

Software Design Decoded 66 Ways Experts Think

Don't engineer by coincidence-design it like you mean it! Filled with practical techniques, Design It! is the perfect introduction to software architecture for programmers who are ready to grow their design skills. Lead your team as a software architect, ask the right stakeholders the right questions, explore design options, and help your team implement a system that promotes the right -ilities. Share your design decisions, facilitate collaborative design workshops that are fast, effective, and fun-and develop more awesome software! With dozens of design methods, examples, and practical know-how, Design It! shows you how to become a software architect. Walk through the core concepts every architect must know, discover how to apply them, and learn a variety of skills that will make you a better programmer, leader, and designer. Uncover the big ideas behind software architecture and gain confidence working on projects big and small. Plan, design, implement, and evaluate software architectures and collaborate with your team, stakeholders, and other architects. Identify the right stakeholders and understand their needs, dig for architecturally significant requirements, write amazing quality attribute scenarios, and make confident decisions. Choose technologies based on their architectural impact, facilitate architecture-centric design workshops, and evaluate architectures using lightweight, effective methods. Write lean architecture descriptions people love to read. Run an architecture design studio, implement the architecture you've designed, and grow your team's architectural knowledge. Good design requires good communication. Talk about your software architecture with stakeholders using whiteboards, documents, and code, and apply architecture-focused design methods in your day-to-day practice. Hands-on exercises, real-world scenarios, and practical team-based decision-making tools will get everyone on board and give you the experience you need to become a confident software architect.

For one-semester courses in software engineering. Introduces software engineering techniques for developing software products and apps With Engineering Software Products, author Ian Sommerville takes a unique approach to teaching software engineering and focuses on the type of software products and apps that are familiar to students, rather than focusing on project-based techniques. Written in an informal style, this book focuses on software engineering techniques that are relevant for software product engineering. Topics covered include personas and scenarios, cloud-based software, microservices, security and privacy and DevOps. The text is designed for students taking their first course in software engineering with experience in programming using a modern programming language such as Java, Python or Ruby.

Users can dramatically improve the design, performance, and manageability of object-oriented code without altering its interfaces or behavior. "Refactoring" shows users exactly how to spot the best opportunities for refactoring and exactly how to do it, step by step.

This is a practical book for computer engineers who want to understand or implement hardware/software systems. It focuses on problems that require one to combine hardware design with software design – such problems can be solved with hardware/software codesign. When used properly, hardware/software co- sign works better than hardware design or software design alone: it can improve the overall performance of digital systems, and it can shorten their design time. Hardware/software codesign can help a designer to make trade-offs between the ?exibility and the performanceof a digital system. To achieve this, a designer needs to combine two radically different ways of design: the sequential way of dec- position in time, using software, with the parallel way of decomposition in space, using hardware. Intended Audience This book assumes that you have a basic understandingof hardware that you are - miliar with standard digital hardware componentssuch as registers, logic gates, and components such as multiplexers and arithmetic operators. The book also assumes that you know how to write a program in C. These topics are usually covered in an introductory course on computer engineering or in a combination of courses on digital design and software engineering.

If you're a student studying computer science or a software developer preparing for technical interviews, this practical book will help you learn and review some of the most important ideas in software engineering—data structures and algorithms—in a way that's clearer, more concise, and more engaging than other materials. By emphasizing practical knowledge and skills over theory, author Allen Downey shows you how to use data structures to implement efficient algorithms, and then analyze and measure their performance. You'll explore the important classes in the Java collections framework (JCF), how they're implemented, and how they're expected to perform. Each chapter presents hands-on exercises supported by test code online. Use data structures such as lists and maps, and understand how they work Build an application that reads Wikipedia pages, parses the contents, and navigates the resulting data tree Analyze code to predict how fast it will run and how much memory it will require Write classes that implement the Map interface, using a hash table and binary search tree Build a simple web search engine with a crawler, an indexer that stores web page contents, and a retriever that returns user query results Other books by Allen Downey include Think Java, Think Python, Think Stats, and Think Bayes.

