to contain a combination of live-action segments and static slides, which I hoped would make the videos dynamic, enjoyable, and informative in proper proportions. I chose to use non-professional actors in the videos, preferring instead to use upper-division CS students with some industrial experience who would already be familiar with the patterns and terminology.

The videos contain four main sections (acts):

1. A skit that introduces the concept of the pattern in a context entirely unrelated to computer science. For example, I used the iPod as a non-CS example of the Strategy pattern, as it allows dynamic selection of songs, videos, etc., in comparison to the static ordering of only songs on a tape player. In the Adapter sketch, a tape player adapter was used as an example of the Adapter design pattern, as a way of allowing two objects with different interfaces to talk with one another. The Observer sketch used the idea of a lookout for a group of students performing a prank as an example of that pattern. The idea here was to try to get students to understand the most fundamental idea of the pattern first, and systematically introduce more depth. Figure 3.1 shows a still from the initial section of the Strategy

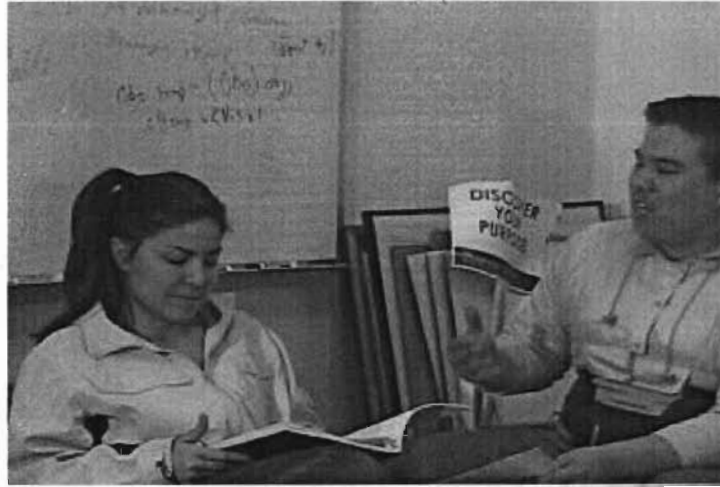


Figure 3.1: Still image from the Strategy video skit.

video.

2. A section that explains the pattern's context specifically within computer science, and where it might be used in a program that they might write. Each of the sections of the video is progressively more concrete than those that precede it. The CS-specific context section is more concrete than the opening skit that precedes it, but less concrete than the following section that deals with what the pattern looks like on a class level. Both this section and the opening skit are primarily focused upon the context in which a design pattern is to be used, although they do this in different ways.

Here is an example of how the CS-specific context section works: in the Adapter pattern module, the video mentions the pattern's utility in code reuse—i.e. two classes from code used from two different sources. Changing all the references in both classes is simply not feasible, so an Adapter is suggested as a better way of solving the problem. Strategy uses the idea of different view classes in an application among which a user can select. And

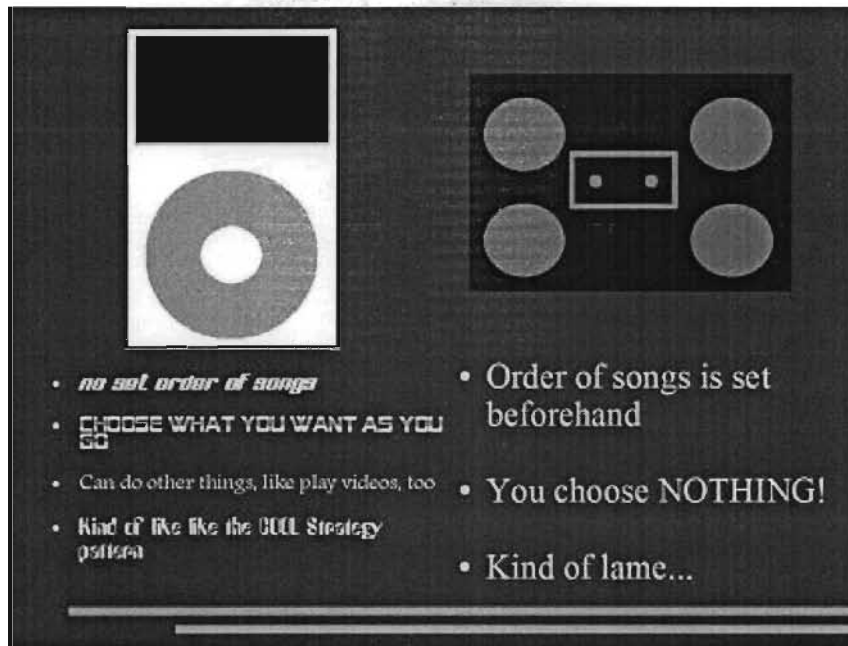


Figure 3.2: Slide from the context portion of the Strategy video.

Observer invokes (and describes) the Model-View-Controller framework as an example of its use within computer science. Figure 3.2 shows a still from this portion of the Strategy video.

3. A section that looks at the structure of the design pattern and how the parts interact with one another. This section introduces the short problem for the pattern. For the Strategy video, students have to match up classes from a code example with classes from the Strategy pattern. For the Adapter video, students have to answer a few short questions about the relationships between the classes in the pattern, and the Observer video asks students to answer why some given code is incorrect. Figure 3.3 shows part of this section from the Adapter video.
4. A section that introduces the longer problem. Students are given a piece of code and will have to refactor it such that it implements the pattern being

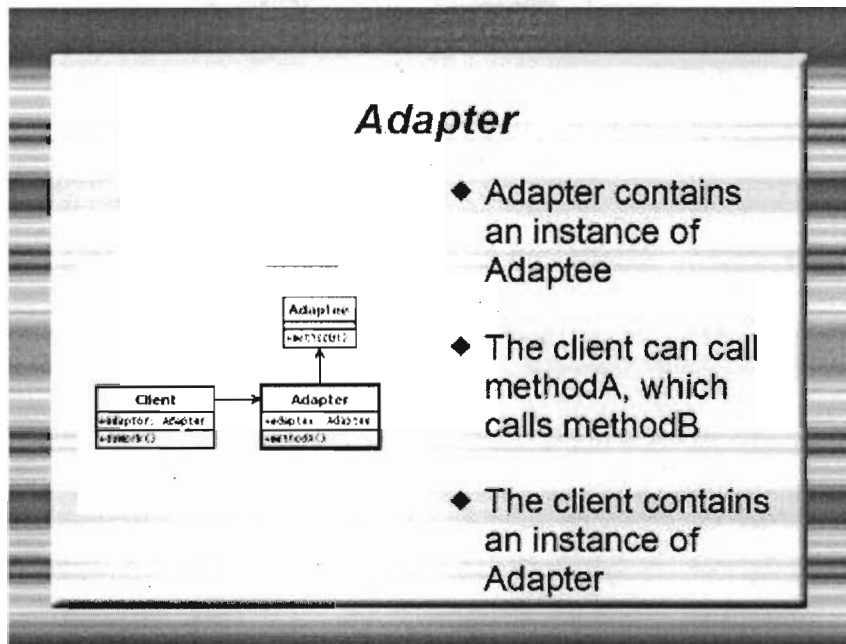


Figure 3.3: Slide from the structure portion of the Adapter video.

taught. A still from the Observer video is shown in Figure 3.4. Our structure stresses the contextual elements of the pattern first. As was previously mentioned, most of the lesson will be context-oriented. Two-thirds of the video aims to inform students of the context, while the other third teaches the structure. This represents a reversal of most research on the subject (which tends to emphasize examples and structure instead of contexts), and whether such a focus turns out to be more effective is at the heart of this project. Each video is about ten minutes in length, and they can be retrieved from <http://users.csc.calpoly.edu/adukovic/DesignPatterns.html>. They are in QuickTime format, which was selected because of its popularity and because the video editing software I used supported output in this format.

