

# PART III

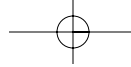
## Design Patterns

### Part Overview

This part introduces design patterns: what they are and how to use them. Several patterns pertinent to the CAD/CAM problem (Chapter 3, “A Problem That Cries Out for Flexible Code”) are described. They are presented individually and then related to the earlier problem. Throughout the pattern-learning chapters, I emphasize the object-oriented strategies espoused by the Gang of Four (as the authors Gamma, Helm, Johnson, and Vlissides are often referred to) in their seminal work, *Design Patterns: Elements of Reusable Object-Oriented Software*.

*In this part*

<b>Chapter</b>	<b>Discusses These Topics</b>
5	<b>An Introduction to Design Patterns</b> <ul style="list-style-type: none"><li>• The concept of design patterns, their origins in architecture, and how they apply in the discipline of software design.</li><li>• The motivations for studying design patterns.</li></ul>
6	<b>The Facade Pattern</b> <ul style="list-style-type: none"><li>• What it is, where it is used, and how it is implemented.</li><li>• How the Facade pattern relates to the CAD/CAM problem.</li></ul>
7	<b>The Adapter Pattern</b> <ul style="list-style-type: none"><li>• What it is, where it is used, and how it is implemented.</li><li>• Comparison between the Adapter pattern and the Facade pattern.</li></ul>

**74 Part III • Design Patterns**

- How the Adapter pattern relates to the CAD/CAM problem.
- 8      **Expanding Our Horizons**
- Some important concepts in object-oriented programming: polymorphism, abstraction, classes, and encapsulation. It uses what has been learned in Chapters 5 through 7.
- 9      **The Strategy Pattern**
- This pattern is the first exposure to patterns as a way of handling variation in a problem domain.
- 10     **The Bridge Pattern**
- This pattern is quite a bit more complex than the previous patterns. It is also much more useful; therefore, I go into great detail with the Bridge pattern.
  - How the Bridge pattern relates to the CAD/CAM problem.
- 11     **The Abstract Factory Pattern**
- This pattern focuses on creating families of objects.
  - What the pattern is, how it is used and implemented.
  - How the Abstract Factory pattern relates to the CAD/CAM problem.
- 

**Objectives**

At the end of this section, you will understand what design patterns are, know why they are useful, and be familiar with several specific patterns. You will also see how these patterns relate to the earlier CAD/CAM problem. This information, however, may not be enough to create a better design than we arrived at by over-relying on inheritance. However, the stage will be set for using patterns in a way more powerful than as just recurring solutions. This is different from the way many design pattern practitioners use them.

# CHAPTER 5

## An Introduction to Design Patterns

### Overview

This chapter introduces the concept of design patterns.

*In this chapter*

This chapter

- Discusses the origins of design patterns in architecture and how they apply in the discipline of software design.
- Examines the motivations for studying design patterns.

Design patterns are part of the cutting edge of object-oriented technology. Object-oriented analysis tools, books, and seminars are incorporating design patterns. Study groups on design patterns abound. It is often suggested that people learn design patterns only after they have mastered basic object-oriented skills. I have found that the opposite is true: Learning design patterns early in the learning of object-oriented skills greatly helps to improve understanding of object-oriented analysis and design.

*Design patterns and object-oriented design reinforce each other*

Throughout the rest of the book, I discuss not only design patterns, but also how they reveal and reinforce good object-oriented principles. I hope to improve both your understanding of these principles and illustrate why the design patterns being discussed here represent good designs.

Some of this material may seem abstract or philosophical. But give it a chance! This chapter lays the foundation for your understanding of design patterns. Understanding this material will enhance your ability to understand and work with new patterns.

*Give this a chance*

I have taken many of my ideas from Christopher Alexander's *The Timeless Way of Building*,<sup>1</sup> an excellent book that presents the philosophy of patterns succinctly. I discuss these ideas throughout this book.

