NET OBJECTIVES EZINE
VOLUME 1, ISSUE 7 JULY 2004

Commonality and Variability Analysis

Alan Shalloway, Net Objectives, alshall@netobjectives.com

James R.Trott, Millennium Relief & Development Services, jtrott@mrds.org

(Chapter 15 from the Second Edition of their book: "Design Patterns Explained: A New Perspective on Object-Oriented Design" released October 2004)

Design patterns do not exist in isolation, but work in concert with other design patterns to help you create more robust applications. In this book, you will gain a solid understanding of twelve core design patterns and a pattern used in analysis. You will gain enough of a foundation that you will be able to read the design pattern literature, if you want to, and possibly discover patterns on your own. Most importantly, you will be better equipped to create flexible and complete software that is easier to maintain.

Overview

In this chapter

I will show how to use Commonality and Variability Analysis (CVA) to develop a high-level application design. Although design patterns can't be used in all designs, the lessons learned from them can. One of the most important of these lessons is that you can identify variation in your system using CVA. You can then follow the lessons of design patterns (program to interfaces, encapsulate variation using aggregation) to create designs that are flexible and easily testable.

Note from Alan Shalloway:

When Design Patterns Explained first came out I felt we were on the edge of something with Commonality – Variability Analysis. Unfortunately, we had to get the book out. This was always an incompletion for me. Our second edition has a great chapter on CVA and here it is!

Articles and events of interest to the software development community

In This Issue:

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Commonality and Variability Analysis and Application Design

Isolating Variation is a Design Pattern Philosophy

Experienced developers know that when adding new function to an existing system, the major cost is often not in writing the new code, but in integrating it into the existing system. The reason for this is that the pieces in most existing systems are fairly tightly coupled. We must eliminate, or greatly limit, this coupling. One reason this occurs is that developers often consider how entities relate to each other before they are clear what the right entities are. From my experience of training people at all different levels of competency, I have come to the conclusion that more experienced developers do this even more than inexperienced developers. Developers need a way first to identify what they have before trying to find the relationships involved.

I suggest that you design applications in the following way: First, use CVA to identify the concepts (commonalities) and concrete implementations (variabilities) that are present in the problem domain. At this point we are mostly interested in identifying the concepts here, but many variabilities will be identified as part of this process. Any entities in the problem domain that are not included in these concepts (e.g., there may be some one-of-a-kind objects present) should also be identified. Next, once the concept for the functionality you need has been identified, you go on to specify the interface for the abstraction that encapsulates this. Derive this interface by considering how the concrete implementations derived from this abstraction will be used.

This approach basically follows Alexander's contextual design approach which is incorporated into the previously mentioned "Dependency Inversion Principle". By defining these interfaces, you are also determining which object use which objects – completing the specification of the design.

Let's see this in action with the CAD/CAM problem.

Solving the CAD/CAM Problem with CVA

Finding the concepts (and therefore abstract classes)

When analyzing the problem domain with CVA, I want to see what concepts are there and then try to organize these pieces as cohesively as possible. Remembering the CAD/CAM system, there are:

- different CAD/CAM systems – V1, V2. In this situation, these are essentially read-only, proprietary databases that will provide the numerical control sets the expert system needs to do its work.
- different kinds of **`Features`** – slots, holes, cutouts, special and irregular.
- different kinds of **`Model`**s – V1-based and V2-based.

What's Happening at Net Objectives This Month

Besides the abundance of seminars we always offer, we are starting a study group on Modern C++, a great book on C++ generic programming. Check out our discussion group at **http://netobjectivesgroups.com/eve/ubb.x?a=cfrm&s=8846083791&f=904106756** *if you are interested.*

To say there are different CAD/CAM systems really means that the concept is "CAD/CAM system" and the variations of it are "V1" and "V2". CVA leads to the following commonalities and their corresponding variations:

commonality: CAD/CAM system variations: V1 V2	commonality: features variations: slot hole cutout special irregular	commonality: model variations: V1-based V2-based

Figure 15-1. Commonality and Variability Analysis Table

An alternative approach is to pick any two items in the problem domain and ask the following questions:

1. Is one of these a variation of the other?

2. Are both of these a variation of something else?

For example, I might notice that there are features and slots. A slot is a kind of feature. I guess that "features" is a commonality and "slots" is a variation of it. Or, I might see that there are slots and holes. Within this problem domain, they do not seem to be variations of each other. They both seem to be variations of "features". Or I might compare the V1 CAD/CAM system and slots. There does not seem to be anything in common with them.

