Read PDF Chapter 7 Object Oriented Software Engineering Addressing

If you ally habit such a referred **Chapter 7 Object Oriented Software Engineering Addressing** books that will pay for you worth, get the unquestionably best seller from us currently from several preferred authors. If you want to comical books, lots of novels, tale, jokes, and more fictions collections are also launched, from best seller to one of the most current released.

You may not be perplexed to enjoy every book collections Chapter 7 Object Oriented Software Engineering Addressing that we will extremely offer. It is not regarding the costs. Its nearly what you compulsion currently. This Chapter 7 Object Oriented Software Engineering Addressing, as one of the most practicing sellers here will unquestionably be along with the best options to review.

SWXB6M - LILLY COLLINS

This book is intended for anyone who plans, designs and implements software systems, for anyone who is involved with quality assurance, and hence for anyone who is interested in the practicability of modern concepts, methods and tools in the software development process. The book aims at software engineers and at students with specialized interests in the area of software engineering. The reader is expected to be familiar with the fundamental concepts of software engineering. In writing the book, the authors tap years of experience in industrial projects and research work in the development of methods and tools that support the software development process. Perhaps now more than ever, the buzzword "software crisis" serves to alert us that software systems are often error-prone, that significant difficulties arise in mastering complexity in the production of software systems, and that the acceptance and adequacy of software products is significantly lower than is the case with other technical products. The following goals have been suggested for the improvement of the software development process: • exact fulfillment of user requirements • increased reliability and robustness • greater modularity of both the development process and the product • simple and adequate operation, i. e. , better ergonomics • easy maintainability and extensibility • cost-effective portability • increased reusability of software components • reduced costs for production, operation and maintenance VI Preface Research and development work in the area of software engineering has in creased dramatically in recent years.

Classical and Object-Oriented Software Engineering, 5/e is designed for an introductory software engineering course. This book provides an excellent introduction to

software engineering fundamentals, covering both traditional and object-oriented techniques.Schach's unique organization and style makes it excellent for use in a classroom setting. It presents the underlying software engineering theory in Part I and follows it up with the more practical life-cycle material in Part II. Many software engineering books are more like reference books, which do not provide the appropriate fundamentals before inundating students with implementation details.In this edition, more practical material has been added to help students understand how to use what they are learning. This has been done through the use of "How To" boxes and greater implementation detail in the case study. Additionally, the new edition contains the references to the most current literature and includes an overview of extreme programmming.The website in this edition will be more extensive. It will include Solutions, PowerPoints that incorporate lecture notes, newly developed self-quiz questions, and source code for the term project and case study.

eBook: Object-Oriented Systems Analysis 4e

This 1998 book presents the underlying principles associated with object-orientation and its practical application.

For courses in Software Engineering, Software Development, or Object-Oriented Design and Analysis at the Junior/Senior or Graduate level. This text can also be utilized in short technical courses or in short, intensive management courses. Shows students how to use both the principles of software engineering and the practices of various object-oriented tools, processes, and products. Using a step-by-step case study to illustrate the concepts and topics in each chapter, Bruegge and Dutoit emphasize learning object-oriented software engineer through practical experience: students can apply the techniques learned in class by implementing a real-world software project. The third edition addresses new trends, in particular agile project management (Chapter 14 Project Management) and agile methodologies (Chapter 16 Methodologies).

EBOOK: OBJECT-ORIENTED SOFTWAR

This textbook provides a progressive approach to the teaching of software engineering. First, readers are introduced to the core concepts of the object-oriented methodology, which is used throughout the book to act as the foundation for software engineering and programming practices, and partly for the software engineering process itself. Then, the processes involved in software engineering are explained in more detail, especially methods and their applications in design, implementation, testing, and measurement, as they relate to software engineering projects. At last, readers are given the chance to practice these concepts by applying commonly used skills and tasks to a hands-on project. The impact of such a format is the potential for quicker and deeper understanding. Readers will master concepts and skills at the most basic levels before continuing to expand on and apply these lessons in later chapters.