## Design Patterns Arose from Architecture and Anthropology

*Is quality objective?*

Years ago, an architect named Christopher Alexander asked himself, "Is quality objective?" Is beauty truly in the eye of the beholder, or would people agree that some things are beautiful and some are not? The particular form of beauty that Alexander was interested in was one of architectural quality: What makes us know when an architectural design is good? For example, if a person were going to design an entryway for a house, how would he or she know that the design was good? Can we know good design? Is there an objective basis for such a judgment? A basis for describing our common consensus?

Alexander postulates that there is such an objective basis within architectural systems. The judgment that a building is beautiful is not simply a matter of taste. We can describe beauty through a measurable objective basis.

The discipline of cultural anthropology discovered the same thing. That body of work suggests that within a culture, individuals will agree to a large extent on what is considered to be a good design, what is beautiful. Cultures make judgments on good design that transcend individual beliefs. I believe that transcending patterns serve as objective bases for judging design. A major branch of cultural anthropology looks for such patterns to describe the behaviors and values of a culture.<sup>2</sup>

---

1. Alexander, C., *The Timeless Way of Building*, New York: Oxford University Press, 1979.

2. The anthropologist Ruth Benedict is a pioneer in pattern-based analysis of cultures. For examples, see Benedict, R., *The Chrysanthemum and the Sword*, Boston: Houghton Mifflin, 1946.

A proposition behind design patterns is that the quality of software systems can be measured objectively.

*How do you get good quality repeatedly?*

If you accept the idea that it is possible to recognize and describe a good quality design, how do you go about creating one? I can imagine Alexander asking himself,

*What is present in a good quality design that is not present in a poor quality design?*

and

*What is present in a poor quality design that is not present in a good quality design?*

These questions spring from Alexander's belief that if quality in design is objective, we should be able to identify what makes designs good and what makes designs bad.

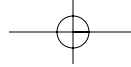
Alexander studied this problem by making many observations of buildings, towns, streets, and virtually every other aspect of living spaces that human beings have built for themselves. He discovered that, for a particular architectural creation, good constructs had things in common with each other.

*Look for the commonalities*

Architectural structures differ from each other, even if they are of the same type. Yet even though they differ, they can still be of high quality.

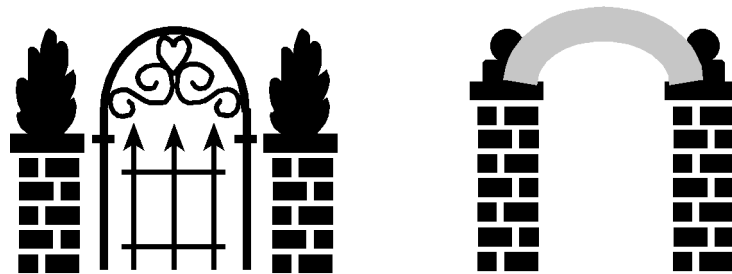
*...especially commonality in the features of the problem to be solved*

For example, two porches may appear structurally different and yet both may still be considered high quality. They might be solving different problems for different houses. One porch may be a transition from the walkway to the front door. Another porch might be a place for shade on a hot day. Or two porches might solve a common problem (transition) in different ways.



## 78 Part III • Design Patterns

Alexander understood this. He knew that structures couldn't be separated from the problem they are trying to solve. Therefore, in his quest to identify and describe the consistency of quality in design, Alexander realized that he had to look at different structures that were designed to solve the same problem. For example, Figure 5-1 illustrates two solutions to the problem of demarcating an entryway.



**Figure 5-1 Structures may look different but still solve a common problem.**

*This led to the concept of a pattern*

Alexander discovered that by narrowing his focus in this way—by looking at structures that solve similar problems—he could discern similarities between designs that were high quality. He called these similarities *patterns*.