One issue per commonality

Of course, it is not always this simple. I may collapse concepts without realizing it. For example, suppose you think about the problem domain as having **`V1Slots`**, **`V1Holes`**, **`V2Slots`**, **`V2Holes`**, etc. The commonality would seem to be "CAD/CAM features". But this is commonality with two concepts: "CAD/CAM version" and "features". CVA

About the authors --

Alan Shalloway

Alan Shalloway is the CEO and senior consultant of Net Objectives. Since 1981, he has been both an OO consultant and developer of software in several industries. Alan is a frequent speaker at prestigious conferences around the world, including: SD Expo, Java One, OOP, OOPSLA. He is the primary author of Design Patterns Explained: A New Perspective on Object-Oriented Design and is currently co-authoring three other books in the software development area. He is a certified Scrum Master and has a Masters in Computer Science from M.I.T.

James Trott

James Trott is a knowledge management consultant, collaboration specialist, and knowledge engineer. He has Master of Science in Applied Mathematics, an MBA, and a Master of Arts in Intercultural Studies. He has twenty years experience as a pattern-based analyst, working in the petroleum, aerospace, software, and financial industries. He divides his consulting efforts between corporate work and humanitarian work with Millennium Relief and Development Services.

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

says that commonalities should really be based on one issue per commonality. Otherwise I will not have strong cohesion in my design. Recognizing that I have two commonalities - CAD/CAM and Features – should lead me to ask which variations of these commonalities do I have? When I do this I come up with the variations I listed in Figure 15-1. This is one of the values of CVA: it results in cohesive concepts.

Representing the concepts

Figure 15-2 is a reproduction of Figure 6-5.

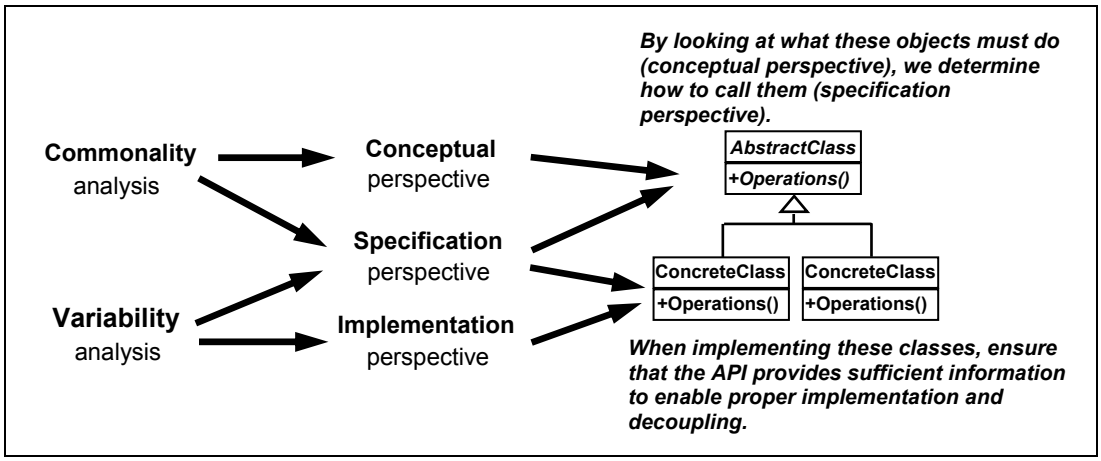

Figure 15-2 The relationship between commonality and variability analysis, perspectives, and abstract classes.

The information in Figure 15-1 can be translated into three different class hierarchies by following the guidelines laid out in Figure 15-2. These are shown in Figure 15-3.

