



systems, and they do not need to be built to code or pass inspections. The range of complexity in software design is similar. Given available software tools and libraries—many of which are free—hobbyists can build small or short-lived computer apps. Yet, design for software longevity, security, and efficiency can be intricate—as is the design of large-scale systems. How can a software developer prepare to manage such complexity? By understanding the essential building blocks of software design and construction. About the Book: *Software Essentials: Design and Construction* explicitly defines and illustrates the basic elements of software design and construction, providing a solid understanding of control flow, abstract data types (ADTs), memory, type relationships, and dynamic behavior. This text evaluates the benefits and overhead of object-oriented design (OOD) and analyzes software design options. With a structured but hands-on approach, the book: Delineates malleable and stable characteristics of software design Explains how to evaluate the short- and long-term costs and benefits of design decisions Compares and contrasts design solutions, such as composition versus inheritance Includes supportive appendices and a glossary of over 200 common terms Covers key topics such as polymorphism, overloading, and more While extensive examples are given in C# and/or C++, often demonstrating alternative solutions, design—not syntax—remains the focal point of *Software Essentials: Design and Construction*.

Geared to experienced C++ developers who may not be familiar with the more advanced features of the language, and therefore are not using it to its full capabilities Teaches programmers how to think in C++—that is, how to design effective solutions that maximize the power of the language The authors drill down into this notoriously complex language, explaining poorly understood elements of the C++ feature set as well as common pitfalls to avoid Contains several in-depth case studies with working code that's been tested on Windows, Linux, and Solaris platforms

This volume provides an overview of current work in software engineering techniques that can enhance the quality of software. The chapters of this volume, organized by key topic area, create an agenda for the IFIP Working Conference on Software Engineering Techniques, SET 2006. The seven sections of the volume address the following areas: software architectures, modeling, project management, software quality, analysis and verification methods, data management, and software maintenance.

This book and its accompanying CD-ROM, are designed to give readers hands-on experience of C++ programming and the entire software engineering life-cycle. Substantial and realistic case studies are used to illustrate alternative approaches to software development, supported by the SELECT Yourdon CASE (Computer Aided Software Engineering) tool provided with the book.

Designing scalable software in C++ requires more than just a sound understanding of the logical design issues covered in most C++ programming books. To succeed, senior developers, architects, and project leaders need a grasp of high-level physical design concepts with which even expert software developers have little or no experience. This concise, approachable guide takes a practitioner's view of large-scale software development, while providing all the information you need to quickly apply architectural-level physical design concepts in your everyday work. All content is presented in self-contained modules that make it easy to find what you need -- and use it. John Lakos presents crucial new material on runtime dependencies and other architectural issues, as well as realistic small examples that add value by illuminating broad and deep issues in large-scale C++ application design.

????:Richard Helm,Ralph Johnson,John Vlissides ?????:???,??,???

Innovative tools and techniques for the development and design of software systems are essential to the problem solving and planning of software solutions. *Software Design and Development: Concepts, Methodologies, Tools, and Applications* brings together the best practices of theory and implementation in the development of software systems. This reference source is essential for researchers, engineers, practitioners, and scholars seeking the latest knowledge on the techniques, applications, and methodologies for the design and development of software systems.

Large-scale C++ Software Design Addison-Wesley Professional

This collection of articles by well-known experts was originally published in 2000 and is intended for researchers in computer science, practitioners of formal methods, and computer programmers working in safety-critical applications or in the technology of component-based systems. The work brings together several elements of this area that were fast becoming the focus of much research and practice in computing. The introduction by Clemens Szyperski gives a snapshot of research in the field. About half the articles deal with theoretical frameworks, models, and systems of notation; the rest of the book concentrates on case studies by researchers who have built prototype systems and present findings on architectures verification. The emphasis is on advances in the technological infrastructure of component-based systems; how to design and specify reusable components; and how to reason about, verify, and validate systems from components. Thus the book shows how theory might move into practice.