He defined a pattern as “a solution to a problem in a context”:

*Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.<sup>3</sup>*

Let's review some of Alexander's work to illustrate this. Table 5-1 presents an excerpt from his *The Timeless Way of Building*.<sup>4</sup>

3. Alexander, C., Ishikawa, S., Silverstein, M., *A Pattern Language*, New York: Oxford University Press, 1977, p. x.

4. Alexander, C., *The Timeless Way of Building*, New York: Oxford University Press, 1979.

**Table 5-1 Excerpts from The Timeless Way of Building**

Alexander Says	My Comments
<p>In the same way, a courtyard, which is properly formed, helps people come to life in it.</p>	<p>A pattern always has a name and has a purpose. Here, the pattern's name is Courtyard, and its purpose is to help energize people, to help them feel alive.</p>
<p>Consider the forces at work in a courtyard. Most fundamental of all, people seek some kind of private outdoor space, where they can sit under the sky, see the stars, enjoy the sun, perhaps plant flowers. This is obvious.</p>	<p>Although it might be obvious sometimes, it is important to state explicitly the problem being solved, which is the reason for having the pattern in the first place. This is what Alexander does here for Courtyard.</p>
<p>But there are more subtle forces too. For instance, when a courtyard is too tightly enclosed, has no view out, people feel uncomfortable, and tend to stay away ...they need to see out into some larger and more distant space.</p>	<p>He points out a difficulty with the simplified solution and then gives us a way to solve the problem that he has just pointed out.</p>
<p>Or again, people are creatures of habit. If they pass in and out of the courtyard, every day, in the course of their normal lives, the courtyard becomes familiar, a natural place to go...and it is used.</p>	<p>Familiarity sometimes keeps us from seeing the obvious. The value of a pattern is that those with less experience can take advantage of what others have learned before them: both what must be included to have a good design, and what must be avoided to keep from a poor design.</p>
<p>But a courtyard with only one way in, a place you only go when you "want" to go there, is an unfamiliar place, tends to stay unused...people go more often to places which are familiar.</p>	<p>Although patterns are often thought of as theoretical constructs, in fact they reflect practical concerns that have arisen repeatedly in the past.</p>
<p>Or again, there is a certain abruptness about suddenly stepping out, from the inside, directly to the outside...it is subtle, but enough to inhibit you.</p>	<p>Alexander describes a force that is present (and important), but can be easily overlooked.</p>
<p>If there is a transitional space—a porch or a veranda, under cover, but open to the air—this is psychologically half way between indoors and outdoors, and makes it much</p>	<p>He proposes a solution to a possibly overlooked challenge to building a great courtyard. Also he shows that one pattern (here a courtyard) often provides the context for</p>

## 80 Part III • Design Patterns

Table 5-1 Excerpts from *The Timeless Way of Building* (cont.)

Alexander Says	My Comments
easier, more simple, to take each of the smaller steps that brings you out into the courtyard...	another pattern (here a porch or veranda). Patterns provide context for each other because recognizing a pattern helps clarify the problem itself, making other patterns easier to see.
When a courtyard has a view out to a larger space, has crossing paths from different rooms, and has a veranda or a porch, these forces can resolve themselves. The view out makes it comfortable, the crossing paths help generate a sense of habit there, the porch makes it easier to go out more often...and gradually the courtyard becomes a pleasant customary place to be.	Alexander is telling us how to build a great courtyard...and then tells us why it is great.

*The four components required of every pattern description*

To review, Alexander says that a description of a pattern involves four items:

- The name of the pattern
- The purpose of the pattern, the problem it solves
- How we could accomplish this
- The constraints and forces we have to consider in order to accomplish it

*Patterns exist for almost any design problem*

Alexander postulated that patterns can solve virtually every architectural problem that one will encounter. He further postulated that patterns could be used together to solve complex architectural problems.

*...and may be combined to solve complex problems*

How patterns work together is discussed later in this book. For now, I want to focus on his claim that patterns are useful to solve specialized problems.