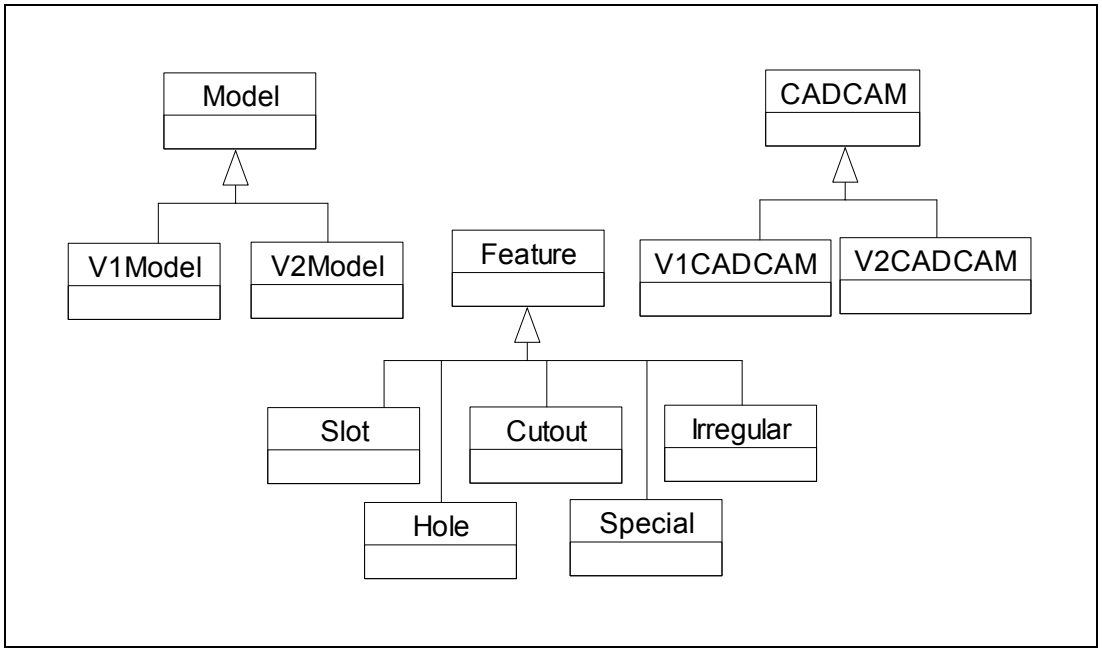

Figure 15-3 Translating CVA table into classes.

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Relating the concepts

The next step involves determining how the concepts relate to each other. Models contain **`Features`** and **`Features`** are extracted from the CAD/CAM system. When I was designing this system, I thought it would be simpler if I built stand-alone **`Features`** that contained all of the information that had been in the CAD/CAM system that related to them. In other words, the CAD/CAM system was like a database to the Features – it contained the information about the **`Features`**. The **`Features`** presented the appropriate methods to make this information available, but the **`Features`** extracted this info from the CAD/CAM to get it. Models would also have to relate to the CAD/CAM system. Based on this analysis, the next level of detail is shown in Figure 15-4.