Fundamentals of Object-Oriented Design in UML shows aspiring and experienced programmers alike how to apply design concepts, the UML, and the best practices in OO development to improve both their code and their success rates with object-based projects.

This book demonstrates the efficiency of the C++ programming language in the

realm of pattern recognition and pattern analysis. For this 4th edition, new features of the C++ language were integrated and their relevance for image and speech processing is discussed.

This book is written for engineering students and working professionals. Technical professionals are increasingly involved in IT issues, such as implementing IT systems, managing them, and taking part in requirements analysis/vendor selection. In this book, the basics of production planning systems (PPS) are covered, as well as their implementation in ERP-Systems like SAP. Readers also learn the basics of practical IT management and software creation through detailed, real-world examples. The book serves as a full 5 ECTS study module, which fits into any engineering curriculum. 150 multiple-choice quizzes, practical exercises and a text filled with experiential examples make it a convenient choice for selfstudy and for classroom use.

I must confess that I stumbled upon the object-oriented (00) world view during my explorations into the world of artificial intelligence (AI) in search of a new solution to the problem of building computer-integrated manufacturing systems (CIM). In 00 computing, I found the constructs to model the manufacturing enterprise in terms of information, a resource that is common to all activities in an organization. It offered a level of modularity, and the coupling/binding neces sary for fostering integration without placing undue restrictions on what the individual applications can do. The implications of 00 computing are more extensive than just being a vehicle for manufacturing applications. Leaders in the field such as Brad Cox see it introducing a paradigm shift that will change our world gradually, but as radically as the Industrial Revolution changed manufacturing. However, it must be borne in mind that simply using an object-oriented language or environment does not, in itself, ensure success in one's applications. It requires a different way of thinking, design discipline, techniques, and tools to exploit what the technology has to offer. In other words, it calls for a paradigm shift (as defined by Kuhn in The Structure of Scientific Revolution, a classic text in the history of science).

Object-oriented programming is a popular buzzword these days. What is the reason for this popularity? Is object-oriented programming the solution to the software crisis or is it just a fad? Is it a simple evolutionary step or a radical change in software methodology? What is the central idea behind object-orien ted design? Are there special applications for which object-oriented program ming is particularly suited? Which object-oriented language should be used? There is no simple answer to these questions. Although object-oriented programming was invented more than twenty years ago, we still cannot claim that we know everything about this programming technique. Many new con cepts have been developed during the past decade, and new applications and implications of object-oriented programming are constantly being discovered. This book can only try to explain the nature of object-oriented program ming in as much detail as possible. It should serve three purposes. First, it is intended as an introduction to the basic concepts of object-oriented program ming. Second, the book describes the concept of prototypes and explains why and how they can improve the way in which object-oriented programs are developed. Third, it introduces the programming language Ome-ga, an object oriented language that was designed with easy, safe and efficient soft-

ware development in mind.

This comprehensive and well-written book presents the fundamentals of object-oriented software engineering and discusses the recent technological developments in the field. It focuses on object-oriented software engineering in the context of an overall effort to present object-oriented concepts, techniques and models that can be applied in software estimation, analysis, design, testing and quality improvement. It applies unified modelling language notations to a series of examples with a real-life case study. The example-oriented approach followed in this book will help the readers in understanding and applying the concepts of object-oriented software engineering quickly and easily in various application domains. This book is designed for the undergraduate and postgraduate students of computer science and engineering, computer applications, and information technology. KEY FEATURES : Provides the foundation and important concepts of object-oriented paradigm. Presents traditional and object-oriented software development life cycle models with a special focus on Rational Unified Process

model. Addresses important issues of improving software quality and measuring various object-oriented constructs using object-oriented metrics. Presents numerous diagrams to illustrate object-oriented software engineering models and concepts. Includes a large number of solved examples, chapter-end review questions and multiple choice questions along with their answers.