## Moving from Architectural to Software Design Patterns

What does all of this architectural stuff have to do with us software developers?

Well, in the early 1990s, some smart developers happened upon Alexander's work in patterns. They wondered whether what was true for architectural patterns would also be true for software design.<sup>5</sup>

*Adapting Alexander  
for software*

- Were there problems in software that occur over and over again that could be solved in somewhat the same manner?
- Was it possible to design software in terms of patterns, creating specific solutions based on these patterns only after the patterns had been identified?

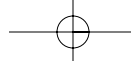
The group felt the answer to both of these questions was “unequivocally yes.” The next step was to identify several patterns and develop standards for cataloging new ones.

Although many people were working on design patterns in the early 1990s, the book that had the greatest influence on this fledging community was *Design Patterns: Elements of Reusable Object-Oriented Software*<sup>6</sup> by Gamma, Helm, Johnson, and Vlissides. In recognition of their important work, these four authors are commonly and affectionately known as the *Gang of Four*.

*The Gang of Four  
did the early work  
on design patterns*

This book served several purposes:

- 
5. The ESPRIT consortium in Europe was doing similar work in the 1980s. ESPRIT's Project 1098 and Project 5248 developed a pattern-based design methodology called *Knowledge Analysis and Design Support (KADS)* that was focused on patterns for creating expert systems. Karen Gardner extended the KADS analysis patterns to object orientation. See Gardner, K., *Cognitive Patterns: Problem-Solving Frameworks for Object Technology*, New York: Cambridge University Press, 1998.
  6. Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Boston: Addison-Wesley, 1995.



- It applied the idea of patterns to software design—calling them *design patterns*.
- It described a structure within which to catalog and describe design patterns.
- It cataloged 23 such patterns.
- It postulated object-oriented strategies and approaches based on these design patterns.

It is important to realize that the Gang of Four did not create the patterns described in the book. Rather, they identified patterns that already existed within the software community, patterns that reflected what had been learned about high-quality designs for specific problems. (Note the similarity to Alexander's work.)

Today there are several different forms for describing design patterns. Because this is not a book about writing design patterns, I do not offer an opinion on the best structure for describing patterns; however, the following items listed in Table 5-2 need to be included in any description.

For each pattern that I present in this book, I present a one-page summary of the key features that describes that pattern.

### **Consequences/Forces**

The term *consequences* is used in design patterns and is often misunderstood. In everyday usage, consequences usually carries a negative connotation. (You never hear someone say, "I won the lottery! As a *consequence*, I now do not have to go to work!") Within the design pattern community, on the other hand, consequences simply refers to cause and effect. That is, if you implement this pattern in such-and-such a way, this is how it will affect and be affected by the forces present.

**Table 5-2 Key Features of Patterns**

Item	Description
Name	All patterns have a unique name that identifies them.
Intent	The purpose of the pattern.
Problem	The problem that the pattern is trying to solve.
Solution	How the pattern provides a solution to the problem in the context in which it shows up.
Participants and collaborators	The entities involved in the pattern.
Consequences	The consequences of using the pattern. Investigates the forces at play in the pattern.
Implementation	How the pattern can be implemented.  Note: Implementations are just concrete manifestations of the pattern and should not be construed as the pattern itself.
Generic structure	A standard diagram that shows a typical structure for the pattern.

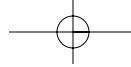
**Note:** Unless otherwise noted, the Key Features summary is extracted from the pattern description in the GoF book.

## Why Study Design Patterns?

Now that you have an idea about what design patterns are, you may still be wondering, “Why study them?” There are several reasons that are obvious and some that are not so obvious.

*Design patterns help with reuse and communication*

The most commonly stated reasons for studying patterns are because patterns enable us to



## 84 Part III • Design Patterns

- **Reuse solutions**—By reusing already established designs, I get a head start on my problems and avoid *gotchas*. I get the benefit of learning from the experience of others. I do not have to reinvent solutions for commonly recurring problems.
- **Establish common terminology**—Communication and teamwork require a common base of vocabulary and a common viewpoint of the problem. Design patterns provide a common point of reference during the analysis and design phase of a project.