Exercise

- ⇒ Main method contains for loop that calls methods at a given iteration
- ⇒ Your task is to implement the observer pattern on this code
- ⇒ Don't worry about threading
- ⇒ Hint: think about what your concrete observers should be and when they should be notified...




Figure 3.4: Slide introducing the exercise in the Observer video.

Chapter 4

Evaluation

The Evaluation chapter summarizes how analyzing the performance of the modules was done. Human Subjects approval was granted under minimal supervision guidelines, and since this work was all anonymous there was no need for signed consent forms. Students were handed a piece of paper containing the pertinent information from the Human Subjects committee.

4.1 Assessment

In this section, assessment of the design patterns videos is discussed. This took place in several phases. On one hand, I wanted to compare the videos to a lecture on the same material to see how well the two methods compare. I performed an experiment that involved instructing a class about three different design patterns using two different types of videos—one which included a skit at the beginning and one that did not—as well as a lecture. I was interested in seeing the extent to which the skit had an effect upon the students' results, if at all. Students answered questions about these patterns, which are described in Section

4.2. The students' scores on these questions constituted the metrics for the first part of the experiment.

In addition, I created some brief exercises for the CSC 309 final exam. These questions were designed to test students' retention of the material, a crucially important element of learning design patterns. Section 4.2 will go over the thinking behind these exercises.

Finally, I performed a final experiment on a different but similar class (CSC 307) which sought to compare the performance of my context-oriented design patterns with a more structuralist model. This, as well as the other experiments, will be further described in the next chapter, and this experiment used the same exact questions as the initial experiment.

4.1.1 Design Pattern Questions

I developed two associated exercises for each pattern: a short exercise (either multiple choice, true/false, or matching) to test students' basic comprehension of the pattern, and a longer exercise to test students' ability to apply the pattern by refactoring an existing piece of code to utilize the design pattern in question. The short and long exercises are intended to take students approximately two and five minutes, respectively.

My three primary objectives with the videos were that the students be able to comprehend the patterns, that they be able to apply them, and that they be able to retain the basic knowledge of what patterns accomplish. I assessed the first two via the exercises previously discussed: comprehension is tested by the student's performance on the shorter question, and application by the longer question. Retention was tested after the fact, with a final exam question that

tested how well students retained the concepts of the design patterns they were taught. The exercises themselves can be found in the appendix. In brief, here's what I asked students to do upon completing the learning modules:

1. For the Strategy design pattern, the short exercise involves matching the parts of a given coding pattern to the elements of the Strategy pattern. The longer exercise involves refactoring a calculator-like program to allow a user to choose an operation to perform.
2. The Adapter pattern's short exercise has the students answer short response and true/false questions that ask which classes communicate with each other in the pattern, and which class (Adapter) contains a reference to which other class (Adaptee). The longer exercise has the student create an Adapter class that will allow two classes to communicate with one another, as well as writing the code to invoke the adapter from within the Client.
3. The shorter question for the Observer pattern asks the student to explain why a given program will not compile (the answer is that the interface lacks a notify method, which does not allow the notifyAll method to work properly). The longer question has students apply the observer pattern to a program that prints out different lines to the console after different amounts of iterations.

The following grading rubric was used for the aforementioned refactoring exercises:

1. For the Strategy exercise, students would receive the following amount of points for each corresponding element:
 - (a) (2) for correct syntax in the Java code.

- (b) (2) for including an Strategy interface.
- (c) (2) for including the decision-making structure in the code.
- (d) (4) for the two ConcreteStrategy classes that should function analogously to the given methods (there are two).

The sum total is ten points

2. For the Adapter exercise, students would receive the following amount of points for each corresponding element:

- (a) (2) for correct syntax.
- (b) (2) for the correct invocation of the method in the Adapter class.
- (c) (1) for creating the new Adapter class.
- (d) (1) if that class contains an instance of the Adaptee class.
- (e) (2) for having the two methods in the Adapter class call the Adaptee's methods.
- (f) (2) if those methods correctly call the Adaptee's methods.

The sum total is ten points.

3. For the Observer exercise, students would receive the following amount of points for each corresponding element:

- (a) (2) for correct syntax.
- (b) (2) for the Observer interface—one point for including it, and one point for including the notify method.
- (c) (3) for including the three ConcreteObserver classes.
- (d) (1) for correctly writing the notifyAll method.

The sum total is eight points.

With this framework, I sought to quantify the effectiveness of the learning modules by identifying the most important elements of the three design patterns for which I created modules, and assigning point values to those elements to generate a numerical score that can easily be visualized and compared with other scores. In the Section 4.2, I will go into greater detail about how the experiment was designed.

4.1.2 Final Questions

The aim of the questions I devised for the final exam for 309, was to test students' retention of the patterns after some time had passed since the experiment. The questions were brief and asked the students to answer simple questions about the patterns that were emphasized in the videos and lecture on the patterns. Retention is an important criterion in determining the success of any learning module that targets design patterns. For this experiment, I operated under the assumption that it was less important that students know every detail about how the patterns work and more important that students know the central ideas of the patterns. Information on the patterns is readily available in books and on the internet, but having the knowledge to know where to use the patterns is key.

The following are the final exam questions devised for CSC 309 and also used for CSC 307 later. Students took the exam roughly a week after the design patterns lesson. Regrettably, the first section of the class took the final on Monday and the second section took the final on Friday, which will no doubt have some effect on the results. This situation could not be avoided.

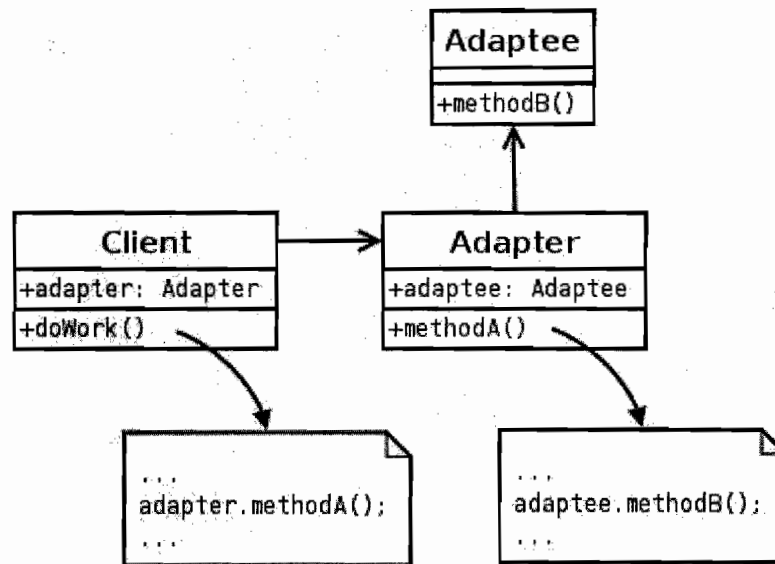


Figure 4.1: This is the UML diagram provided to the 309 students on the final exam.