Nothing provided

Addressing general readers as well as software practitioners, "Software and Mind" discusses the fallacies of the mechanistic ideology and the degradation of minds caused by these fallacies. Mechanism holds that every aspect of the world can be represented as a simple hierarchical structure of entities. But, while useful in fields like mathematics and manufacturing, this idea is generally worthless, because most aspects of the world are too complex to be reduced to simple hierarchical structures. Our software-related affairs, in particular, cannot be represented in this fashion. And yet, all programming theories and development systems, and all software applications, attempt to reduce real--world problems to neat hierarchical struc-

tures of data, operations, and features. Using Karl Popper's famous principles of demarcation between science and pseudoscience, the book shows that the mechanistic ideology has turned most of our software-related activities into pseudoscientific pursuits. Using mechanism as warrant, the software elites are promoting invalid, even fraudulent, software notions. They force us to depend on generic, inferior systems, instead of allowing us to develop software skills and to create our own systems. Software mechanism emulates the methods of manufacturing, and thereby restricts us to high levels of abstraction and simple, isolated structures. The benefits of software, however, can be attained only if we start with low-level elements and learn to create complex, interacting structures. Software, the book argues, is a non-mechanistic phenomenon. So it is akin to language, not to physical objects. Like language, it permits us to mirror the world in our minds and to communicate with it. Moreover, we increasingly depend on software in everything we do, in the same way that we depend on language. Thus, being restricted to mechanistic software is like thinking and communi-

cating while being restricted to some ready-made sentences supplied by an elite. Ultimately, by impoverishing software, our elites are achieving what the totalitarian elite described by George Orwell in "Nineteen Eighty-Four" achieves by impoverishing language: they are degrading our minds.

Covers advanced features of Perl, how the Perl interpreter works, and presents areas of modern computing technology such as networking, user interfaces, persistence, and code generation.

This is a textbook for a course in object-oriented software engineering at advanced undergraduate and graduate levels, as well as for software engineers. It contains more than 120 exercises of diverse complexity.The book discusses fundamental concepts and terminology on object-oriented software development, assuming little background on software engineering, and emphasizes design and maintenance rather than programming.It also presents up-to-date and easily understood methodologies and puts forward a software life cycle model which explicitly encourages reusability during software development and maintenance.

Software architectures have gained wide popularity in the last decade. They generally play a fundamental role in coping with the inherent difficulties of the development of large-scale and complex software systems. Component-oriented and aspect-oriented programming enables software engineers to implement complex applications from a set of pre-defined components. Software Architectures and Component Technology collects excellent chapters on software architectures and component technologies from well-known authors, who not only explain the advantages, but also present the shortcomings of the current approaches while introducing novel solutions to overcome the shortcomings. The unique features of this book are: evaluates the current architecture design methods and component composition techniques and explains their shortcomings; presents three practical architecture design methods in detail; gives four industrial architecture design examples; presents conceptual models for distributed message-based architectures; explains techniques for refining architectures into components; presents the recent develop-

ments in component and aspect-oriented techniques; explains the status of research on Piccola, Hyper/J®, Pluggable Composite Adapters and Composition Filters. Software Architectures and Component Technology is a suitable text for graduate level students in computer science and engineering, and as a reference for researchers and practitioners in industry.

Software Engineering: A Methodical Approach (Second Edition) provides a comprehensive, but concise introduction to software engineering. It adopts a methodical approach to solving software engineering problems, proven over several years of teaching, with outstanding results. The book covers concepts, principles, design, construction, implementation, and management issues of software engineering. Each chapter is organized systematically into brief, reader-friendly sections, with itemization of the important points to be remembered. Diagrams and illustrations also sum up the salient points to enhance learning. Additionally, the book includes the author's original methodologies that add clarity and creativity to the software engineering experience. New in the Second Edition are chapters on software engi-