A revised new edition of the bestselling toolkit for creating, building, and maintaining a strong brand From research and analysis through brand strategy, design development through application design, and identity standards through launch and governance, Designing Brand Identity, Fourth Edition offers brand managers, marketers, and designers a proven, universal five-phase process for creating and implementing effective brand identity. Enriched by new case studies showcasing successful world-class brands, this Fourth Edition brings readers up to date with a detailed look at the latest trends in branding, including social networks, mobile devices, global markets, apps, video, and virtual brands. Features more than 30 all-new case studies showing best practices and world-class Updated to include more than 35 percent new material Offers a proven, universal five-phase process and methodology for creating and implementing effective brand identity

The Architecture of Computer Hardware, Systems Software and Networking is designed help students majoring in information technology (IT) and information systems (IS) understand the structure and operation of computers and computer-based devices. Requiring only basic computer skills, this accessible textbook introduces the basic principles of system architecture and explores current technological practices and trends using clear, easy-to-understand language. Throughout the text, numerous relatable examples, subject-specific illustrations, and in-depth case studies reinforce key learning points and show students how important concepts are applied in the real world. This fully-updated sixth edition features a wealth of new and revised content that reflects today's technological landscape. Organized into five parts, the book first explains the role of the computer in information systems and provides an overview of its components.

Subsequent sections discuss the representation of data in the computer, hardware architecture and operational concepts, the basics of computer networking, system software and operating systems, and various interconnected systems and components. Students are introduced to the material using ideas already familiar to them, allowing them to gradually build upon what they have learned without being overwhelmed and develop a deeper knowledge of computer architecture.

Integrate big data into business to drive competitive advantage and sustainable success Big Data MBA brings insight and expertise to leveraging big data in business so you can harness the power of analytics and gain a true business advantage. Based on a practical framework with supporting methodology and hands-on exercises, this book helps identify where and how big data can help you transform your business. You'll learn how to exploit new sources of customer, product, and operational data, coupled with advanced analytics and data science, to optimize key processes, uncover monetization opportunities, and create new sources of competitive differentiation. The discussion includes guidelines for operationalizing analytics, optimal organizational structure, and using analytic insights throughout your organization's user experience to customers and front-end employees alike. You'll learn to "think like a data scientist" as you build upon the decisions your business is trying to make, the hypotheses you need to test, and the predictions you need to produce. Business stakeholders no longer need to relinquish control of data and analytics to IT. In fact, they must champion the organization's data collection and analysis efforts. This book is a primer on the business approach to analytics, providing the practical understanding you need to convert data into opportunity. Understand where and how to leverage big data Integrate analytics into everyday operations Structure your organization to drive analytic insights Optimize processes, uncover opportunities, and stand out from the rest Help business stakeholders to "think like a data scientist" Understand appropriate business application of different analytic techniques If you want data to transform your business, you need to know how to put it to use. Big Data MBA shows you how to implement big data and analytics to make better decisions.

Table of contents

Accident investigation and risk assessment have for decades focused on the human factor, particularly 'human error'. This bias towards performance failures leads to a neglect of normal performance. It assumes that failures and successes have different origins so there is little to be gained from studying them together. Erik Hollnagel believes this assumption is false and that safety cannot be attained only by eliminating risks and failures. The alternative is to understand why things go right and to amplify that. The ETTO Principle looks at the common trait of people at work to adjust what they do to match the conditions. It proposes that this efficiency-thoroughness trade-off (ETTO) is normal. While in some cases the adjustments may lead to adverse outcomes, these are due to the same processes that produce successes.

Software Design: Creating Solutions for Ill-Structured Problems, Third Edition provides a balanced view of the many and varied software design practices used by practitioners. The book provides a general overview of software design within the context of software development and as a means of addressing ill-structured problems. The third edition has been expanded and reorganised to focus on the structure and process aspects of software design, including architectural issues, as well as design notations and models. It also describes a variety of different ways of creating design solutions such as plan-driven development, agile approaches, patterns, product lines, and other forms. Features •Includes an overview and review of representation forms used for modelling design solutions •Provides a concise review of design practices and how these relate to ideas about software architecture •Uses an evidence-informed basis for discussing design concepts and when their use is appropriate This book is suitable for undergraduate and graduate students taking courses on software engineering and software design, as well as for software engineers. Author David Budgen is a professor emeritus of software engineering at Durham University. His research interests include evidence-based software engineering (EBSE), software design, and healthcare informatics.