1. Briefly explain the idea of the Strategy design pattern. What is its purpose and when is it supposed to be used?
2. What service does the Adapter class provide? In other words, what does the Client need it for, and what does it do with the Adaptee?
3. In the above example, it would probably be easier to just change the method call in the Client class to correspond with what is found in Adaptee. Please explain why this is not a good strategy to use for more sophisticated classes that want to communicate with each other.

It should be noted that the final two questions about the Adapter pattern were accompanied by a UML diagram that can be seen in figure 4.1. This was provided because the first question, on Strategy, was a pure recall question. The Adapter questions were meant to test the retention of the relationships and the purpose of that pattern. The “right” answers for these questions involve some version of the responses that follow:

1. The Strategy pattern allows a user to select among different algorithms at runtime.
2. The Adapter class allows the Client and Adaptee classes, who are currently using incompatible interfaces, to communicate.
3. Modifying the code in this fashion would require changing potentially many references in the Client class, which presents a maintenance risk.

4.2 First Classroom Experiment

I conducted a controlled experiment in an undergraduate software engineering course (CSC 309, Software Engineering II), intended for third-year computer science and software engineering majors. Here is the course entry in the Cal Poly Course Catalog:

Continuation of the software lifecycle. Methods and tools for the implementation, integration, testing and maintenance of large software systems. Software development and test environments. Software quality assurance. Group laboratory project. Technical presentation methods and practice.

It is expected that students enrolled in this course will have knowledge of the software lifecycle, requirements and specification. These are taught in the prerequisite course, CSC 308. Additionally, students taking CSC 309 (predominantly juniors) will have experience writing code in at least two different programming languages. The 308-309 series teaches students about the software design process by means of a project to create a software product. This project spans both courses in the series. The experiment that I performed to determine the effectiveness of my videos involved two parallel sections of CSC 309 taught by the

	Pattern	Section 1	Section 2
Activity 1	Strategy	Video with skit	Video w/o skit
Activity 2	Adapter	Lecture	Video with skit
Activity 3	Observer	Video w/o skit	Lecture

Table 4.1: Experiment design for CSC 309

same instructor, and the experiment took place in the tenth (and final) week of instruction of the course in the Winter Quarter of 2008. The first section met from 10:00 a.m. - 12:00 p.m., the second section from 3:00 p.m. - 5:00 p.m.

The experiment proceeds as follows: one section is shown a full video on one of the patterns (i.e. Strategy). After this, the same section will undergo a lecture on another pattern (Adapter) and then will be shown a video on the third pattern (Observer) but without an initial skit. The second lab section will receive parallel instruction on the patterns in the same order, but the methods will be different. In the second lab section, the strategy pattern will be taught with the video minus the skit, followed by the adapter pattern taught by the video with the skit, and concluded with the observer pattern taught by lecture. Table 1 summarizes the experiment organization.

Why proceed in this manner? Before I compare my context-oriented approach of teaching design patterns to what other researchers have done in the past, it is important to make sure that the method I chose to use to create the modules—the videos—does not handicap the students’ ability to comprehend the material. I wanted to make sure that the videos did not represent any real dropoff from other methods of instruction. This was the reasoning behind this experiment.

Just to make sure that the variables in play here are understood: the order in which the patterns are presented is constant, as are the exercises used to evaluate students’ understanding of the patterns. The independent variable for each

pattern is the method of instruction—video with a short skit about the pattern, lecture, or video without the skit. The lecture material will be substantially the same as the video without the skit, and both will still have information on the context in which a pattern is to be used. At this point, I will not have proven my method as being superior to anything—that will be the point of the next experiment.

Unfortunately, the experiment ran a bit long on time in the quick-paced 50-minute lab sessions. As a result, many students were unable to complete the final question on the Observer pattern. Many did not even start. Please see Chapter 5.5 for a fuller discussion on these issues.

4.3 Second Classroom Experiment

After the conclusion of my first experiment on how best to teach the patterns, a second controlled experiment was to compare the context-oriented approach to the structuralist approach advocated by Pecinovsky [26] and his compatriots. The experiment used students enrolled in a different (but similar in terms of experience) course—CSC 307—and will involve splitting up the class and presenting each half of the class a lesson on the same pattern: one half of the section will be presented the material on a pattern with a context-oriented approach, the other with a more structuralist bent. Table 4.2 shows how this worked. Then a different pattern will be presented, and the sections will be switched in terms of the approach (context/structuralist) that is used to teach the pattern. For this experiment, the independent variable is the type of method used to teach the pattern, and the dependent variable is the students' performance on the problems associated with the modules.

	Pattern	Group 1	Group 2
Activity 1	Strategy	Video with skit	Structural Video
Activity 2	Adapter	Structural Video	Video with skit

Table 4.2: Experiment design for CSC 307

Video Type	Skit	CS-Specific Context	Structure	Question
Long Video	x	x	x	x
Short Video		x	x	x
Lecture		x	x	x
Structural Video			x	x

Table 4.3: Differences between the different kinds of videos.

The differences between the videos can be found in Table 4.3.

4.4 Final Exam Questions for CSC 309 (and 307)

The final exam questions were discussed in detail in the prior chapter. These were administered along with the standard final exam for 309. Additionally, the same questions were asked of students in CSC 307 approximately the same length of time after 309 students received them.

4.5 Expected Outcomes

I expect to find that my approach to teaching design patterns, which I have dubbed context-oriented, will be more effective than the prevailing model of teaching the structure of the patterns as the primary aspect of design patterns, or at least as effective as that method. In addition, I hope that the learning modules I create will become widely used among educators and professionals in

the field as a way of introducing these particular patterns.

Additionally, I would like information on how experience factors into the equation—intuitively, I expect more experienced students to do better on the exercises. Information will also be taken on which methods students prefer the most. I would hope that students prefer the longer videos the most.