neering projects, management support systems, software engineering frameworks and patterns as a significant building block for the design and construction of contemporary software systems, and emerging software engineering frontiers. The text starts with an introduction of software engineering and the role of the software engineer. The following chapters examine in-depth software analysis, design, development, implementation, and management. Covering object-oriented methodologies and the principles of object-oriented information engineering, the book reinforces an object-oriented approach to the early phases of the software development life cycle. It covers various diagramming techniques and emphasizes object classification and object behavior. The text features comprehensive treatments of: Project management aids that are commonly used in software engineering An overview of the software design phase, including a discussion of the software design process, design strategies, architectural design, interface design, database design, and design and development standards User interface design Operations design Design considerations including system catalog, product documentation, user message management, design for real-time software, design for reuse, system security, and the agile effect Human resource management from a software engineering perspective Software economics Software implementation issues that range from operating environments to the marketing of software Software maintenance, legacy systems, and re-engineering This textbook can be used as a one-semester or two-semester course in software engineering, augmented with an appropriate CASE or RAD tool. It emphasizes a practical, methodical approach to software engineering, avoiding an overkill of theoretical calculations where possible. The primary objective is to help students gain a solid grasp of the activities in the software development life cycle to be confident about taking on new software engineering projects.

Software Development with C++: Maximizing Reuse with Object Technology is about software development and object-oriented technology (OT), with applications implemented in C++. The basis for any software development project of complex systems is the process, rather than an individual method, which simply supports the overall process. This book is not intended as a general, all-encompassing treatise on OT. The intent is to provide practical information that is directly applicable to a development project. Explicit guidelines are offered for the infusion of OT into the various development phases. The book is divided into five major parts. Part I describes why we need a development process, the phases and steps of the software process, and how we use individual methods to support this process. Part II lays the foundation for the concepts included in OT. Part III describes how OT is used in the various phases of the software development process, including the domain analysis, system requirements analysis, system design, software requirements analysis, software design, and implementation. Part IV deals exclusively with design issues for an anticipated C++ implementation. Part V is devoted to object-oriented programming with C++. This book is intended for practicing software developers, software managers, and computer science and software engineering students. Sufficient guidelines are included to aid project leaders in establishing an overall development process for small, medium, and large system applica-

tions.

Business Component-Based Software Engineering, an edited volume, aims to complement some other reputable books on CBSE, by stressing how components are built for large-scale applications, within dedicated development processes and for easy and direct combination. This book will emphasize these three facets and will offer a complete overview of some recent progresses. Projects and works explained herein will prompt graduate students, academics, software engineers, project managers and developers to adopt and to apply new component development methods gained from and validated by the authors. The authors of Business Component-Based Software Engineering are academic and professionals, experts in the field, who will introduce the state of the art on CBSE from their shared experience by working on the same projects. Business Component-Based Software Engineering is designed to meet the needs of practitioners and researchers in industry, and graduate-level students in Computer Science and Engineering.

Addressing various aspects of object-oriented software techniques with respect to their impact on testing, this text argues that the testing of object-oriented software is not restricted to a single phase of software development. The book concentrates heavily on the testing of classes and of components or sub-systems, and a major part is devoted to this subject. C++ is used throughout this book that is intended for software practitioners, managers, researchers, students, or anyone interested in object-oriented technology and its impacts throughout the software engineering life-cycle.

An industry insider explains why there is so much bad software—and why academia doesn't teach programmers what industry wants them to know. Why is software so prone to bugs? So vulnerable to viruses? Why are software products so often delayed, or even canceled? Is software development really hard, or are software developers just not that good at it? In The Problem with Software, Adam Barr examines the proliferation of bad software, explains what causes it, and offers some suggestions on how to improve the situation. For one thing, Barr points out, academia doesn't teach programmers what they actually need to know to do their jobs: how to work in a team to create code that works reliably and can be maintained by somebody other than the original authors. As the size and complexity of commercial software have grown, the gap between academic computer science and industry has widened. It's an open secret that there is little engineering in software engineering, which continues to rely not on codified scientific knowledge but on intuition and experience. Barr, who worked as a programmer for more than twenty years, describes how the industry has evolved, from the era of mainframes and Fortran to today's embrace of the cloud. He explains bugs and why software has so many of them, and why today's interconnected computers offer fertile ground for viruses and worms. The difference between good and bad software can be a single line of code, and Barr includes code to illustrate the consequences of seemingly inconsequential choices by programmers. Looking to the future, Barr writes that the best prospect for improving software engineering is the move to the cloud. When software is a service and not a product, companies will have more incentive to

make it good rather than "good enough to ship."