A text on networking theory and practice, providing information on general networking concepts, routing algorithms and protocols, addressing, and mechanics of bridges, routers, switches, and hubs. Describes all major network algorithms and protocols in use today, and explores engineering trade-offs that each different approach represents. Includes chapter homework problems and a glossary. This second edition is expanded to cover recent developments such as VLANs, Fast Ethernet, and AppleTalk. The author is a Distinguished Engineer at Sun Microsystems, Inc., and holds some 50 patents. Annotation copyrighted by Book News, Inc., Portland, OR

An update to the bestselling UML classic, this title has been revised to cover the unified process and Rational Software's processes. Larman also shows developers how to make practical use of the most significant recent developments in object-oriented analysis and design.

Nowadays, engineering large-scale software systems means dealing with complex systems composed of pervasive software components that move around and adapt to nondeterministic and open environments, like the Internet, in order to achieve systems design goals through the coordination of autonomously distributed services. The agent metaphor, in particular software agents and multi-agent systems (MAS), constitutes a promising approach for covering most of the software development life cycle, from conceptual modeling and requirements specification to architectural definition, design, and implementation. This book presents 17 carefully reviewed papers arranged in order to provide a coherent survey of how to exploit agent properties and MAS issues in

today's software systems. The book offers the following topical sections: - software engineering foundations - requirements engineering and software architecture - coordination and mobility - reuse -dependability -empirical studies and applications This book presents a coherent and well-balanced survey of recent advances in software engineering approaches to the design and analysis of realistic large-scale multi-agent systems (MAS). The chapters included are devoted to various techniques and methods used to cope with the complexity of real-world MAS. The power of agent-based software engineering is illustrated using examples that are representative of successful applications. The 16 thoroughly reviewed and revised full papers are organized in topical sections on agent methodologies and processes, requirements engineering and software architectures, modeling languages, and dependability and coordination. Most of the papers were initially presented at the 3rd International Workshop on Software Engineering for Large-Scale Multi-agent Systems, SELMAS 2004, held in Edinburgh, UK in May 2004 in association with ICSE 2004. Other papers were invited to complete coverage of all relevant aspects.

Hispanic Engineer & Information Technology is a publication devoted to science and technology and to promoting opportunities in those fields for Hispanic Americans.

For more than 40 years, Computerworld has been the leading source of technology news and information for IT influencers worldwide. Computerworld's award-winning Web site (Computerworld.com), twice-monthly publication, focused conference series and custom research form the hub of the world's largest global IT media network.

This book contains the refereed proceedings of the 17th International Conference on Agile Software Development, XP 2016, held in Edinburgh, UK, in May 2016. While agile development has already become mainstream in industry, this field is still constantly evolving and continues to spur an enormous interest both in industry and academia. To this end, the XP conference attracts a large number of software practitioners and researchers, providing a rare opportunity for interaction between the two communities. The 14 full papers accepted for XP 2016 were selected from 42 submissions. Additionally, 11 experience reports (from 25 submissions) 5 empirical studies (out of 12 submitted) and 5 doctoral papers (from 6 papers submitted) were selected, and in each case the authors were shepherded by an experienced researcher. Generally, all of the submitted papers went through a rigorous peer-review process.

API Design for C++ provides a comprehensive discussion of Application Programming Interface (API) development, from initial design through implementation, testing, documentation, release, versioning, maintenance, and deprecation. It is the only book that teaches the strategies of C++ API development, including interface design, versioning, scripting, and plug-in extensibility. Drawing from the author's experience on large scale, collaborative software projects, the text offers practical techniques of API design that produce robust code for the long term. It presents patterns and practices that provide real value to individual developers as well as organizations. API Design for C++ explores often overlooked issues, both technical and non-technical, contributing to successful design decisions that product high quality, robust, and long-lived APIs. It focuses on various API styles and patterns that will allow you to produce elegant and durable libraries. A discussion on testing strategies concentrates on automated API testing techniques rather than attempting to include end-user application testing techniques such as GUI testing, system testing, or manual testing. Each concept is illustrated with extensive C++ code examples, and fully functional examples and working source code for experimentation are available online. This book will be helpful to new programmers who understand the fundamentals of C++ and who want to advance their design skills, as well as to senior engineers and software architects seeking to gain new expertise to complement their existing talents. Three specific groups of readers are targeted: practicing software engineers and architects, technical managers, and students and educators. The only book that teaches the strategies of C++ API development, including design, versioning, documentation, testing, scripting, and extensibility. Extensive code examples illustrate each concept, with fully functional examples and working source code for experimentation available online.