According to Rosalind Picard, if we want computers to be genuinely intelligent and to interact naturally with us, we must give computers the ability to recognize, understand, even to have and express emotions. The latest scientific findings indicate that emotions play an essential role in decision making, perception, learning, and more—that is, they influence the very mechanisms of rational thinking. Not only too much, but too little emotion can impair decision making. According to Rosalind Picard, if we want computers to be genuinely intelligent and to interact naturally with us, we must give computers the ability to recognize, understand, even to have and express emotions. Part 1 of this book provides the intellectual framework for affective computing. It includes background on human emotions, requirements for emotionally intelligent computers, applications of affective computing, and moral and social questions raised by the technology. Part 2 discusses the design and construction of affective computers. Although this material is more technical than that in Part 1, the author has kept it less technical than typical scientific publications in order to make it accessible to newcomers. Topics in Part 2 include signal-based representations of emotions, human affect recognition as a pattern recognition and learning problem, recent and ongoing efforts to build models of emotion for synthesizing emotions in computers, and the new application area of affective wearable computers.

The performance of software systems is dramatically affected by how well software designers understand the basic hardware technologies at work in a system. Similarly, hardware designers must understand the far-reaching effects their design decisions have on software applications. For readers in either category, this classic introduction to the field provides a look deep into the computer. It demonstrates the relationships between the software and hardware and focuses on the foundational concepts that are the basis for current computer design.

Join the technological revolution that's taking the financial world by storm. Mastering Bitcoin is your guide through the seemingly complex world of bitcoin, providing the knowledge you need to participate in the internet of money. Whether you're building the next killer app, investing in a startup, or simply curious about the technology, this revised and expanded second edition provides essential detail to get you started. Bitcoin, the first successful decentralized digital currency, is still in its early stages and yet it's already spawned a multi-billion-dollar global economy open to anyone with the knowledge and passion to participate. Mastering Bitcoin provides the knowledge. You simply supply the passion. The second edition includes: A broad introduction of bitcoin and its underlying blockchain—ideal for non-technical users, investors, and business executives An explanation of the technical foundations of bitcoin and cryptographic currencies for developers, engineers, and software and systems architects Details of the bitcoin decentralized network, peer-to-peer architecture, transaction lifecycle, and security principles New developments such as Segregated Witness, Payment Channels, and Lightning Network A deep dive into blockchain applications, including how to combine the building blocks offered by this platform into higher-level applications User stories, analogies, examples, and code snippets illustrating key technical concepts

Until the late 1980s, information processing was associated with large mainframe computers and huge tape drives. During the 1990s, this trend shifted toward information processing with personal computers, or PCs. The trend toward miniaturization continues and in the future the majority of information processing systems will be small mobile computers, many of which will be embedded into larger products and interfaced to the physical environment. Hence, these kinds of systems are called embedded systems. Embedded systems together with their physical environment are called cyber-physical systems.

Examples include systems such as transportation and fabrication equipment. It is expected that the total market volume of embedded systems will be significantly larger than that of traditional information processing systems such as PCs and mainframes. Embedded systems share a number of common characteristics. For example, they must be dependable, efficient, meet real-time constraints and require customized user interfaces (instead of generic keyboard and mouse interfaces). Therefore, it makes sense to consider common principles of embedded system design. Embedded System Design starts with an introduction into the area and a survey of specification models and languages for embedded and cyber-physical systems. It provides a brief overview of hardware devices used for such systems and presents the essentials of system software for embedded systems, like real-time operating systems. The book also discusses evaluation and validation techniques for embedded systems. Furthermore, the book presents an overview of techniques for mapping applications to execution platforms. Due to the importance of resource efficiency, the book also contains a selected set of optimization techniques for embedded systems, including special compilation techniques. The book closes with a brief survey on testing. Embedded System Design can be used as a text book for courses on embedded systems and as a source which provides pointers to relevant material in the area for PhD students and teachers. It assumes a basic knowledge of information processing hardware and software. Courseware related to this book is available at <http://ls12-www.cs.tu-dortmund.de/~marwedel>.

