[PDF] Object-Oriented Design Heuristics

Arthur J. Riel - pdf download free book

Books Details:
Title: Object-Oriented Design Heuris
Author: Arthur J. Riel
Released: 1996-05-10
Language:
Pages: 400
ISBN: 020163385X
ISBN13: 978-0201633856
ASIN: 020163385X

---

pdf, mobi, epub, azw, kindle

Description:

From the Inside Flap In the process of teaching object-oriented analysis, design, and implementation to several thousand students, it became clear to me that the industry was in serious need of guidelines to help developers make proper decisions. Since 1987 I have scoured the literature in search of productivity and complexity metrics that can be applied at different levels of development to improve an object-oriented application. I added my own "homemade" guidelines to those found in the literature and came up with approximately 60 guidelines, several of which are tongue-in-cheek yet no less important than any others. I briefly considered calling them the "Sixty Golden Rules of OOA/D," but I recalled Dykstra's legendary "Goto Considered Harmful" paper, which branded users of goto statements heretics who should be burned at the stake in the company

courtyard. That paper was important in that it provided an industry rule that stopped the users of goto statements who were destroying, wittingly or unwittingly, the maintainability of their systems. Unfortunately, the side effect of such a rule was the breeding of a group of pathological authors who, for the past 25 years, have published articles stating that the judicious use of a goto statement in some picky little piece of an application is more readable than a corresponding piece of structured code. Of course, these papers were followed up by a half-dozen rebuttal papers, which were themselves rebutted ad nauseam.

In order to prevent the same pathology from occurring, I refer to these 60 guidelines as "heuristics," or rules of thumb. They are not hard and fast rules that must be followed under penalty of heresy. Instead, they should be thought of as a series of warning bells that will ring when violated. The warning should be examined, and if warranted, a change should be enacted to remove the violation of the heuristic. It is perfectly valid to state that the heuristic does not apply in a given example for one reason or another. In fact, in many cases, two heuristics will be at odds with one another in a particular area of an object-oriented design. The developer is required to decide which heuristic plays the more important role.

This book does not invent yet another object-oriented analysis or design methodology, though the idea of creating "Riel's OOA/D Methodology" was tempting. The industry already has enough methodologies offering similar or overlapping advice, using a completely different vocabulary for common concepts. The typical problem of the object-oriented developer - which has not been seriously addressed - occurs once a design has been completed, regardless of the methodology used. The developer's main question is, "Now that I have my design, is it good, bad, or somewhere in between?" In asking an object-oriented guru, the developer is often told that a design is good when "it feels right." While this is of little use to the developer, there is a kernel of truth to such an answer. The guru runs through a subconscious list of heuristics, built up through his or her design experience, over the design. If the heuristics pass, then the design feels right, and if they do not pass, then the design does not feel right.

This book attempts to capture that subconscious list of heuristics in a concrete list backed up by real-world examples. The reader will become immediately aware that some heuristics are much stronger than others. The strength of a heuristic comes from the ramifications of violating it. The reader does not get a prioritized ordering of the heuristics. It is my feeling that in many cases the sense of priority is defined by a combination of the application domain and the user's needs and cannot be quantified here. For example, a common area of design where two heuristics might request opposite directions are those that trade complexity with flexibility. Ask yourself which attribute a software designer desires most, increased flexibility or decreased complexity, and you begin to see the problem of prioritizing heuristics.

The design heuristics are defined on a backdrop of real-world examples focusing on the area of design to which each heuristic belongs. The foundation of real-world examples provides an ideal vehicle for explaining the concepts of object-oriented technology to the novice. The end result is that this book is appropriate to the newcomer who would like a fast track to understanding the concepts of object-oriented programming without having to muddle through the proliferation of buzzwords that permeates the field. Yet, at the same time, it appeals to the experienced object-oriented developer who is looking for some good analysis and design heuristics to help in his or her development efforts.

The first chapter looks at the motivation for object-oriented programming, starting with several issues which Frederick Brooks argued in his "No Silver Bullet" paper published in 1987 (see reference 1). My perspective on object-oriented programming is that it is a natural progression or evolution from action-oriented development. As software has become more complex, we are required

to remove ourselves one more level away from the machine in order to maintain the same grasp we have on the software development process. Just as structured methodologies removed one level from bottom-up programming, object-oriented technology removes one level from structured methodologies. It is not that bottom-up programming or structured methodologies are wrong and object-oriented programming is right. Bottom-up programming is perfectly valid when there exists only 4K of memory to develop, just as structured methodologies are perfectly valid when only 256K of memory exists. With the advent of increasingly cheaper and more powerful hardware, the complexity of software has skyrocketed. Developers of the early 1980s did not have to consider the complexity of graphical user interfaces and multithreaded applications; simpler menu-driven, single-threaded systems were the norm. In the very near future, no one will buy a software product unless it incorporates multimedia with moving video and voice recognition. The more complex systems require a greater level of abstraction, which the object-oriented paradigm provides. This is no revolution in software development; it is simply an evolution.