While Java texts are plentiful, it's difficult to find one that takes a real-world approach, and encourages novice programmers to build on their Java skills through practical exercise. Written by an expert with 19 experience teaching computer programming, Java Programming Fundamentals presents object-oriented programming by employing examples taken

This ground-breaking book presents a complete methodology for adaptive programming in any object-oriented programming language. Lieberherr's adaptive method signals a new approach to object-oriented program design that goes beyond object encapsulation and hard-coded navigation paths to achieve more flexible interactions among objects. Programmers using this method work at a higher, schematic level of abstraction; graph notation represents the class structure and a "propagation pattern" language tells how to distribute meaningful methods - including navigation - across the structure. Using this method, programmers can easily adapt and modify programs as they evolve. This book can be

used with any object-oriented programming environment, or with the Demeter Tools Version 5.5, a complete, professional software system for creating and maintaining adaptive programs.

David A. Sykes is a member of Wofford College's faculty.

The purpose of this book is to make the reader famliar with software engineering for distributed systems. Software engineering is a valuable discipline in the develop ment of software. The reader has surely heard of software systems completed months or years later than scheduled with huge cost overruns, systems which on completion did not provide the performance promised, and systems so catastrophic that they had to be abandoned without ever doing any useful work. Software engi neering is the discipline of creating and maintaining software; when used in con junction with more general methods for effective management its use does reduce the incidence of horrors mentioned above. The book gives a good impression of software engineering particularly for dis tributed systems. It emphasises the relationship between software life cycles, meth ods, tools and project management,

and how these constitute the framework of an open software engineering environment, especially in the development of distrib uted software systems. There is no closed software engineering environment which can encompass the full range of software missions, just as no single flight plan, airplane or pilot can perform all aviation missions. There are some common activities in software engi neering which must be addressed independent of the applied life cycle or methodol ogy. Different life cycles, methods, related tools and project management ap proaches should fit in such a software engineering framework.

This book provides the software engineering fundamentals, principles and skills needed to develop and maintain high quality software products. It covers requirements specification, design, implementation, testing and management of software projects. It is aligned with the SWEBOK, Software Engineering Undergraduate Curriculum Guidelines and ACM Joint Task Force Curricula on Computing.

The ultimate goal of program verification is not the theory behind the tools or the tools themselves, but the application of

the theory and tools in the software engineering process. Our society relies on the correctness of a vast and growing amount of software. Improving the software engineering process is an important, long-term goal with many steps. Two of those steps are the KeY tool and this KeY book.