Chapter 5

Results

The results of these experiments will be broken down as follows: the students' results from the two experiments will be broken down on the long and short questions, separated according to the type of learning module (Lecture, Long Video (i.e. with skit), Short Video (i.e. without skit), and Shorter Video (i.e. Short Video minus the CS-specific context). The results can be seen for the results of all three patterns in the three accompanying figures. Figure 5.1 breaks down the results for the Strategy pattern, figure 5.2 shows the results for the Adapter pattern, and figure 5.3 gives the results for the Observer pattern. The y-axis of the figures represents the amount of points awarded on each long question. Each of these figures shows the relative differences in long question performance between students who learned the design pattern by the various methods. The short questions' results can be found in table 5.1. The table shows the reported scores on the short questions—it was not possible to establish statistical significance on these figures, in part because of the distribution of the data, which was heavily slanted in favor of higher scores. The distribution was uneven. The scores are given in the table—more detail can be found in Section 4.1.1. Table 5.2 contains

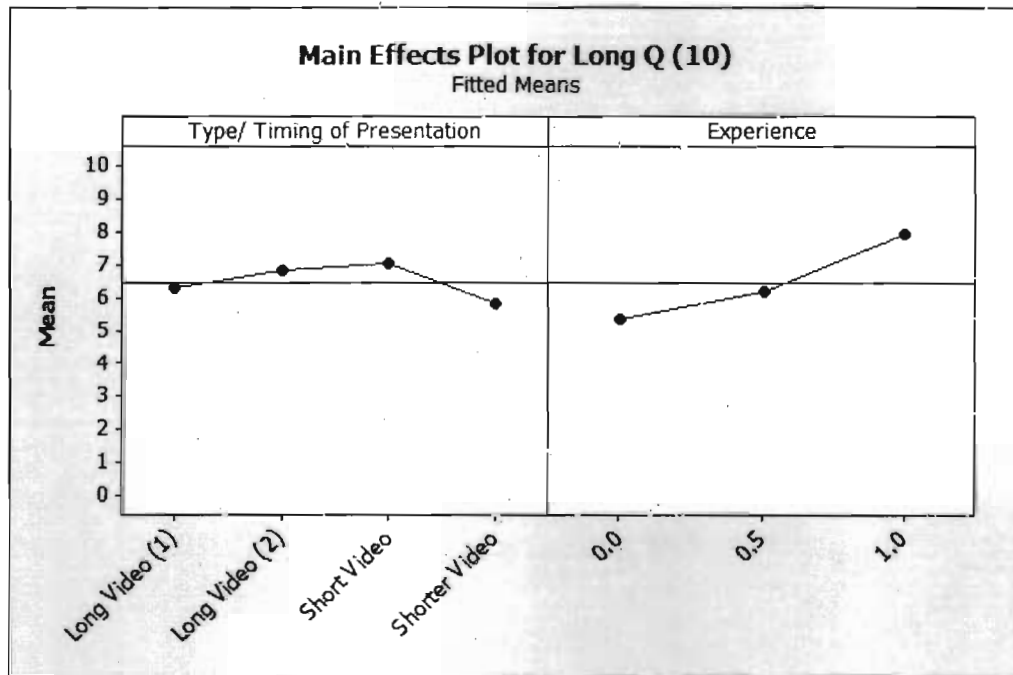


Figure 5.1: Results of Various Methods of Teaching the Strategy Design Pattern.

the results for the long questions.

The three figures that break down results on the long question also contain a second graph to the side that charts students' experience with the patterns. This deserves an explanation. Even within classes, experience can vary widely, so I took data from students about their experience with the patterns in question. This relationship is plotted separately from the main plot, which uses method of instruction as the independent variable. As for the scores, 0 represents no experience, 0.5 represents familiarity with the pattern (but never having used it) and 1 represents past use. The results from the three figures generally show that more experience unsurprisingly translates into greater success with the patterns, although the significance of this relationship cannot be substantiated because of the low number of students who rated a 0.5 on experience.

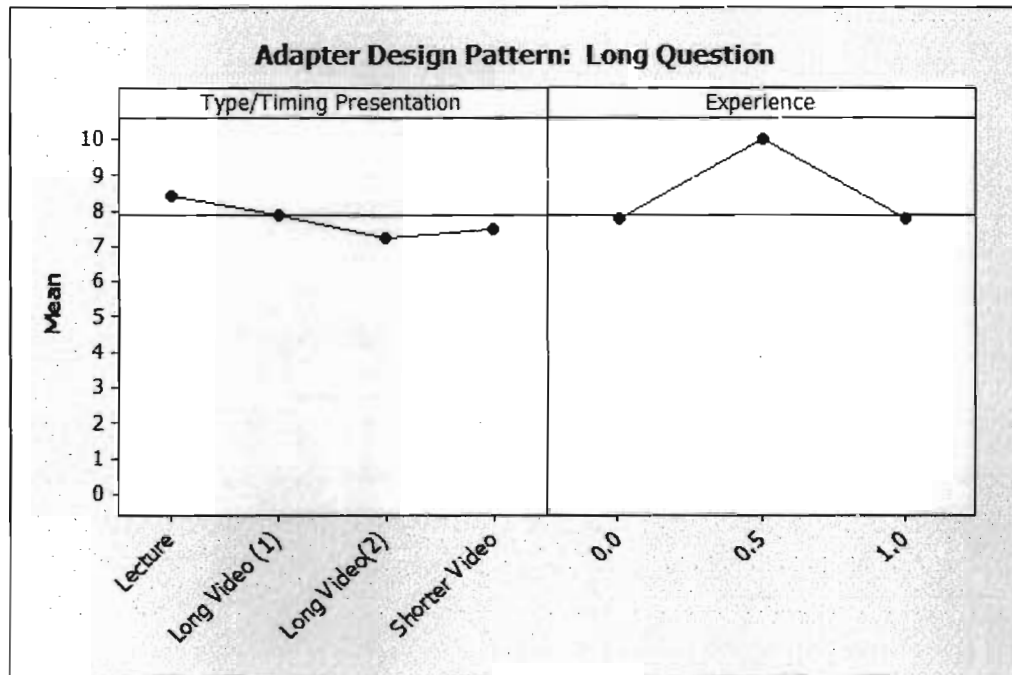


Figure 5.2: Results of Various Methods of Teaching the Adapter Design Pattern.