Chapter 2 discusses the concepts of class and object, the basic building blocks of object-oriented technology. They are viewed as the encapsulation of data and its related behavior in a bidirectional relationship. The notion of sending messages, defining methods, and inventing protocols are explored through real-world examples. This is the first chapter to list heuristics. Given the small subset of the object paradigm with which to work, these heuristics are fairly simple but no less useful than the more complex heuristics of subsequent chapters.

The third chapter examines the difference between an action-oriented topology and an object-oriented topology. The different topologies of these methodologies contain the kernel of truth behind object-oriented development. Action-oriented development focuses largely on a centralized control mechanism controlling a functionally decomposed set of tasks, while object-oriented development focuses on a decentralized collection of cooperating entities. I am convinced that the notion of a paradigm shift is the change in thinking required to move from a centralized to a decentralized control model. The learning curve of object-oriented development is an equally large unlearning curve for those of us reared in the world of action-oriented development. The real world in which we live is more attuned to the object model than to a centralized control mechanism. The lack of a paradigm shift manifests itself in systems that consist of a central godlike object that sits in the middle of a collection of trivial classes. These systems are built by developers stuck in the mindset of an action-oriented topology. This chapter proposes numerous heuristics for developing optimal application topologies.

Chapters 4 through 7 examine each of the five main object-oriented relationships: uses (Chapter 4); containment (Chapter 4); single inheritance (Chapter 5); multiple inheritance (Chapter 6); and association (Chapter 7) through a series of real-world examples. Most of the heuristics of interest to the object-oriented designer can be found in these chapters. The chapters on inheritance include many examples of the common misuses of the inheritance relationship. This information is vital in reducing the proliferation of classes problem, such as designing too many classes for a given application. The class proliferation problem is a major cause of failure in object-oriented development.

Chapter 8 examines the role of class-specific data and behavior, as opposed to object-specific data and behavior. The invoice class is used as an example of an abstraction that requires class-specific data and behavior. Both the SmallTalk metaclass and the C++ keyword mechanisms are illustrated. In addition, the notion of C++ metaclasses (i.e., templates) is compared and contrasted to the SmallTalk notion of metac

From the Back Cover

Upon completion of an object-oriented design, you are faced with a troubling question: "Is it good, bad, or somewhere in between?" Seasoned experts often answer this question by subjecting the design to a subconscious list of guidelines based on their years of experience. Experienced developer Arthur J. Riel has captured this elusive, subconscious list, and in doing so, has provided a set of metrics that help determine the quality of object-oriented models.

Object-Oriented Design Heuristics offers insight into object-oriented design improvement. The more than sixty guidelines presented in this book are language-independent and allow you to rate the integrity of a software design. The heuristics are not written as hard and fast rules; they are meant to serve as warning mechanisms which allow the flexibility of ignoring the heuristic as necessary. This tutorial-based approach, born out of the author's extensive experience developing software, teaching thousands of students, and critiquing designs in a variety of domains, allows you to apply the guidelines in a personalized manner.

The heuristics cover important topics ranging from classes and objects (with emphasis on their relationships including association, uses, containment, and both single and multiple inheritance) to physical object-oriented design. You will gain an understanding of the synergy that exists between design heuristics and the popular concept of design patterns; heuristics can highlight a problem in one facet of a design while patterns can provide the solution.

Programmers of all levels will find value in this book. The newcomer will discover a fast track to understanding the concepts of object-oriented programming. At the same time, experienced programmers seeking to strengthen their object-oriented development efforts will appreciate the insightful analysis. In short, with **Object-Oriented Design Heuristics** as your guide, you have the tools to become a better software developer.

020163385XB04062001

- Title: Object-Oriented Design Heuristics
- Author: Arthur J. Riel
- Released: 1996-05-10
- Language:
- Pages: 400
- ISBN: 020163385X
- ISBN13: 978-0201633856
- ASIN: 020163385X