*Design patterns give a higher perspective on analysis and design*

Patterns give you a higher-level perspective on the problem and on the process of design and object orientation. This frees you from the tyranny of dealing with the details too early.

By the end of this book, I hope you will agree that this is one of the greatest reasons to study design patterns. It will shift your mindset and make you a more powerful analyst.

To illustrate this advantage, I want to relate a conversation between two carpenters about how to build the drawers for some cabinets.<sup>7</sup>

*Example of the tyranny of details: Carpenters making a set of drawers*

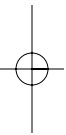
Consider two carpenters discussing how to build the drawers for some cabinets.

**Carpenter 1:** How do you think we should build these drawers?

**Carpenter 2:** Well, I think we should make the joint by cutting straight down into the wood, and then cut back up 45 degrees, and then going straight back down, and then back up the other way 45 degrees, and then going straight back down, and then...

---



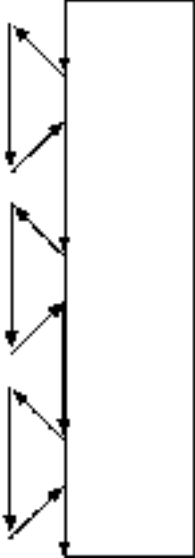
7. This section is inspired by a talk given by Ralph Johnson and is adapted by the authors.



Now, your job is to figure out what they are talking about!

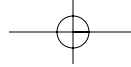
Isn't that a confusing description? What is Carpenter 2 prescribing? The details certainly get in the way! Let's try to draw out his description.

*The details may confuse the solution*

Carpenter Says...	Which Looks Like...
“Well, I think we should make the joint by cutting straight down into the wood, and then cut back up 45 degrees...”	
“...then going straight back down, and then back up the other way 45 degrees, and then going straight back down, and then...”	
“...until you end up with a <i>dovetail joint</i> . That is what I was describing!”	

Doesn't this sound like code reviews you have heard? The one where the programmer describes the code in terms such as

*This sounds like so many code reviews: Details, details, details*

**86 Part III • Design Patterns**

And then, I use a WHILE LOOP here to do...followed by a series of IF statements to do...and here I use a SWITCH to handle...

You get a description of the details of the code, but you have no idea *what the program is doing and why it is doing it!*

*Carpenters do not really talk at that level of detail*

Of course, no self-respecting carpenter would talk like this. What would really happen is something like this:

**Carpenter 1:** Should we use a dovetail joint or a miter joint?

Already we see a qualitative difference. The carpenters are discussing differences in the quality of solutions to a problem; their discussion is at a higher level, a more abstract level. They avoid getting bogged down in the details of a particular solution.

When the carpenter speaks of a miter joint, he or she has the following characteristics of the solution in mind:

- **It is a simpler solution**—A miter joint is a simple joint to make. You cut the edges of the joining pieces at 45 degrees, abut them, and then nail or glue them together, as shown in Figure 5-2.



**Figure 5-2** A miter joint.

- **It is lightweight**—A miter joint is weaker than a dovetail. It cannot hold together under great stress.
- **It is inconspicuous**—The miter joint's single cut is much less noticeable than the dovetail joint's multiple cuts.

When the carpenter speaks of a dovetail joint, he or she has other characteristics of the solution in mind. These characteristics may not be obvious to a layman, but would be clearly understood by any carpenter:

- **It is a more complex solution**—It is more involved to make a dovetail joint. Thus, it is more expensive.
- **It is impervious to temperature and humidity**—As these change, the wood expands or contracts. However, the dovetail joint will remain solid.
- **It is independent of the fastening system**—In fact, dovetail joints do not even depend upon glue to work.
- **It is a more aesthetically pleasing joint**—It is beautiful to look at when made well.