A theoretical examination of the surprising emergence of software as a guiding metaphor for our neoliberal world. New media thrives on cycles of obsolescence and renewal: from celebrations of cyber-everything to Y2K, from the dot-com bust to the next big things—mobile mobs, Web 3.0, cloud computing. In Programmed Visions, Wendy Hui Kyong Chun argues that these cycles result in part from the ways in which new media encapsulates a logic of programmability. New media proliferates “programmed visions,” which seek to shape and predict—even embody—a future based on past data. These programmed visions have also made computers, based on metaphor, metaphors for metaphor itself, for a general logic of substitutability. Chun argues that the clarity offered by software as metaphor should make us pause, because software also engenders a profound sense of ignorance: who knows what lurks behind our smiling interfaces, behind the objects we click and manipulate? The combination of what can be seen and not seen, known (knowable) and not known—its separation of interface from algorithm and software from hardware—makes it a powerful metaphor for everything we believe is invisible yet generates visible, logical effects, from genetics to the invisible hand of the market, from ideology to culture.

Beginning with a basic primer on reverse engineering—including computer internals, operating systems, and assembly language—and then discussing the various applications of reverse engineering, this book provides readers with practical, in-depth techniques for software reverse engineering. The book is broken into two parts, the first deals with security-related reverse engineering and the second explores the more practical aspects of reverse engineering. In addition, the author explains how to reverse engineer a third-party software library to improve interfacing and how to reverse engineer a competitor's software to build a better product. * The first popular book to show how software reverse engineering can help defend against security threats, speed up development, and unlock the secrets of competitive products * Helps developers plug security holes by demonstrating how hackers exploit reverse engineering techniques to crack copy-protection schemes and identify software targets for viruses and other malware * Offers a primer on advanced reverse-engineering, delving into "disassembly"-code-level reverse engineering—and explaining how to decipher assembly language

The new RISC-V Edition of Computer Organization and Design features the RISC-V open source instruction set architecture, the first open source architecture designed to be used in modern computing environments such as cloud computing, mobile devices, and other embedded systems. With the post-PC era now upon us, Computer Organization and Design moves forward to explore this generational change with examples, exercises, and material highlighting the emergence of mobile computing and the Cloud. Updated content featuring tablet computers, Cloud infrastructure, and the x86 (cloud computing) and ARM (mobile computing devices) architectures is included. An online companion Web site provides advanced content for further study, appendices, glossary, references, and recommended reading. Features RISC-V, the first such architecture designed to be used in modern computing environments, such as cloud computing, mobile devices, and other embedded systems Includes relevant examples, exercises, and material highlighting the emergence of mobile computing and the cloud

Cleanroom software engineering is a process for developing and certifying high-reliability software. Combining theory-based engineering technologies in project management, incremental development, software specification and design, correctness verification, and statistical quality certification, the Cleanroom process answers today's call for more reliable software and provides methods for more cost-effective software development. Cleanroom originated with Harlan D. Mills, an IBM Fellow and a visionary in software engineering. Written by colleagues of Mills and some of the most experienced developers and practitioners of Cleanroom, Cleanroom Software Engineering provides a roadmap for software management, development, and testing as disciplined engineering practices. This book serves both as an introduction for those new to Cleanroom and as a reference guide for the growing practitioner community. Readers will discover a proven way to raise both quality and productivity in their software-intensive products, while reducing costs. Highlights Explains basic Cleanroom theory Introduces the sequence-based specification method Elaborates the full management, development, and certification process in a Cleanroom Reference Model (CRM) Shows how the Cleanroom process dovetails with the SEI's Capability Maturity Model for Software (CMM) Includes a large case study to illustrate how Cleanroom methods scale up to large projects.