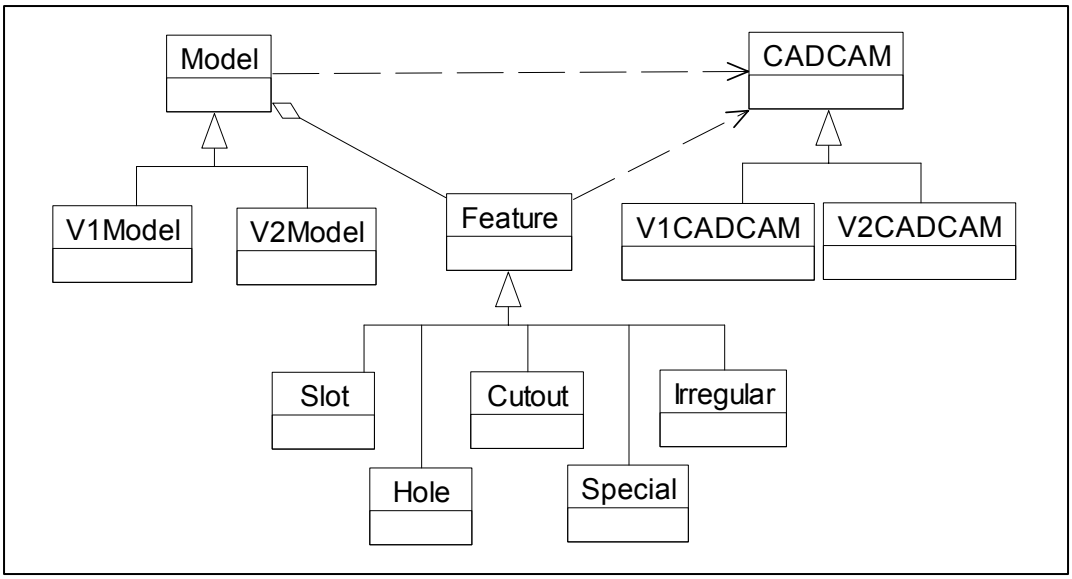

Figure 15-4 Our class diagram showing relationships between the classes.

I am faced with a design decision here. Do I want to have different types of models or have one type of model that uses the different CAD/CAM systems through the **`CADCAM`** interface? In other words, if I move the methods in **`V1Model`** and **`V2Model`** that are peculiar to the CAD/CAM system into the **`CADCAM`** class hierarchy, I can probably avoid having different types of **`Model`**s. This approach seems superior because Models use the CAD/CAM systems to implement them, but the concepts in **`Model`** are not inherently part of the CAD/CAM system. I show this approach in Figure 15-5. Don't get hung up on this distinction: there is not a great difference in code quality between the two of them as long as you make sure you do not have any redundancy. Personally, I like the solution shown in Figure 15-5 better because it has classes that are more cohesive. Therefore, I will use that for the rest of the design.

Personal Book Recommendation from Alan Shalloway:

I've just finished reading Wealth 101: Wealth Is Much More Than Money by John-Roger and McWilliams (ISBN 0-931580-52-8). This is a great book about creating your life the way you want it to be (not necessarily about how much money you have). Part of the Life 101 series which I also recommend. I have just completed it and already am seeing results from my efforts.

Volume 1, Issue 7
July 2004

NET OBJECTIVES

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

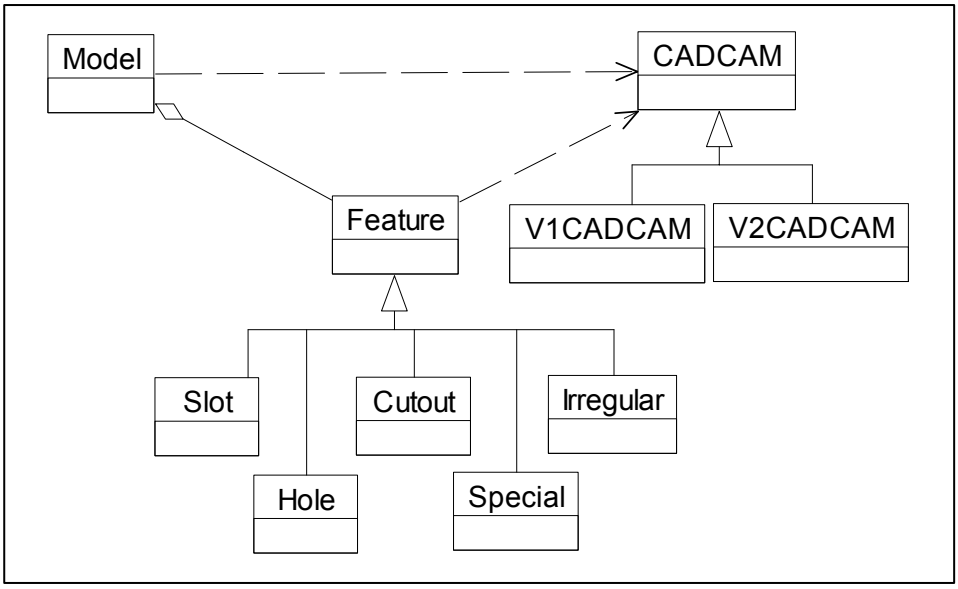

Figure 15-5 Handling variations in Models by using the CADCAM classes.