Covers various API styles and patterns with a focus on practical and efficient designs for large-scale long-term projects. This research-oriented book presents key contributions on architecting the digital transformation. It includes the following main sections covering 20 chapters: · Digital Transformation · Digital Business · Digital Architecture · Decision Support · Digital Applications Focusing on digital architectures for smart digital products and services, it is a valuable resource for researchers, doctoral students, postgraduates, graduates, undergraduates, academics and practitioners interested in digital transformation.

The purpose of large-scale software architecture is to capture and describe practical representations to make development teams more effective. In this book the authors show how to utilise software architecture as a tool to guide the development instead of capturing the architectural details after all the design decisions have been made. \* Offers a concise description of UML usage for large-scale architecture \* Discusses software architecture and design principles \* Technology and vendor independent

This book explores coordination within and between teams in the context of large-scale agile software development, providing readers a deeper understanding of how coordinated action between teams is achieved in multiteam systems. An exploratory multiple case study with five multiteam systems and a total of 66 interviewees from development teams at SAP SE is presented and analyzed. In addition, the book explores stereotypes of coordination in large-scale agile settings and shares new perspectives on integrating conditions for coordination. No previous study has researched this topic with a similar data set, consisting of insights from professional software development teams. As such, the book will be of interest to all researchers and practitioners whose work involves software product development across several teams.

The topic is of prime importance to software professionals involved in large development efforts such as databases, operating systems, compilers, and frameworks. This volume explains the process of decomposing large systems into physical (not inheritance) hierarchies of small, manageable components. Concepts and techniques are illustrated with "war stories" from the

development firm, Mentor Graphics, as well as with a large-scale example comprising some 12,000 lines of code. Annotation copyright by Book News, Inc., Portland, OR

Software Design for Engineers and Scientists integrates three core areas of computing: . Software engineering - including both traditional methods and the insights of 'extreme programming' . Program design - including the analysis of data structures and algorithms . Practical object-oriented programming Without assuming prior knowledge of any particular programming language, and avoiding the need for students to learn from separate, specialised Computer Science texts, John Robinson takes the reader from small-scale programming to competence in large software projects, all within one volume. Copious examples and case studies are provided in C++. The book is especially suitable for undergraduates in the natural sciences and all branches of engineering who have some knowledge of computing basics, and now need to understand and apply software design to tasks like data analysis, simulation, signal processing or visualisation. John Robinson introduces both software theory and its application to problem solving using a range of design principles, applied to the creation of medium-sized systems, providing key methods and tools for designing reliable, efficient, maintainable programs. The case studies are presented within scientific contexts to illustrate all aspects of the design process, allowing students to relate theory to real-world applications. Core computing topics - usually found in separate specialised texts - presented to meet the specific requirements of science and engineering students

Demonstrates good practice through applications, case studies and worked examples based in real-world contexts