Explore various verticals in software engineering through high-end systems using Python Key Features Master the tools and techniques used in software engineering Evaluates available database options and selects one for the final Central Office system-components Experience the iterations software go through and craft enterprise-grade systems Book Description Software Engineering is about more than just writing code—it includes a host of soft skills that apply to almost any development effort, no matter what the language, development methodology, or scope of the project. Being a senior developer all but requires awareness of how those skills, along with their expected technical counterparts, mesh together through a project's life cycle. This book walks you through that discovery by going over the entire life cycle of a multi-tier system and its related software projects. You'll see what happens before any development takes place, and what impact the decisions and designs made at each step have on the development process. The development of the entire project, over the course of several iterations based on real-world Agile iterations, will be executed, sometimes starting from nothing, in one of the fastest growing languages in the world—Python. Application of practices in Python will be laid out, along with a number of Python-specific capabilities that are often overlooked. Finally, the book will implement a high-performance computing solution, from first principles through complete foundation. What you will learn Understand what happens over the course of a system's life (SDLC) Establish what to expect from the pre-development life cycle steps Find out how the development-specific phases of the SDLC affect development Uncover what a real-world development process might be like, in an Agile way Find out how to do more than just write the code Identify the

existence of project-independent best practices and how to use them Find out how to design and implement a high-performance computing process Who this book is for Hands-On Software Engineering with Python is for you if you are a developer having basic understanding of programming and its paradigms and want to skill up as a senior programmer. It is assumed that you have basic Python knowledge.

Salary surveys worldwide regularly place software architect in the top 10 best jobs, yet no real guide exists to help developers become architects. Until now. This book provides the first comprehensive overview of software architecture's many aspects. Aspiring and existing architects alike will examine architectural characteristics, architectural patterns, component determination, diagramming and presenting architecture, evolutionary architecture, and many other topics. Mark Richards and Neal Ford—hands-on practitioners who have taught software architecture classes professionally for years—focus on architecture principles that apply across all technology stacks. You'll explore software architecture in a modern light, taking into account all the innovations of the past decade. This book examines: Architecture patterns: The technical basis for many architectural decisions Components: Identification, coupling, cohesion, partitioning, and granularity Soft skills: Effective team management, meetings, negotiation, presentations, and more Modernity: Engineering practices and operational approaches that have changed radically in the past few years Architecture as an engineering discipline: Repeatable results, metrics, and concrete valuations that add rigor to software architecture

The real challenge of programming isn't learning a language's syntax—it's learning to creatively solve problems so you can build something great. In this one-of-a-kind text, author V. Anton Spraul breaks down the ways that programmers solve problems and teaches you what other introductory books often ignore: how to Think Like a Programmer. Each chapter tackles a single programming concept, like classes, pointers, and recursion, and open-ended exercises throughout challenge you to apply your knowledge. You'll also learn how to: –Split problems into discrete components to make them easier to solve –Make the most of code reuse with functions, classes, and libraries –Pick the perfect data structure for a particular job –Master more advanced programming tools like recursion and dynamic memory –Organize your thoughts and develop strategies to tackle particular types of problems Although the book's examples are written in C++, the creative problem-solving concepts they illustrate go beyond any particular language; in fact, they often reach outside the realm of computer science. As the most skillful programmers know, writing great code is a creative art—and the first step in creating your masterpiece is learning to Think Like a Programmer.

Principles of Computer System Design is the first textbook to take a principles-based approach to the computer system design. It identifies, examines, and illustrates fundamental concepts in computer system design that are common across operating systems, networks, database systems, distributed systems, programming languages, software engineering, security, fault tolerance, and architecture. Through carefully analyzed case studies from each of these disciplines, it demonstrates how to apply these concepts to tackle practical system design problems. To support the focus on design, the text identifies and explains abstractions that have proven successful in practice such as remote procedure call, client/service organization, file systems, data integrity, consistency, and authenticated messages. Most computer systems are built using a handful of such abstractions. The text describes how these abstractions are implemented, demonstrates how they are used in different systems, and prepares the reader to apply them in future designs. The book is recommended for junior and senior undergraduate students in Operating Systems, Distributed Systems, Distributed Operating Systems and/or Computer Systems Design courses; and professional computer systems designers. Features: Concepts of computer system design guided by fundamental principles. Cross-cutting approach that identifies abstractions common to networking, operating systems, transaction systems, distributed systems, architecture, and software engineering. Case studies that make the abstractions real: naming (DNS and the URL); file systems (the UNIX file system); clients and services (NFS); virtualization (virtual machines); scheduling (disk arms); security (TLS). Numerous pseudocode fragments that provide concrete examples of abstract concepts. Extensive support. The authors and MIT OpenCourseWare provide on-line, free of charge, open educational resources, including additional chapters, course syllabi, board layouts and slides, lecture videos, and an archive of lecture schedules, class assignments, and design projects.