Expanding the design

I still need to expand the design to relate the **`CADCAM`** classes to the actual V1 and V2 implementations. Recalling what we know about the Facade and Adapter patterns, it should be clear that **`V1CADCAM`** should simply be a facade to the V1 system while **`V2CADCAM`** should wrap (adapt) the V2 system (**`OOG_Part`**). I show this in Figure 15-6.

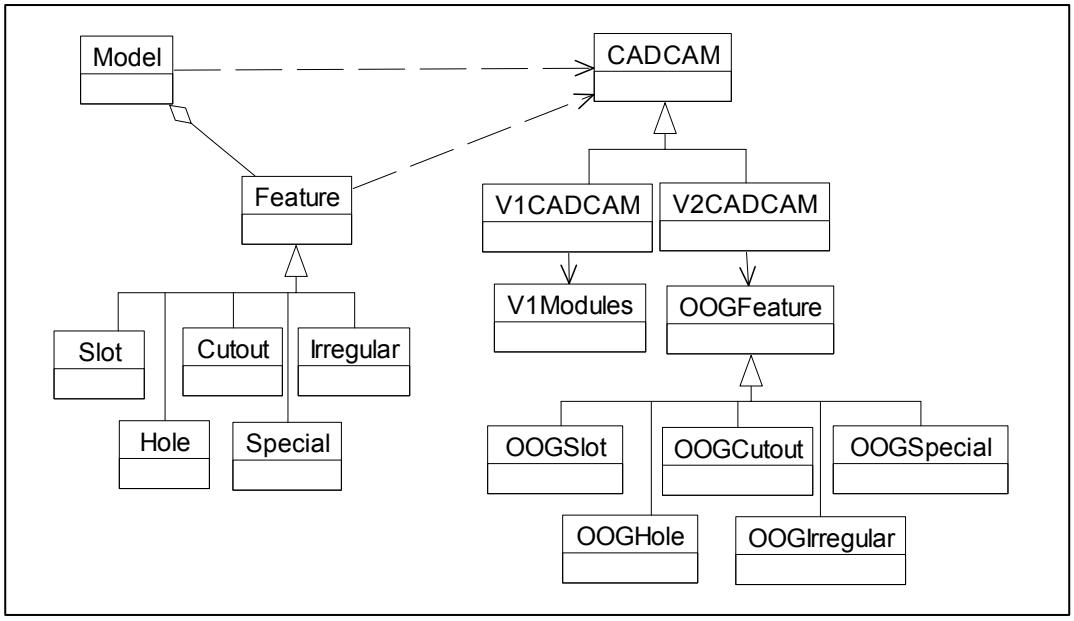

Figure 15-6 A completed design.

Contrast this with the solution arrived at using design patterns directly. This is shown in Figure 15-7.