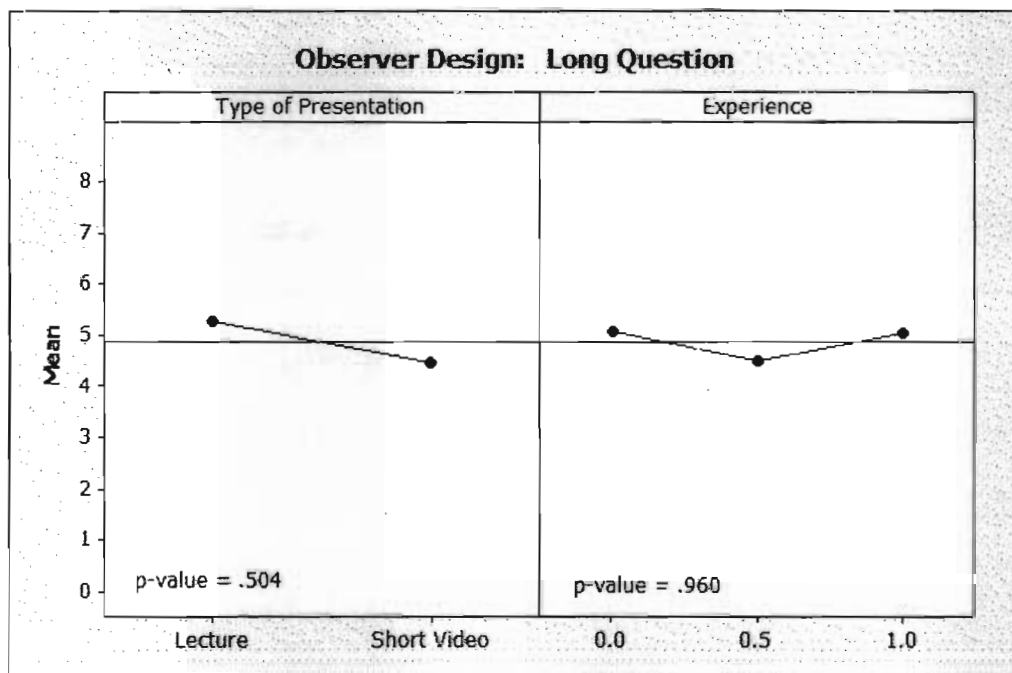


Figure 5.3: Results of Various Methods of Teaching the Observer Design Pattern.

Short Question Results for Type of Video					
	Lecture	Long	Long (2)	Short	Shorter
Strategy	N/A	4.94 / 5	3.45 / 5	4.64 / 5	2.7 / 5
Adapter	3.83 / 4	3.44 / 4	3.55 / 4	N/A	3.00 / 4
Observer Video	0.92 / 1	N/A	N/A	0.68 / 1	N/A

Table 5.1: Experiment Results for Short Questions. The (2) Denotes the Long Video shown in the Second Experiment (CSC 307).

Long Question Results for Type of Video					
	Lecture	Long	Long (2)	Short	Shorter
Strategy	N/A	6.38 / 10	6.18 / 10	7.87 / 10	5.20 / 10
Adapter	8.44 / 10	7.87 / 10	5.80 / 10	N/A	5.45 / 10
Observer Video	5.4 / 8	N/A	N/A	4.56 / 8	N/A

Table 5.2: Experiment Results for Long Questions. The (2) Denotes the Long Video shown in the Second Experiment (CSC 307).

5.1 Analysis of Experimental Results

This section breaks down the results for student-answered questions that were completed immediately after viewing the videos/lectures. These questions test the ability to apply the pattern after having seen the videos.

As can be seen from Figures 5.1, 5.2 and 5.3, it does not appear that the style of presentation had much bearing on the performance of the various students on the design patterns questions. It appeared that lecturing was the most effective method of instruction: on both the long and short questions, sections who received lecture did better on the questions than any other section. On the other end is the shorter video—i.e. the structural video. On each and every attempt to use it, students who learned from the shorter video did worse than any other group. The 307 students not only reported far less experience with the patterns, but they did worse than their 309 counterparts when presented with the long video. The differences between the short and long videos (i.e. the ones lacking a skit and those containing one, respectively) appear to be ambiguous—long video

students performed slightly better on the Strategy short question, while the converse was true on the long question. It does not seem possible to draw a definitive conclusion on the merit of the skit with respect to this criterion, although there is not a statistically significant difference between the difference that showed up under statistical analysis. Please see Chapter 5.5 for a discussion of these issues. The statistical calculations on the datasets for this experiment were performed using Analysis of Variance Between Groups.

In essence, there did not seem to be wildly diverging results between the different methods of instruction. This suggests that the effect of the method of instruction of these design patterns is muted, at least according to this metric.

5.2 Results for the Final/Midterm Questions

Though the results from the initial classroom experiment do not necessarily bode well for my videos, the results from the final questions are significantly more favorable. For the first question (“Briefly explain the idea of the Strategy design pattern. What is its purpose and when is it supposed to be used?”) students in the first section generally got the question correct. Out of 24 students in the class that viewed the Strategy video with the skit, 14 got the answer correct and six got the question incorrect. Out of the 24 students in the second section that saw the video without the skit, only five got the question correct. Fifteen students got the question incorrect. In both sections, four students did not give a response. The results are given in the Table 5.3. The second section of students claimed more experience with strategy and generally did better on the questions than did the first section. Such a contrast cannot help but feel a little hard to swallow considering that the first section took the test four days before the second section.

Problem 1-Strategy		
	Section 1 (Video w/Skit)	Section 2 (Video w/o Skit)
Correct	14	5
Incorrect	6	15
No answer	4	4
Problem 2-Adapter		
	Section 1 (Lecture)	Section 2 (Video w/Skit)
Two parts correct	4	8
One part correct	11	11
No parts correct	6	3
No answer	3	2

Table 5.3: 309 Final Exam Questions

The second section had much more time to forget the material, and surely did.

Nevertheless, the benefits of the videos comes even more into focus when considering how the two sections performed on the questions for the Adapter pattern. If you will recall, section 1 received a lecture on the Adapter pattern, while section 2 received a video with a skit. Section 1 performed better than section 2 on the questions, and as has been mentioned several times, section 1 took the exam several days before section 2.

It turns out, once again, that the section that had a video with a skit did much better than the other section when it came to retaining the fundamental idea of the design pattern in question. If you recall, the Adapter question had two parts (“What service does the Adapter class provide? In other words, what does the Client need it for, and what does it do with the Adaptee?” ”In the above example, it would probably be easier to just change the method call in the Client class to correspond with what is found in Adaptee. Please explain why this is not a good strategy to use for more sophisticated classes that want to communicate with each other.”) All in all, four students in the first (lecture) section got both parts of the answer correct, 11 got one part correct, and six got zero parts correct

Problem 1-Strategy		
	Section 1 (Long Video)	Section 2 (Shorter Video)
Correct	3	4
Incorrect	8	5
Problem 2-Adapter		
	Section 1 (Shorter Video)	Section 2 (Long Video)
Two parts correct	2	3
One part correct	6	4
No parts correct	3	3

Table 5.4: 307 Midterm Questions

(three declined to answer). In the second (video section), eight students got both parts of the question correct, eleven got one part correct, and only three got zero parts correct, with two students leaving the question blank. The results can be found in Table 5.3, under the label "Problem 2-Adapter."

Now, let's briefly discuss the results from the 307 section. They were given the same questions on a midterm, and Table 5.4 has the responses. For this test, nonresponses are counted no differently from zeroes, since everyone in the 307 class attended class the day of the experiment. This is opposed to the 309 class, where 6-7 students missed each course. Unfortunately, the sample sizes are too small to make any determination as to whether the shorter video outstrips the long video on retention. There might have been other factors at play as well—please see section 5.5.

So, what can all of this tell us finally? Section 5.1 showed a decided trend—lecture seemed to be the best alternative in terms of ability to understand and apply the patterns, and the shorter videos—the ones that had a bare-bones structural focus—showed the worst results. There was some ambiguity among the two remaining methods of teaching design patterns: was the introductory skit in the long video beneficial?

