Debugging

Debugging by Thinking: A Multi-Disciplinary Approach is the first book to apply the wisdom of six disciplines-logic, mathematics, psychology, safety analysis, computer science, and engineering-to the problem of debugging. It uses the methods of literary detectives such as Sherlock Holmes, the techniques of mathematical problem solving, the results of research into the cognitive psychology of human error, the root cause analyses of safety experts, the compiler analyses of computer science, and the processes of modern engineering to define a systematic approach to identifying and correcting software errors. * Language Independent Methods: Examples are given in Java and C++ * Complete source code shows actual bugs, rather than contrived examples * Examples are accessible with no more knowledge than a course in Data Structures and Algorithms requires * A "thought process diary" shows how the author actually resolved the problems as they occurred

Learn to find software bugs faster and discover how other developers have solved similar problems. For intermediate to advanced iOS/macOS developers already familiar with either Swift or Objective-C who want to take their debugging skills to the next level, this book includes topics such as: LLDB and its subcommands and options; low-level components used to extract information from a program; LLDB's Python module; and DTrace and how to write D scripts.

Use Windows debuggers throughout the development

cycle—and build better software Rethink your use of Windows debugging and tracing tools—and learn how to make them a key part of test-driven software development. Led by a member of the Windows Fundamentals Team at Microsoft, you'll apply expert debugging and tracing techniques—and sharpen your C++ and C# code analysis skills—through practical examples and common scenarios. Learn why experienced developers use debuggers in every step of the development process, and not just when bugs appear. Discover how to: Go behind the scenes to examine how powerful Windows debuggers work Catch bugs early in the development cycle with static and runtime analysis tools Gain practical strategies to tackle the most common code defects Apply expert tricks to handle user-mode and kernel-mode debugging tasks Implement postmortem techniques such as JIT and dump debugging Debug the concurrency and security aspects of your software Use debuggers to analyze interactions between your code and the operating system Analyze software behavior with Xperf and the Event Tracing for Windows (ETW) framework

Tips for the practical use of debuggers, such as NuMega SoftIce, Microsoft Visual Studio Debugger, and Microsoft Kernel Debugger, with minimum binding to a specific environment are disclosed in this debugger guide. How debuggers operate and how to overcome obstacles and repair debuggers is demonstrated. Programmers will learn how to look at what is inside a computer system, how to reconstruct the operating algorithm of a program distributed without source code, how to modify the

program, and how to debug drivers. The use of debugging applications and drivers in Windows and Unix operating systems on Intel Pentium/DEC Alpha-based processors is also detailed.

Debugging Embedded Microprocessor Systems provides techniques for engineers, technicians, and students who need to correct design faults in embedded systems. Using real-world scenarios, designers can learn practical, time-saving ways to avoid and repair potentially costly problems. Prevention is stressed. In this book, the author addresses hardware and software issues, including up-front design techniques to prevent bugs and contain design creep. Practical advice includes descriptions of common tools which can be used to help identify and repair bugs, as well as test routines. RTOS and embedded PC environments are also covered. Each chapter of Debugging Embedded Microprocessor Systems opens with an example design problem which illustrates real-world issues such as design changes, time pressures, equipment or component availability, etc. Case studies of past debugging projects are presented in the final chapter. Addresses real-world issues like design changes, time pressures, equipment or component availability Practical, time-saving methods for preventing and correcting design problems Covers debugging tools and programmer test routines

A troubleshooting guide to the debugging of Visual FoxPro applications examines a variety of tools available to Visual FoxPro programmers and discusses such topics as quality insurance, applying scientific methods to debugging, VFP debugging tools, and enterprise

solutions. Original. (Advanced)

This book teaches by example. It walks in detail through development of a sample application, illustrating each step via complete working code and either screenshots or console snippets. The cumbersome and time consuming task of debugging will be a cake walk with this book. If you are a Django application developer who wants to create robust applications quickly that work well and are easy to maintain in the long term, this book is for you. This book is the right pick if you want to be smartly tutored to make best use of Django's rich testing and debugging support and make testing an effortless task. Basic knowledge of Python, Django, and the overall structure of a database-driven web application is assumed. However, the code samples are fully explained so that even beginners who are new to the area can learn a great deal from this book.