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

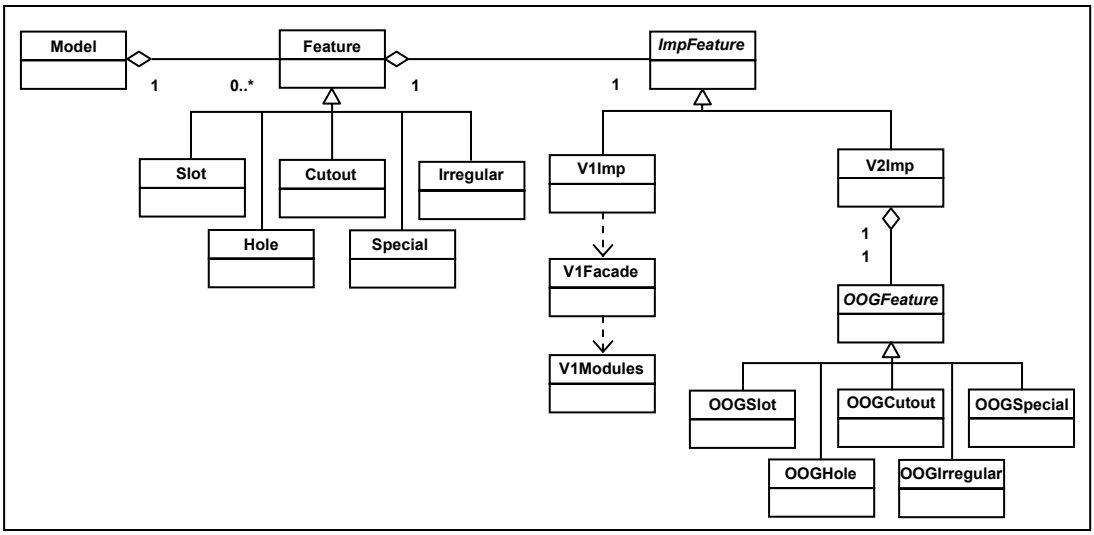

Figure 15-7 A solution using design patterns

The two look surprisingly similar. Or, maybe not. The solution arrived at using design patterns was derived by using patterns in a contextual way. I applied one pattern at a time until the solution unfolded. This is very similar to the CVA approach:

1. Identify commonalities first.
2. Create abstractions from these.
3. Identify derivations with the variations of the commonalities.
4. See how the commonalities relate to each other.

This is another type of design by context. The interfaces of the classes are defined within the context of how they are used by other abstractions. The class definitions are similar in both approaches because CVA is just another way of finding what varies and encapsulating it in cohesive, loosely coupled classes – principles upon which the design patterns are based. The same principles and approaches therefore lead to remarkably similar solutions.

In truth, the two approaches are highly synergistic. CVA says to focus on abstractions early, which increases the probability that I will find the most useful ones. Design patterns focus on the relationships between those abstractions, but do not help identify those abstractions in the first place.

On the other hand design patterns allow me to apply specific insights from past successful designs, whereas CVA does not. For example, I know it is often desirable to keep Facades

Technical Book Recommendation:

Waltzing With Bears: Managing Risk on Software Projects by Tom Demarco and Timothy Lister.

Demarco and Lister are two of my favorite authors and they hit another homer with this one. An important read for project managers and team leaders. Very insightful on the issue of risk and how to face reality. The book does not lean toward or away from agile – it's an essential read regardless of your development approach.

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Check www.netobjectives.com/events/pr_main.htm#FreeSeminars
for Public Seminars in
Western Washington State, Midwest, and Northern California

Seminars We Can Give

Transitioning to Agile – More and more companies are beginning to see the need for Agile Development. In this seminar, we discuss what problems agility will present and how to deal with these.

Test-First Techniques Using xUnit and Mock Objects – This seminar explains the basics of unit testing, how to use unit tests to drive coding forward (test-first), and how to resolve some of the dependencies that make unit testing difficult.

Pattern Oriented Development: Design Patterns From Analysis To Implementation – This seminar discusses how design patterns can be used to improve the entire software development process - not just the design aspect of it.

Agile Planning Game – The Planning Game was created by Kent Beck and is well described in his excellent book: Extreme Programming Explained. Unfortunately, the Planning Game as described is not complete enough - even for pure, XP teams. This seminar describes the other factors which must often typically be handled.

Comparing RUP, XP, and Scrum: Mixing a Process Cocktail for Your Team - This seminar discusses how combining the best of some popular processes can provide a successful software development environment for your project.

Design Patterns and Extreme Programming – Patterns are often thought of as an up-front design approach. That is not accurate. This seminar illustrates how the principles and strategies learned from patterns can actually facilitate agile development. This talk walks through a past project of the presenter.

Introduction to Use Cases – In this seminar we present different facets of the lowly Use Case; what they are, why they're important, how to write one, and how to support agile development with them.

Unit Testing For Emergent Design – This seminar illustrates why design patterns and refactoring are actually two sides of the same coin.

Check www.netobjectives.com/events/pr_main.htm#UpcomingPublicCourses
For a complete schedule of upcoming Public Courses
in Western Washington State, Midwest, and Northern California
and information on how to register

Volume 1, Issue 7
July 2004

NET OBJECTIVES

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

stateless due to the fact that they tend to be large, and so creating multiple instances can affect performance. Pulling the state needed by the features out of the Facade and placing it in the Adapter allows me to implement the Facade as a Singleton. CVA would not lead me to that conclusion.