More than ever, mission-critical and business-critical applications depend on object-oriented (OO) software. Testing techniques tailored to the unique challenges of OO technology are necessary to achieve high reliability and quality. "Testing Object-Oriented Systems: Models, Patterns, and Tools" is an authoritative guide to designing and automating test suites for OO applications. This comprehensive book explains why testing must be model-based and provides in-depth coverage of techniques to develop testable models from state machines, combinational logic, and the Unified Modeling Language (UML). It introduces the test design pattern and presents 37 patterns that explain how to design responsibility-based test suites, how to tailor integration and regression testing for OO code, how to test reusable components and frameworks, and how to develop highly effective test suites from use cases. Effective testing must be automated and must leverage object technology. The author describes how to design and code specification-based assertions to offset testability losses due to inheritance and polymorphism. Fifteen micro-patterns present oracle strategies--practical solutions for one of the hardest problems in test design. Seventeen design patterns explain how to automate your test suites with a coherent OO test harness framework. The author provides thorough coverage of testing issues such as: The bug hazards of OO programming and differences from testing procedural code How to design responsibility-based tests for classes, clusters, and subsystems using class invariants, interface data flow models, hierarchic state machines, class associations, and scenario analysis How to support reuse by effective testing of abstract classes, generic classes, components, and frameworks How to choose an integration strategy that supports iterative and incremental development How to achieve comprehensive system testing with testable use cases How to choose a regression test approach How to develop expected test results and evaluate the post-test state of an object How to automate testing with assertions, OO test drivers, stubs, and test frameworks Real-world experience, world-class best practices, and the latest research in object-oriented testing are included. Practical examples illustrate test design and test automation for Ada 95, C++, Eiffel, Java, Objective-C, and Smalltalk. The UML is used throughout, but the test design patterns apply to systems developed with any OO language or methodology.

0201809389B04062001

Advances in networking technology have revitalized the investigation of agent technology as a promising paradigm for engineering complex distributed software systems. Agent technology has been applied to a wide range of application - mains, including e-commerce, human-computer interfaces, telecommunications, and software assistants. Multi-agent systems (MASs) and their underlying theories provide a more natural support for ensuring important properties such as autonomy, mobility, environment heterogeneity, organization, openness, and intelligence. As a consequence, agent-based systems are likely to provide new approaches to dealing with the complexity of developing and maintaining modern software. However, developing robust large-scale agent-based systems will require new software engineering approaches. There are currently many methods and techniques for working with individual agents or with systems built using only a few agents. Unfortunately, agent-based software engineering is still in its infancy and existing software engineering approaches are unable to cope with large MASs. The complexity associated with a large MAS is considerable. When a huge number of agents interact over heterogeneous environments, various phenomena occur which are not as easy to capture as when only a few agents are working together. As the multiple software agents are highly collaborative and operate in networked environments, they have to be context-aware and deal with environment uncertainty. This makes their coordination and management more difficult and increases the likelihood of exceptional situations, such as security holes, privacy violations, and unexpected global effects. Moreover, as users and software engineers delegate more autonomy to their MASs, and put more trust in their results, new concerns arise in real-life applications.

????????????????

The puzzles and problems in Exceptional C++ not only entertain, they will help you hone your skills to become the sharpest C++ programmer you can be. Many of these problems are culled from the famous Guru of the Week feature of the Internet newsgroup comp.lang.c++. moderated, expanded and updated to conform to the official ISO/ANSI C++ Standard. Try your skills against the C++ masters and come away with the insight and experience to create more efficient, effective, robust, and portable C++ code. More C++ Gems picks up where the first book left off, presenting tips, tricks, proven strategies, easy-to-follow techniques, and usable source code.

This book contains the refereed proceedings of the 15th International Conference on Agile Software Development, XP 2014, held in Rome, Italy, in May 2014. Because of the wide application of agile approaches in industry, the need for collaboration between academics and practitioners has increased in order to develop the body of knowledge available to support managers, system engineers, and software engineers in their managerial/economic and architectural/project/technical decisions. Year after year, the XP conference has facilitated such improvements and provided evidence on the advantages of agile methodologies by examining the latest theories, practical applications, and implications of agile and lean methods. The 15 full papers, seven short papers, and four experience reports accepted for XP 2014 were selected from 59 submissions and are organized in sections on: agile development, agile challenges and contracting, lessons learned and agile maturity, how to evolve software engineering teaching, methods and metrics, and lean development.