As it turns out, this question can now be answered. During the 309 experiment, section 1 received a video with a skit, and section 2 received a video without a skit. Section 1 scored much higher than section 2 did, and the results are statistically significant, with a p-value of 0.0024. *It is thus acceptable to say that, in terms of retention, the long video is superior to the short video.* The students in 309 who had the long video for Adapter also did superior to their counterparts who had a lecture on the same pattern, but the results are not statistically significant ($p = 0.13$), which rules out broader claims. And one cannot make any claims as to the difference between the long video and the shorter video from the results from 307 because of the proximity of the two groups' results, as well as the validity factors. These statistical comparisons were performed by the use of a simple t-test.

In short, it appears that the long video has some merit when it comes to retention.

5.3 Student-Reported Results

This section will cover some of the more subjective data gathered during the experiments. During both experiments, students were asked several questions about the videos as a post-mortem, including questions that asked students to rate on a scale from one to five (five being very good) how well the method of instruction conveyed to use the pattern (i.e. the context); how well the method in question conveyed information about the structure of the pattern; and how much the student liked the presentations of each. I decided to categorize these by type of presentation, and the results of these questions can be seen in Table 5.5.

Method of Instruction	Conveys Context	Conveys Structure	Like
Lecture	3.31	3.30	2.52
Long Video	3.48	3.22	2.98
Short Video	3.09	2.85	2.48
Shorter Video	3.29	2.93	2.64

Table 5.5: Results from subjective questions in both experiments.

As one can see from the reported results, students felt that the long video was superior in terms of conveying the context, and they just plain liked it more than the other methods of instruction. The lecture just narrowly beat out the long video in terms of conveying structure. These results indicate that, at least in the opinions of the students who participated in the experiments, that the long video succeeds at its main goal—focusing on the context—and it’s a more satisfying experience overall. However, these results should perhaps be taken with a grain of salt, as statistical validation is once again hampered by the small sample size for the shorter video.

5.4 Comparison to Text-based Tutorials

As a part of this experiment I worked with John Dalbey’s CSC 305 class to compare the videos with a text tutorial I found on the internet by Bob Tarr at the University of Maryland [33]. CSC 305 is slightly different than CSC 307 and 309—the focus is on individual, rather than group, programming, although the same prerequisites apply to both classes, and both are junior-level classes. For this part of the project I created an online survey. Students would either watch the video or read through the tutorial first (both focused on the Strategy pattern), answer the short matching question, and then look at the other method of instruction and answer the following questions:

1. Which of the two modes of teaching—the text tutorial or the video—did you feel did a better job at conveying the basic ideas of the design pattern?
2. Which of the two modes did you find more enjoyable?
3. Which did you prefer overall?

Needless to say, the tutorial did not emphasize contexts. The point of the survey was just to get a sense of whether students would prefer to learn by text or by video.

Only 10 people filled out the survey. These data will therefore have to be descriptive statistics. Nevertheless, virtually everyone got the answers correct for the short question, aside from one text student. The responses were evenly split as to which of the two methods was better at conveying the basic idea of the pattern, while six of the students said they enjoyed the video experience more. Nevertheless, six out of ten thought the text tutorial was better overall. The results were a bit inconclusive, which might have been due to the low rate of response.

5.5 Threats to Validity

This chapter discusses the various threats to validity that this study faces.

5.5.1 Threats to Internal Validity

The following are threats to internal validity:

1. There is some danger that the students in this course already know the

design patterns presented, but I control for this eventuality by having students state on a questionnaire whether or not they have already used the patterns.

2. The p-values for the experimental results are high. This might suggest that the method of instruction has a minor effect upon students' ability to learn the patterns (at least, with respect to being able to apply the patterns) or it might mean that the metrics used in this paper need to be reevaluated. For Strategy and Observer, the p-values for the test scores across the different modes of testing were 0.674 and 0.504. In the case of the Adapter pattern it was not possible to extract p-values because the data did not fit the ANOVA model—the quantity of perfect scores on the distribution of data made the model fail.
3. In both sections of the initial experiment, the experiment ran long. Many students did not attempt the questions pertaining to the Observer pattern, while few left questions blank for the other patterns. I excluded the missing answers from the calculation of scores on Observer, but the results nevertheless need to be taken with an additional grain of salt.
4. Several students in CSC 307 complained after the fact that they had trouble hearing the videos.
5. It is assumed that the 309 sections were roughly equivalent in terms of the students' respective GPAs, skill sets, etc. This is not certain, and might pose a threat to validity.

5.5.2 Threats to External Validity

Here are a few possible threats to external validity:

1. Cal Poly students might differ in terms of their academic acumen from other schools. Additionally, Cal Poly's "learn by doing" approach might cause different results from schools that are more focused on theory.
2. Prerequisites for software engineering classes might vary at different institutions, and students taking an equivalent class at another university might be differently equipped skill-wise.
3. Cal Poly has smaller class sizes than many universities. This might play an effect if a professor has to lecture many more people than instructors do at Cal Poly. The sample sizes of these experiments were too small to establish statistical significance. This was partially due to the aforementioned small class sizes, and the logistics of getting instructors to cooperate with this experiment. Other institutions might or might not have such problems.

Chapter 6

Conclusion

In this paper I have written about the idea of a context-oriented method of teaching design patterns, and I have described a set of learning modules that I have created to teach design patterns according to this method, which I feel is a better way of thinking about design patterns. As design patterns continue to become more of an essential piece of software engineering, the necessity to teach these patterns becomes ever more paramount. I submit to you a way of thinking about design pattern pedagogy, in hopes that it will spur further interest and research in the area.

During my research for this thesis I became aware of some vital deficiencies among other attempts to teach design patterns: nearly every published paper avoided the proper teaching of the contexts in which design patterns are meant to be used, and few of the published attempts to teach design patterns were easily deployable. At the outset the idea for this thesis was to create some easily deployable learning modules, which eventually came to mean some instructional videos, but as the project progressed I became more interested in the question of contexts, which I incorporated into the modules at an early phase of the project.

The creation of the modules was accomplished with some assistance from Cal Poly MDS and the CSC Fee Committee, as well as some online tips such as [24]. The videos for the Adapter, Strategy, and Observer patterns were completed on time and below budget, despite the usual (and expected) setbacks in a project of this nature. The assessment of the videos showed that students tended to respond more to a lecture about a design pattern than to a video about the same pattern and performed better on the corresponding exercises, but the dropoff was minor. On the other hand, students viewing videos that included short introductory skits about a design pattern tended to retain information on that pattern at a significantly better rate. And the contextual information provided in the videos turned out to be valuable: students who viewed videos that included material on design pattern contexts did much better on exercises than students who did not see that material. In short, while the results are not uniformly glowing for my context-oriented design pattern multimedia learning modules, the indication is that my approach has some definite built-in advantages to other approaches, and that much future work on this subject remains to be done to ascertain the impact of context-oriented design patterns pedagogy.