Summary

In this chapter

I explained how CVA can be used to create high-level application designs. By defining our commonalities first, we eliminate coupling between our special cases. Because design patterns are really about isolating variations and commonality and variability analysis, used the way I described, does the same thing, we can get similar solutions with CVA that we get using design patterns. The advantage of the CVA approach, however, is that it can be used all the time. We can only design with patterns when we know the patterns involved. Something that my experience shows doesn't happen that often.

Review Questions

Observations

1. What are two approaches to identifying commonalities and variabilities?

Interpretations

1. CVA says you should have only one issue per commonality. Why is this important?

2. How do CVA and design patterns complement each other?

Opinions and Applications

1. "Experienced developers - even more than inexperienced ones - often focus on entity relationships too early, before they are clear what the right entities are". Is that your experience? Give an example to confirm or refute this statement.

2. Relate the approach to design starting with CVA with Alexander's approach.

To join a discussion about this article, please go to
http://www.netobjectivesgroups.com/6/ubb.x
and look under the E-zines category.

"I cannot believe that the purpose of life is to be "happy." I think the purpose of life is to be useful, to be responsible, to be honorable, to be compassionate. It is, above all, to matter: to count, to stand for something, to have made some difference that you lived at all.
– Leo Rosten

But being happy while you do all these is a good thing.
– Alan Shalloway

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Go to http://www.netobjectives.com/subscribe.htm to subscribe to our mailing list and receive future ezines and seminar announcements

Volume 1, Issue 7
July 2004

Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Net Objectives - What We Do

Net Objectives provides enterprises with a full selection of training, coaching and consulting services. Our Vision is "Effective software development without suffering". We facilitate software development organizations migration to more effective and efficient processes. In particular, we are specialists in agility, effective analysis, design patterns, refactoring and test-driven development.

We provide a blend of training, follow up coaching and staff supplementation that enables your team to become more effective in all areas of software development. Our engagements often begin with an assessment of where you are and detail a plan of how to become much more effective. Our trainers and consultants are experts in their fields (many of them published authors).

When you've taken a course from Net Objectives, you will see the world of software development with new clarity and new insight. Our graduates often tell us they are amazed at how we can simplify confusing and complicated subjects, and make them seem so understandable, and applicable for everyday use. Many of our students remark that it is the best training they have ever received.

The following courses are among our most often requested. This is not a complete list, though it is representative of the types of courses we offer

Agile Project Management - This 2-day course analyzes what it means to be an agile project, and provides a number of best practices that provide and/or enhance agility. Different agile practices from different processes (including RUP, XP and Scrum) are discussed.

Agile Use Case Analysis - This 3-day course provides theory and practice in deriving software requirements from use cases. This course is our recommendation for almost all organizations, and is intended for audiences that mix both business and software analysts.

Design Patterns Explained: A New Perspective on Object-Oriented Design - This 3-day course teaches several useful design patterns as a way to explain the principles and strategies that make design patterns effective. It also takes the lessons from design patterns and expands them into both a new way of doing analysis and a general way of building application architectures.

Test-Driven Development: Iterative Development, Refactoring and Unit Testing – This course teaches how to use either Junit, NUnit or CppUnit to create unit tests. Lab work is done in both Refactoring and Unit Testing together to develop very solid code by writing tests first. The course explains how the combination of Unit Testing Refactoring can result in emerging designs of high quality.

Effective Object-Oriented Analysis, Design and Programming In C++, C#, Java, or VB.net - This 5-day course goes beyond the basics of object-oriented design and presents 14 practices which will enable your developers to do object-oriented programming effectively. These practices are the culmination of the best coding practices of eXtreme Programming and of Design Patterns.

Software Dev Using an Agile (RUP, XP, SCRUM) Approach and Design Patterns - This 5-day course teaches several design patterns and the principles underneath them, the course goes further by showing how patterns can work together with agile development strategies to create robust, flexible, maintainable designs

If you are interested in any of these offerings, if your user group or company is interested in Net Objectives making a free technical presentation to them, or if you would like to be notified of upcoming Net Objectives events, please visit our website, or contact us by the email address or phone number below:

www.netobjectives.com • mike.shalloway@netobjectives.com • 404-593-8375