Object-Oriented Design with Applications has long been the essential reference to object-oriented technology, which, in turn, has evolved to join the mainstream of industrial-strength software development. In this third edition--the first revision in 13 years--readers can learn to apply object-oriented methods using new paradigms such as Java, the Unified Modeling Language (UML) 2.0, and .NET. The authors draw upon their rich and varied experience to offer improved methods for object development and numerous examples that tackle the complex problems faced by software engineers, including systems architecture, data acquisition, cryptoanalysis, control systems, and Web development. They illustrate essential concepts, explain the method, and show successful applications in a variety of fields. You'll also find pragmatic advice on a host of issues, including classification, implementation strategies, and cost-effective project management. New to this new edition are An introduction to the new UML 2.0, from the notation's most fundamental and advanced elements with an emphasis on key changes New domains and contexts A greatly enhanced focus on modeling--as eagerly requested by readers--with five chapters that each delve into one phase of the overall development lifecycle. Fresh approaches to reasoning about complex systems An examination of the conceptual foundation of the widely misunderstood fundamental elements of the object model, such as abstraction, encapsulation, modularity, and hierarchy How to allocate the resources of a team of developers and mange the risks associated with developing complex software systems An appendix on object-oriented programming languages This is the seminal text for anyone who wishes to use object-oriented technology to manage the complexity inherent in many kinds of systems. Sidebars Preface Acknowledgments About the Authors Section I: Concepts Chapter 1: Complexity Chapter 2: The Object Model Chapter 3: Classes and Objects Chapter 4: Classification Section II: Method Chapter 5: Notation Chapter 6: Process Chapter 7: Pragmatics Chapter 8: System Architecture: Satellite-Based Navigation Chapter 9: Control System: Traffic Management Chapter 10: Artificial Intelligence: Cryptanalysis Chapter 11: Data Acquisition: Weather Monitoring Station Chapter 12: Web Application: Vacation Tracking System Appendix A: Object-Oriented Programming Languages Appendix B: Further Reading Notes Glossary Classified Bibliography Index

Test-Driven Development (TDD) is now an established technique for delivering better software faster. TDD is based on a simple idea: Write tests for your code before you write the code itself. However, this "simple" idea takes skill and judgment to do well. Now there's a practical guide to TDD that takes you beyond the basic concepts. Drawing on a decade of experience building real-world systems, two TDD pioneers show how to let tests guide your development and "grow" software that is coherent, reliable, and maintainable. Steve Freeman and Nat Pryce describe the processes they use, the design principles they strive to achieve, and some of the tools that help them get the job done. Through an extend-

ed worked example, you'll learn how TDD works at multiple levels, using tests to drive the features and the object-oriented structure of the code, and using Mock Objects to discover and then describe relationships between objects. Along the way, the book systematically addresses challenges that development teams encounter with TDD—from integrating TDD into your processes to testing your most difficult features. Coverage includes Implementing TDD effectively: getting started, and maintaining your momentum throughout the project Creating cleaner, more expressive, more sustainable code Using tests to stay relentlessly focused on sustaining quality Understanding how TDD, Mock Objects, and Object-Oriented Design come together in the context of a real software development project Using Mock Objects to guide object-oriented designs Succeeding where TDD is difficult: managing complex test data, and testing persistence and concurrency

The Pernambuco School on Software Engineering (PSSE) 2007 was the second in a series of events devoted to the study of advanced computer science and to the promotion of international scienti?c collabora-

tion. The main theme in 2007 was testing. Testing is nowadays a key activity for assuring software quality. The summer school and its proceedings were intended to give a detailed tutorial introduction to the scienti?c basis of this activity and its state of the art. Theseproceedingsrecordthecontributionsfromtheinvitedlecturers.Eachof thechaptersistheresultofathoroughrevisionoftheinitialnotesprovidedtothe participants of the school. The revision was inspired by the synergy generated by the opportunity for the lecturers to present and discuss their work among themselves and with the school's attendees. The editors have tried to produce a coherent view of the topic by harmonizing these contributions, smoothing out di?erences in notation and approach, and providing links between the lectures. We apologize to the authors for any errors introduced by our extensive editing. Although the chapters are linked in severalways, each one is su?ciently se- contained to be read in isolation. Nevertheless, Chap. 1 should be read ?rst by those interested in an introduction to testing. Chapter 1 introduces the terminology adopted in this book. It also provides an overview of the testing process, and of

the types (functional, structural, and so on)anddimensions(unit,integration,and-soon)ofthetestingactivity.Themain strategies employed in the central activity of test selection are also discussed. Most of the material presented in this introductory chapter is addressedin more depth in the following chapters.