In the final analysis, the central problem this thesis sought out to tackle was to develop a deployable set of context-oriented learning modules. This thesis proves that such a set of modules was, indeed, created. As design patterns continue to insinuate themselves into the professional arena the import of this work will become progressively more salient, and it is my hope that this thesis will spur along more experiments and more investment in the subfield of design patterns pedagogy and the context-oriented paradigm for teaching design patterns.

Finally, it is my hope that instructors would find these videos a useful tool in teaching design patterns: while not perfect, they have been proven to be effective

according to my metrics, and it has been shown that they can meet reasonable expectations. In conclusion, teaching design patterns is a hard task, and it will continue to be hard. I can only hope that my thesis provides some insight so that future researchers will have an easier go of it.

Bibliography

- [1] C. Alphonse and P. Ventura. Object orientation in cs1-cs2 by design. In *ITiCSE '02: Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 70–74, New York, NY, USA, 2002. ACM.
- [2] C. Alphonse and P. Ventura. Using graphics to support the teaching of fundamental object-oriented principles in cs1. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 156–161, New York, NY, USA, 2003. ACM.
- [3] F. Arcelli, S. Masiero, and C. Raibulet. Elemental design patterns recognition in java. *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*, pages 196–205, 24-25 Sept. 2005.
- [4] O. Astrachan. Oo overkill: when simple is better than not. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 302–306, New York, NY, USA, 2001. ACM.
- [5] O. Astrachan, K. Bruce, E. Koffman, M. Kölling, and S. Reges. Resolved: objects early has failed. In *SIGCSE '05: Proceedings of the 36th SIGCSE*

- technical symposium on Computer science education*, pages 451–452, New York, NY, USA, 2005. ACM.
- [6] O. Astrachan, G. Mitchener, G. Berry, and L. Cox. Design patterns: an essential component of cs curricula. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 153–160, New York, NY, USA, 1998. ACM.
- [7] K. Beck, R. Crocker, G. Meszaros, J. Coplien, L. Dominick, F. Paulisch, and J. Vlissides. Industrial experience with design patterns. *Software Engineering, 1996., Proceedings of the 18th International Conference on*, pages 103–114, 25-29 Mar 1996.
- [8] M. Callaghan and H. Hirschmüller. 3-d visualisation of design patterns and java programs in computer science education. In *SIGCSE Bull.*, volume 30, pages 37–40, New York, NY, USA, 1998. ACM.
- [9] H. B. Christensen. Implications of perspective in teaching objects first and object design. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 94–98, New York, NY, USA, 2005. ACM.
- [10] M. J. Clancy and M. C. Linn. Patterns and pedagogy. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 37–42, New York, NY, USA, 1999. ACM.
- [11] J. W. Cooper. Using design patterns. *Commun. ACM*, 41(6):65–68, 1998.
- [12] L. Della and D. Clark. Teaching object-oriented development with emphasis on pattern application. In *ACSE '00: Proceedings of the Australasian con-*

- ference on Computing education*, pages 56–63, New York, NY, USA, 2000. ACM.
- [13] P. Dewan. Teaching inter-object design patterns to freshmen. *SIGCSE Bull.*, 37(1):482–486, 2005.
- [14] E. Freeman, E. Freeman, B. Bates, and K. Sierra. *Head First Design Patterns*. O’Reilly Media, Inc., 1 edition, 2004.
- [15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
- [16] P. Graham. Revenge of the nerds. Available at <http://www.paulgraham.com/icad.html>.
- [17] J. Hamer. A musical approach to teaching design patterns. In *ITiCSE ’02: Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 197–197, New York, NY, USA, 2002. ACM.
- [18] C. W. Johnson and I. Barnes. Redesigning the intermediate course in software design. In *ACE ’05: Proceedings of the 7th Australasian conference on Computing education*, pages 249–258, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.
- [19] J. E. Lang, B. R. Bogovich, S. C. Barry, B. G. Durkin, M. R. Katchmar, J. H. Kelly, J. M. McCollum, and M. Potts. Object-oriented programming and design patterns. *SIGCSE Bull.*, 33(4):68–70, 2001.
- [20] T. C. Lethbridge. What knowledge is important to a software professional? *Computer*, 33(5):44–50, 2000.

- [21] T. L. Lewis, M. B. Rosson, and n. Manuel A. Pérez-Qui' What do the experts say?: teaching introductory design from an expert's perspective. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 296–300, New York, NY, USA, 2004. ACM.
- [22] G. Licea, J. R. Juarez, L. G. Martinez, and L. Aguilar. Toward a deeper level of programming expertise for engineering students. *Computer Science, 2006. ENC '06. Seventh Mexican International Conference on*, pages 180–190, Sept. 2006.
- [23] O. Muller. Pattern oriented instruction and the enhancement of analogical reasoning. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 57–67, New York, NY, USA, 2005. ACM.
- [24] G. Network. Your guide to better movie lighting. Available at http://www.g4tv.com/techtv/vault/features/41742/Your_Guide_to_Better_Movie_Lighting.html.
- [25] C. Nevison and B. Wells. Teaching objects early and design patterns in java using case studies. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 94–98, New York, NY, USA, 2003. ACM.
- [26] R. Pecinovský, J. Pavlíčková, and L. Pavlíček. Let's modify the objects-first approach into design-patterns-first. *SIGCSE Bull.*, 38(3):188–192, 2006.
- [27] R. Porter and P. Calder. Patterns in learning to program: an experiment? In *ACE '04: Proceedings of the sixth conference on Australasian computing ed-*

- ucation, pages 241–246, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
- [28] L. Prechelt, B. Unger, W. Tichy, P. Brossler, and L. Votta. A controlled experiment in maintenance: comparing design patterns to simpler solutions. In *Software Engineering, IEEE Transactions on*, volume 27, pages 1134–1144, Dec 2001.
- [29] D. Reed. Incorporating problem-solving patterns in cs1. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 6–9, New York, NY, USA, 1998. ACM.
- [30] A. Sterkin. Teaching design patterns. Available at <http://www.hadassah-col.ac.il/cs/staff/asterkin/advCPlusProg/Teachinggn>
- [31] S. Stuurman and G. Florijn. Experiences with teaching design patterns. *SIGCSE Bull.*, 36(3):151–155, 2004.
- [32] Y. Tao. Teaching software tools via design patterns. In *ACSE '00: Proceedings of the Australasian conference on Computing education*, pages 248–252, New York, NY, USA, 2000. ACM.
- [33] B. Tarr. The state and strategy patterns. Available at <http://userpages.umbc.edu/tarr/dp/lectures/StateStrategy.pdf>.
- [34] E. Wallingford. Toward a first course based on object-oriented patterns. *SIGCSE Bull.*, 28(1):27–31, 1996.
- [35] S. Weiss. Teaching design patterns by stealth. *SIGCSE Bull.*, 37(1):492–494, 2005.

- [36] M. R. Wick. Kaleidoscope: using design patterns in cs1. *SIGCSE Bull.*, 33(1):258–262, 2001.
- [37] M. R. Wick. Teaching design patterns in cs1: a closed laboratory sequence based on the game of life. In *SIGCSE Bull.*, volume 37, pages 487–491, New York, NY, USA, 2005. ACM.