Debugging Strategies for .NET Developers teaches developers how to think about debugging in Microsoft .NET rather than with the specific tools. Author Darin Dillon describes debugging concepts, such as assertions and logging, and immediately follows each discussion with an example from his experiences of when that technique was used to solve a real-world bug. While other debugging books focus on obscure techniques for advanced users, this book is a highly readable exploration that conveys the basic thought process of debugging, as well as the specific techniques and when to apply those techniques.

Equation-based object-oriented (EOO) modeling languages such as Modelica provide a convenient, declarative method

for describing models of cyber-physical systems. Because of the ease of use of EOO languages, large and complex models can be built with limited effort. However, current state-of-the-art tools do not provide the user with enough information when errors appear or simulation results are wrong. It is of paramount importance that such tools should give the user enough information to correct errors or understand where the problems that lead to wrong simulation results are located. However, understanding the model translation process of an EOO compiler is a daunting task that not only requires knowledge of the numerical algorithms that the tool executes during simulation, but also the complex symbolic transformations being performed. As part of this work, methods have been developed and explored where the EOO tool, an enhanced Modelica compiler, records the transformations during the translation process in order to provide better diagnostics, explanations, and analysis. This information is used to generate better error-messages during translation. It is also used to provide better debugging for a simulation that produces unexpected results or where numerical methods fail. Meeting deadlines is particularly important for real-time applications. It is usually essential to identify possible bottlenecks and either simplify the model or give hints to the compiler that enable it to generate faster code. When profiling and measuring execution times of parts of the model the recorded information can also be used to find out why a particular system model executes slowly. Combined with debugging information, it is possible to find out why this system of equations is slow to solve, which helps understanding what can be done to simplify the model. A tool with a graphical user interface has been developed to make debugging and performance profiling easier. Both debugging and profiling have been combined into a single view so that performance metrics are mapped to equations, which are

mapped to debugging information. The algorithmic part of Modelica was extended with meta-modeling constructs (MetaModelica) for language modeling. In this context a quite general approach to debugging and compilation from (extended) Modelica to C code was developed. That makes it possible to use the same executable format for simulation executables as for compiler bootstrapping when the compiler written in MetaModelica compiles itself. Finally, a method and tool prototype suitable for speeding up simulations has been developed. It works by partitioning the model at appropriate places and compiling a simulation executable for a suitable parallel platform.

Debugging becomes more and more the bottleneck to chip design productivity, especially while developing modern complex integrated circuits and systems at the Electronic System Level (ESL). Today, debugging is still an unsystematic and lengthy process. Here, a simple reporting of a failure is not enough, anymore. Rather, it becomes more and more important not only to find many errors early during development but also to provide efficient methods for their isolation. In Debugging at the Electronic System Level the state-of-the-art of modeling and verification of ESL designs is reviewed. There, a particular focus is taken onto SystemC. Then, a reasoning hierarchy is introduced. The hierarchy combines well-known debugging techniques with whole new techniques to improve the verification efficiency at ESL. The proposed systematic debugging approach is supported amongst others by static code analysis, debug patterns, dynamic program slicing, design visualization, property generation, and automatic failure isolation. All techniques were empirically evaluated using real-world industrial designs. Summarized, the introduced approach enables a systematic search for errors in ESL designs. Here, the debugging techniques improve and accelerate error detection,

observation, and isolation as well as design understanding. The design and development of digital computer software for distributed concurrent programming environments has increased significantly in the past few years. The presence of remote processors and concurrency greatly complicates the creation, analysis, testing, and debugging of all software produced for these environments. It appears that few tools developed for sequential environments are adequate for debugging software programs in a distributed concurrent environment. The distributed concurrent environment also presents the need for special debugging tools that were not needed for sequential environments. This research will present a new model for debugging programs in a distributed concurrent programming environment. This new model was used to design, develop, and implement an integrated, cooperating set of concurrent debugging tools. The new debugging model and tool set were used in a distributed Concurrent C development environment running under the UNIX® operating system and connected by an Ethernet local area network. Actual results obtained from using the new debugging scheme and integrated debugging tool set to detect, locate, and correct software faults in distributed Concurrent C programs are also presented.