In other words, the dovetail joint is a strong, dependable, beautiful joint that is complex (and therefore expensive) to make.

*There is a meta-level conversation going on*

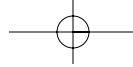
So, when Carpenter 1 asked

*Should we use a dovetail joint or a miter joint?*

The real question that was being asked was

*Should we use a joint that is expensive to make but is both beautiful and durable, or should we just make a quick and dirty joint that will last at least until the check clears?*

We might say the carpenters' discussion really occurs at two levels: the surface level of their words, and the *real* conversation, which



occurs at a higher level (a *meta-level*) that is hidden from the layman and which is much richer in meaning. This higher level is the level of “carpenter patterns” and reflects the real design issues for the carpenters.

In the first case, Carpenter 2 obscures the real issues by discussing the details of the implementations of the joints. In the second case, Carpenter 1 wants to decide which joint to use based on costs and quality of the joint.

Who is more efficient? Who would you rather work with?

*Patterns help us see the forest and the trees*

This is one thing I mean when I say that patterns can help raise the level of your thinking. You will learn later in the book that when you raise your level of thinking like this, new design methods become available. This is where the real power of patterns lies.

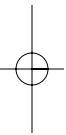
### **Other Advantages of Studying Design Patterns**

*Improve team communications and individual learning*

My experience with development groups working with design patterns is that design patterns helped both individual learning and team development. This occurred because the more junior team members saw that the senior developers who knew design patterns had something of value and these junior members wanted it. This provided motivation for them to learn some of these powerful concepts.

*Improved modifiability and maintainability of code*

Most design patterns also make software more modifiable and maintainable. The reason for this is that they are time-tested solutions. Therefore, they have evolved into structures that can handle change more readily than what often first comes to mind as a solution. They also handle this with easier to understand code—making it easier to maintain.





Design patterns, when they are taught properly, can be used to greatly increase the understanding of basic object-oriented design principles. I have seen this countless times in the introductory object-oriented courses I teach. In those classes, I start with a brief introduction to the object-oriented paradigm. I then proceed to teach design patterns, using them to illustrate the basic object-oriented concepts (encapsulation, inheritance, and polymorphism). By the end of the three-day course, although we've been talking mostly about patterns, these concepts—which were just introduced to many of the participants—feel as if they are old friends.

*Design patterns illustrate basic object-oriented principles*

The Gang of Four suggests a few strategies for creating good object-oriented designs. In particular, they suggest the following:

*Adoption of improved strategies—even when patterns aren't present*

- Design to interfaces.
- Favor aggregation over inheritance.<sup>8</sup>
- Find what varies and encapsulate it.

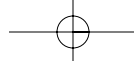
These strategies were employed in most of the design patterns discussed in this book. Even if you do not learn a lot of design patterns, studying a few should enable you to learn why these strategies are useful. With that understanding comes the ability to apply them to your own design problems even if you do not use design patterns directly.

Another advantage is that design patterns enable you or your team to create designs for complex problems that do not require large inheritance hierarchies. Again, even if design patterns are not used directly, avoiding large inheritance hierarchies will result in improved designs.

*Learn alternatives to large inheritance hierarchies*

---

8. Remember, *aggregation* means a collection of things that are not part of it (airplanes at an airport), whereas *composition* means something is a part of another thing (like wheels on an airplane). The distinction loses much of its importance in languages that have garbage collection, because you do not have to concern yourself with the life of the object. (When the object goes away, should its parts also go away?)



## Summary

*In this chapter*

This chapter described what design patterns are. Christopher Alexander says, “Patterns are solutions to a problem in a context.” They are more than a kind of template to solve one’s problems. They are a way of describing the motivations by including both what we want to have happen along with the problems that are plaguing us.