Widely praised for its balanced treatment of computer ethics, Ethics for the Information Age offers a modern presentation of the moral controversies surrounding information technology. Topics such as privacy and intellectual property are explored through multiple ethical theories, encouraging readers to think critically about these issues and to make their own ethical decisions.

What are the ingredients of robust, elegant, flexible, and maintainable software architecture? Beautiful Architecture answers this question through a collection of intriguing essays from more than a dozen of today's leading software designers and architects. In each essay, contributors present a notable software architecture, and analyze what makes it innovative and ideal for its purpose. Some of the engineers in this book reveal how they developed a specific project, including decisions they faced and tradeoffs they made. Others take a step back to investigate how certain architectural aspects have influenced computing as a whole. With this book, you'll discover: How Facebook's architecture is the basis for a data-centric application ecosystem The effect of Xen's well-designed architecture on the way operating systems evolve How community processes within the KDE project help software architectures evolve from rough sketches to beautiful systems How creeping featurism has helped GNU Emacs gain unanticipated functionality The magic behind the Jikes RVM self-optimizable, self-hosting runtime Design choices and building blocks that made Tandem the choice platform in high-availability environments for over two decades Differences and similarities between object-oriented and functional architectural views How architectures can affect the software's evolution and the developers' engagement Go behind the scenes to learn what it takes to design elegant software architecture, and how it can shape the way you approach your own projects, with Beautiful Architecture.

This book provides the advanced issues of FPGA design as the underlying theme of the work. In practice, an engineer typically needs to be mentored for several years before these principles are appropriately utilized. The topics that will be discussed in this book are essential to designing FPGA's beyond moderate complexity. The goal of the book is to present practical design techniques that are otherwise only available through mentorship and real-world experience.

Embedded Software Development With C offers both an effectual reference for professionals and researchers, and a valuable learning tool for students by laying the groundwork for a solid foundation in the hardware and software aspects of embedded systems development. Key features include a resource for the fundamentals of embedded systems design and development with an emphasis on software, an exploration of the 8051 microcontroller as it pertains to embedded systems, comprehensive tutorial materials for instructors to provide students with labs of varying lengths and levels of difficulty, and supporting website including all sample codes, software tools and links to additional online references.

Teaches readers how to test and analyze software to achieve an acceptable level of quality at an acceptable cost Readers will be able to minimize software failures, increase quality, and effectively manage

costs Covers techniques that are suitable for near-term application, with sufficient technical background to indicate how and when to apply them Provides balanced coverage of software testing & analysis approaches By incorporating modern topics and strategies, this book will be the standard software-testing textbook

An engaging, illustrated collection of insights revealing the practices and principles that expert software designers use to create great software. What makes an expert software designer? It is more than experience or innate ability. Expert software designers have specific habits, learned practices, and observed principles that they apply deliberately during their design work. This book offers sixty-six insights, distilled from years of studying experts at work, that capture what successful software designers actually do to create great software. The book presents these insights in a series of two-page illustrated spreads, with the principle and a short explanatory text on one page, and a drawing on the facing page. For example, "Experts generate alternatives" is illustrated by the same few balloons turned into a set of very different balloon animals. The text is engaging and accessible; the drawings are thought-provoking and often playful. Organized into such categories as "Experts reflect," "Experts are not afraid," and "Experts break the rules," the insights range from "Experts prefer simple solutions" to "Experts see error as opportunity." Readers learn that "Experts involve the user"; "Experts take inspiration from wherever they can"; "Experts design throughout the creation of software"; and "Experts draw the problem as much as they draw the solution." One habit for an aspiring expert software designer to develop would be to read and reread this entertaining but essential little book. The insights described offer a guide for the novice or a reference for the veteran—in software design or any design profession. A companion web site provides an annotated bibliography that compiles key underpinning literature, the opportunity to suggest additional insights, and more.