Appendix: Code and Exercises

```
Design Pattern Coding Patterns
Strategy Design Pattern Short Example
//Processor.java
public class Processor
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        TextOperation op = null;
        String option = sc.nextLine();
        if (option.equals("wordcount"))
        {
            op = new WordCount(this);
        }
        else if (option.equals("replaceword"))
        {
            op = new ReplaceWord(this);
        }
        else if (option.equals("printpreview"))
        {
            op = new PrintPreview(this);
        }
        op.initiate();
        //Do other stuff...
    }

    //From TextOperation.java
    public interface TextOperation
    {
        void initiate();
    }
}
```

```

    //From WordCount.java
public class WordCount implements TextOperation
{
    private Processor proc;

    public WordCount(Processor proc)
    {
        this.proc = proc;
    }

    public void initiate()
    {
        //Count words here
    }
}

//From ReplaceWord.java

public class ReplaceWord implements TextOperation
{
    private Processor proc;

    public ReplaceWord(Processor proc)
    {
        this.proc = proc;
    }

    public void initiate()
    {
        //Count words here
    }
}

//From PrintPreview.java
public class PrintPreview implements TextOperation
{
    private Processor proc;
    public PrintPreview(Processor proc)
    {
        this.proc = proc;
    }
}

```

```

    public void initiate()
    {
        //Preview the print version of the document here
    }
}
Strategy Short Question

```

Please match the following numbered items from the preceding example with the corresponding lettered Strategy elements.

- 1) _ WordCount
- 2) _ Processor
- 3) _ PrintPreview
- 4) _ TextOperation
- 5) _ ReplaceWord

- a) Context
- b) Strategy
- c) Concrete Strategy

Strategy Long Exercise

//This is the Code that needs to be fixed up...we want to be
//able to choose an operation to perform!

```
package exercises;
```

```
import java.util.Scanner;
```

```

public class Calculator {
    public static void main(String[] args)
    {
        Calculator calc = new Calculator();
        int one, two;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        one = sc.nextInt();
        System.out.print("Enter another number: ");
        two = sc.nextInt();
    }
}

```

```

        System.out.println("Add: " + calc.addition(one, two));
        System.out.println("Sub: " + calc.subtraction(one, two));
    }
    public int addition(int one, int two)
    {
        return one + two;
    }
    public int subtraction(int one, int two)
    {
        return one - two;
    }
}

```

//Answer Space

```

public class Calculator
{
    public static void main(String[] args)
    {
        //Fill in the blank on the next line

        _ = null;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int first = sc.nextInt();
        System.out.print("Enter another number: ");
        int second = sc.nextInt();
        System.out.println("Enter an operation: ");
        //Fill in this part

    }
}

```

Adapter Design Pattern Example

```

//This is the (not functional) code before the Adapter pattern
//is applied.
//TapeAdapter.java
package examples;

/**
 * A class representing a tape adapter.

```



```

*/
public class TapeAdapter {
    public static void main(String[] args)
    {
        TapeDeck deck = new TapeDeck();
        //These two calls will not work!
        deck.play();
        deck.forward(20);
    }
}

//TapeDeck.java
public class TapeDeck
{
    public void playTape() { }
    public void rewind(int time) { }
    public void fastForward(int time) { }
}

//This is the code after the Adapter pattern is applied.

package examples;
/**
 * A class representing a tape adapter.
 * This is the Client.
 */
public class TapeAdapter {
    public static void main(String[] args)
    {
        TapeAdapterFixed deck = new TapeAdapterFixed();
        deck.play();
        deck.forward(20);
    }
}

/**
 * A class representing a tape deck.
 * This is the Adaptee.
 */
public class TapeDeck
{
    public void playTape() { }
    public void rewind(int time) { }
    public void fastForward(int time) { }
}

```

```

}

/**
 * This is the Adapter.
 */
public class TapeAdapterFixed
{
    TapeDeck deck;
    public TapeAdapter() {deck = new TapeDeck(); }
    public void play() {deck.playTape(); }
    public void forward(int time) {deck.fastForward(time); }
    public void rewind(int time) {deck.rewind(time); }
}

```

Adapter Short Questions

Please answer the following questions about the Adapter Design Pattern.

1. What are the names of the two classes (out of Adapter, Adaptee, Client) that want to communicate with each other, but cannot?
2. The Client calls the Adapter class, true or false?
3. True or false, the Adapter class contains a reference to an object of the Adaptee type?

Adapter Long Exercise

//This is the Code that needs to be fixed up

```

package exercises;
public class SomeClass {
    public static void main(String[] args)
    {
        SomeOtherClass x = new SomeOtherClass();
        double pi, e;
        pi = x.computePi();
        e = x.computeE();
        System.out.println("Result: " + (e * pi));
    }
}

class SomeOtherClass {

```

```

public double getPi()
{
    return Math.PI;
}

public double getE()
{
    return Math.E;
}
}

/**
 * This class needs only one minor change.
 */
public class SomeClass
{
    public static void main(String[] args)
    {
        //SomeOtherClass x = new SomeOtherClass();
        //Fill in the line that replaces the line above
        double pi, e;
        pi = x.computePi();
        e = x.computeE();
        System.out.println("Result: " + (e * pi));
    }
}

/**
 * This class need not be touched.
 */
class SomeOtherClass {

    public double getPi()
    {
        return Math.PI;
    }

    public double getE()
    {
        return Math.E;
    }
}
pagebreak

```

Observer Example

```
package examples;
```

```
import java.util.ArrayList;
```

```
//Observer Example
```

```
public class ObserverExample {
    ArrayList<Observer> list = new ArrayList<Observer>();
    int one = 17, two = 13;
    public static void main(String[] args)
    {
        ObserverExample oe = new ObserverExample();
        oe.list.add(new ObserverOne());
        oe.notifyAllObservers();
        oe.one--;
        oe.notifyAllObservers();
    }

    public void notifyAllObservers()
    {
        for (Observer o : list)
        {
            o.notify(one, two);
        }
    }
}
```

```
interface Observer
```

```
{
    void notify(int x, int y);
}
```

```
class ObserverOne implements Observer
```

```
{
    public void notify(int x, int y)
    {
        if ((x + y) % 10 == 0)
        {
            System.out.println("Divisible by Ten");
        }
        else
    }
}
```

```

        {
            System.out.println("Nondivisible by Ten");
        }
    }
}

```

Observer Long Exercise

package exercises;

public class Looper {

public static void main(String[] args)

```

{
    for (int i = 0; i < 100; i++)
    {
        if (i == 5)
        {
            method1(i);
        }
        else if (i == 10)
        {
            method2(i);
        }
        else if (i == 20)
        {
            method3(i);
        }
    }
}

```

public static void method1(int x)

```

{
    System.out.println("Method 1: " + x);
}

```

public static void method2(int x)

```

{
    System.out.println("Method 2: " + x);
}

```

public static void method3(int x)

```

{
    System.out.println("Method 3: " + x);
}

```

```
}
```

```
//Write New classes here!
```