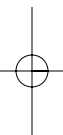
This chapter examined reasons for studying design patterns. Such study helps to

- Reuse existing, high-quality solutions to commonly recurring problems.
- Establish common terminology to improve communications within teams.
- Shift the level of thinking to a higher perspective.
- Decide whether I have the right design, not just one that works.
- Improve individual learning and team learning.
- Improve the modifiability of code.
- Facilitate adoption of improved design alternatives, even when patterns are not used explicitly.
- Discover alternatives to large inheritance hierarchies.

## Review Questions

### Observations

1. Who is credited with the idea for design patterns?
2. Alexander discovered that by looking at structures that solve similar problems, he could discern what?
3. Define *pattern*.



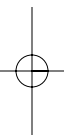
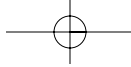
4. What are the key elements in the description of a design pattern?
5. What are three reasons for studying design patterns?
6. The Gang of Four suggests a few strategies for creating good object-oriented designs. What are they?

### Interpretations

1. “Familiarity sometimes keeps us from seeing the obvious.” In what ways can patterns help avoid this?
2. The Gang of Four cataloged 23 patterns. Where did these patterns come from?
3. What is the relationship between “consequence” and “forces” in a pattern?
4. What do you think “find what varies and encapsulate it” means?
5. Why is it desirable to avoid large inheritance hierarchies?

### Opinions and Applications

1. Think of a building or structure that felt particularly “dead.” What does it not have in common with similar structures that seem to be more “alive”?
2. “The real power of patterns is the ability to raise your level of thinking.” Have you had an experience in which this was true? Give an example.



**Business-Driven Software Development (BDSD)** is Net Objectives' proprietary integration of Lean-Thinking with Agile methods across the business, management and development teams to maximize the value delivered from a software development organization. BDSD has built a reputation and track record of delivering higher quality products faster and with lower cost than other methods

BDSD goes beyond the first generation of Agile methods such as Scrum and XP by viewing the entire value stream of development. Lean-Thinking enables product portfolio management, release planning and critical metrics to create a top-down vision while still promoting a bottom-up implementation.

BDSD integrates business, management and teams. Popular Agile methods, such as Scrum, tend to isolate teams from the business side and seem to have forgotten management's role altogether. These are critical aspects of all successful organizations. In BDSD:

- **Business** provides the vision and direction; properly selecting, sizing and prioritizing those products and enhancements that will maximize your investment
- **Teams** self-organize and do the work; consistently delivering value quickly while reducing the risk of developing what is not needed
- **Management** bridges the two; providing the right environment for successful development by creating an organizational structure that removes impediments to the production of value. This increases productivity, lowers cost and improves quality

## Become a Lean-Agile Enterprise

All levels of your organization will experience impacts and require change management. We help prepare executive, mid-management and the front-line with the competencies required to successfully change the culture to a Lean-Agile enterprise.

**Prioritization is only half the problem.** Learn how to both prioritize and size your initiatives to enable your teams to implement them quickly.

**Learn to come from business need not just system capability.** There is a disconnect between the business side and development side in many organizations. Learn how BDSD can bridge this gap by providing the practices for managing the flow of work.

## Why Net Objectives

While many organizations are having success with Agile methods, many more are not. Much of this is due to organizations either starting in the wrong place (e.g., the team when that is not the main problem) or using the wrong method (e.g., Scrum, just because it is popular). Net Objectives is experienced in all of the Agile team methods (Scrum, XP, Kanban, Scrumban) and integrates business, management and teams. This lets us help you select the right method for you.

<p><b>Assessments</b></p> <p>See where you are, where you want to go, and how to get there.</p> <p><b>Business and Management Training</b></p> <p>Lean Software Development Product Portfolio Management Enterprise Release Planning</p>	<p><b>Productive Lean-Agile Team Training</b></p> <p>Team training in Kanban, Scrum Technical Training in ATDD, TDD, Design Patterns</p> <p><b>Roles Training</b></p> <p>Lean-Agile Project Manager Product Owner</p>
--	---