This book thoroughly explains how computers work. It starts by fully examining a NAND gate, then goes on to build every piece and part of a small, fully operational computer. The necessity and use of codes is presented in parallel with the appropriate pieces of hardware. The book can be easily understood by anyone whether they have a technical background or not. It could be used as a textbook.

Based on the popular Artech House classic, Digital Communication Systems Engineering with Software-Defined Radio, this book provides a practical approach to quickly learning the software-defined radio (SDR) concepts needed for work in the field. This up-to-date volume guides readers on how to quickly prototype wireless designs using SDR for real-world testing and experimentation. This book explores advanced wireless communication techniques such as OFDM, LTE, WLA, and hardware targeting. Readers will gain an understanding of the core concepts behind wireless hardware, such as the radio frequency front-end, analog-to-digital and digital-to-analog converters, as well as various processing technologies. Moreover, this volume includes chapters on timing estimation, matched filtering, frame synchronization message decoding, and source coding. The orthogonal frequency division multiplexing is explained and details about HDL code generation and deployment are provided. The book concludes with coverage of the WLAN toolbox with OFDM beacon reception and the LTE toolbox with downlink reception. Multiple case studies are provided throughout the book. Both MATLAB and Simulink source code are included to assist readers with their projects in the field.

This collection of short expository, critical and speculative texts offers a field guide to the cultural, political, social and aesthetic impact of software. Experts from a range of disciplines each take a key topic in software and the understanding of software, such as algorithms and logical structures.

The Programmer's Brain unlocks the way we think about code. It offers scientifically sound techniques that can radically improve the way you master new technology, comprehend code, and memorize syntax. You'll learn how to benefit from productive struggle and turn confusion into a learning tool. Along the way, you'll discover how to create study resources as you become an expert at teaching yourself and bringing new colleagues up to speed.

This title gives students an integrated and rigorous picture of applied computer science, as it comes to play in the construction of a simple yet powerful computer system.

Spiritual seekers are feeling an increasingly strong urge from within to connect with and manifest their life purpose. They are seeking out tools and knowledge to awaken to this purpose and are doing whatever it takes to achieve this. This book is the answer to that search. A Soul Contract Reading is an accurate, in depth, channelled system of spiritual interpretation that will empower you to do this by decoding the secrets of the blueprint of your life hidden within your birth name. It is based on an ancient system of numerology that uses Hebrew glyphs, revealing the true essence of the reader's life purpose as well as their karma, talents and goals. This will show you the karmic patterns you face, the talents you have to overcome them, the goals you are trying to achieve and your soul destiny. This book will enable you to discover why you are in the perfect place in your life - be it in crisis, stuck, seeking a new way forward or just needing to make sense of it all. It will empower you to move forward from there. Your Soul Contract reading will place your entire life in context, and will enable you to navigate your life properly for the first time.

Commentary by Swami Tejomayananda. Nirvana Shatkam, also known as atma shatkam, composed by Sri Adi Shankaracharya is a magnificently lucid piece of poetry, a razor-sharp distinction between reality and false identification and the product born from the very highest states of meditation. One by Shankaracharya negates all the usual identifications that we have and repeatedly insists, "chidananda roopa shivoham shivoham."

In the course of their 20+-year engineering careers, authors Brian Fitzpatrick and Ben Collins-Sussman have picked up a treasure trove of wisdom and anecdotes about how successful teams work together. Their conclusion? Even among people who have spent decades learning the technical side of their jobs, most haven't really focused on the human component. Learning to collaborate is just as important to success. If you invest in the "soft skills" of your job, you can have a much greater impact for the same amount of effort. The authors share their insights on how to lead a team effectively, navigate an organization, and build a healthy relationship with the users of your software. This is valuable information from two respected software engineers whose popular series of talks—including "Working with Poisonous People"—has attracted hundreds of thousands of followers.

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."

Software Design Decoded 66 Ways Experts Think MIT Press

[Copyright: 58e3270bb4f239e1c14d122a92967f7b](#)