In today's modernized environment, a growing number of software companies are changing their traditional engineering approaches in response to the rapid development of computing technologies. As these businesses adopt modern software engineering practices, they face various challenges including the integration of current methodologies and contemporary design models and the refactoring of existing systems using advanced approaches. Applications and Approaches to Object-Oriented Software Design: Emerging Research and Opportunities is a pivotal reference source that provides vital research on the development of modern software practices that impact maintenance, design, and developer productivity. While highlighting topics such as augmented reality, distributed computing, and big data processing, this publica-

tion explores the current infrastructure of software systems as well as future advancements. This book is ideally designed for software engineers, IT specialists, data scientists, business professionals, developers, researchers, students, and academicians seeking current research on contemporary software engineering methods.

"Supply Chain Event Management (SCEM)" is one of the major topics in application-oriented Supply Chain Management. However, many solutions lack conceptual precision and currently available client-server SCEM-systems are ill-suited for complex supply networks in today's business environment. Agent-based proactive information logistics promises to overcome existing deficits by providing event-related information to all participants in the distributed environment. Hence, follow-up costs of disruptive events are significantly reduced for all network participants and performance of a supply network is increased. In this book a thorough analysis of the event management problem domain is the starting point to develop a generic agent-based approach to Supply Network Event Management. The main focus lies on practical issues of event management (e.g., seman-

tic interoperability) and economic benefits to be achieved with agent technology in this state-of-the-art problem domain.

Object-oriented programming (OOP) has been the leading paradigm for developing software applications for at least 20 years. Many different methodologies, approaches, and techniques have been created for OOP, such as UML, Unified Process, design patterns, and eXtreme Programming. Yet, the actual process of building good software, particularly large, interactive, and long-lived software, is still emerging. Software engineers familiar with the current crop of methodologies are left wondering, how does all of this fit together for designing and building software in real projects? This handbook from one of the world's leading software architects and his team of software engineers presents guidelines on how to develop high-quality software in an application-oriented way. It answers questions such as: * How do we analyze an application domain utilizing the knowledge and experience of the users? * What is the proper software architecture for large, distributed interactive systems that can utilize UML and design patterns? * Where and how should we utilize the techniques and

methods of the Unified Process and eXtreme Programming? This book brings together the best of research, development, and day-to-day project work. "The strength of the book is that it focuses on the transition from design to implementation in addition to its overall vision about software development." -Bent Bruun Kristensen, University of Southern Denmark, Odense

Software systems play a central role in modern society, and their correctness is often crucially important. Formal specification and verification are promising approaches for ensuring correctness more rigorously than just by testing. This work presents an approach for deductively verifying design-by-contract specifications of object-oriented programs. The approach is based on dynamic logic, and addresses the challenges of modularity and automation using dynamic frames and predicate abstraction.

Here's a complete guide to building reliable component-based software systems. Written by world-renowned experts in the component-based software engineering field, this unique resource helps you manage complex software through the develop-

ment, evaluation and integration of software components. You quickly develop a keen awareness of the benefits and risks to be considered when developing reliable systems using components. A strong software engineering perspective helps you gain a better understanding of software component design, to build systems with stronger requirements, and avoid typical errors throughout the process, leading to improved quality and time to market.

If you want an up-to-date, in-depth understanding of next generation intelligent networks (IN), this book is essential reading. It provides you with a comprehensive survey of current and emerging intelligent telecommunications networks, including underlying software, implementation, deployment and standards. It assesses the influence of mobile networks and IP technology on the directions that IN is taking now, and looks at the way middleware is reducing the dependence of service logic on the underlying network protocols. Moreover, it discusses the role of IN in tomorrow's network. Next Generation Intelligent Networks investigates how service creation is taking on board state-of-the-art distributed object-oriented programming techniques. The book focuses on emerging software architectures for programming and deploying value-added services, and proposes a vision of value-added services in the network of the future. Learn how new initiatives such as Soft switches, JAIN, Parlay, OSA, TINA and CAMEL help you to more effectively handle the new challenges brought on by the rapid growth of cellular mobile networks and the Internet.