Offers application debugging techniques for Microsoft .NET Framework and Windows, covering topics such as exception monitoring, crash handlers, and multithreaded deadlocks.

Our cells harbor many secrets to a long and healthy life; working with our DNA and restoring it to function is where the secret lies. ?Debugging Human DNA? does not refer to research in biology labs and stem cells studies, but rather it presents a technique to debug DNA inside the human body. Most of us think of DNA as the code biologists study in research labs, but have no clue how DNA affects our daily lives. DNA is the code that regulates every move in our being;

without it, we will not be able to think or move. DNA is no different than the computer binary language; similarities exist between the two languages. DNA is not just a code but also a conducting circuit; the helix positive-negative polarities allows of a two-way communication from point A in the brain to point B anywhere in the body; this can only be achieved electrically. Once we understand this concept, it will be easier to undertake the debugging process of our DNA.

Debugging Embedded and Real-Time Systems: The Art, Science, Technology and Tools of Real-Time System Debugging gives a unique introduction to debugging skills and strategies for embedded and real-time systems. Practically focused, it draws on application notes and white papers written by the companies who create design and debug tools. Debugging Embedded and Real Time Systems presents best practice strategies for debugging real-time systems, through real-life case studies and coverage of specialized tools such as logic analysis, JTAG debuggers and performance analyzers. It follows the traditional design life cycle of an embedded system and points out where defects can be introduced and how to find them and prevent them in future designs. It also studies application performance monitoring, the execution trace recording of individual applications, and other tactics to debug and control individual running applications in the multitasking OS. Suitable for the professional engineer and student, this book is a compendium of best practices based on the literature as well as the author's considerable experience as a tools' developer. Provides a unique reference on Debugging Embedded and Real-Time Systems Presents best practice strategies for debugging real-time systems Written by an author with many years of experience as a tools developer Includes real-life case studies that show how debugging skills can be improved Covers logic analysis, JTAG debuggers and

performance analyzers that are used for designing and debugging embedded systems

Every software developer and IT professional understands the crucial importance of effective debugging. Often, debugging consumes most of a developer's workday, and mastering the required techniques and skills can take a lifetime. In Effective Debugging, Diomidis Spinellis helps experienced programmers accelerate their journey to mastery, by systematically categorizing, explaining, and illustrating the most useful debugging methods, strategies, techniques, and tools. Drawing on more than thirty-five years of experience, Spinellis expands your arsenal of debugging techniques, helping you choose the best approaches for each challenge. He presents vendor-neutral, example-rich advice on general principles, high-level strategies, concrete techniques, high-efficiency tools, creative tricks, and the behavioral traits associated with effective debugging. Spinellis's 66 expert techniques address every facet of debugging and are illustrated with step-by-step instructions and actual code. He addresses the full spectrum of problems that can arise in modern software systems, especially problems caused by complex interactions among components and services running on hosts scattered around the planet. Whether you're debugging isolated runtime errors or catastrophic enterprise system failures, this guide will help you get the job done—more quickly, and with less pain. Key features include High-level strategies and methods for addressing diverse software failures Specific techniques to apply when programming, compiling, and running code Better ways to make the most of your debugger General-purpose skills and tools worth investing in Advanced ideas and techniques for escaping dead-ends and the maze of complexity Advice for making programs easier to debug Specialized approaches for debugging multithreaded,

asynchronous, and embedded code Bug avoidance through improved software design, construction, and management Debugging is crucial to successful software development, but even many experienced programmers find it challenging. Sophisticated debugging tools are available, yet it may be difficult to determine which features are useful in which situations. The Art of Debugging is your guide to making the debugging process more efficient and effective. The Art of Debugging illustrates the use three of the most popular debugging tools on Linux/Unix platforms: GDB, DDD, and Eclipse. The text-command based GDB (the GNU Project Debugger) is included with most distributions. DDD is a popular GUI front end for GDB, while Eclipse provides a complete integrated development environment. In addition to offering specific advice for debugging with each tool, authors Norm Matloff and Pete Salzman cover general strategies for improving the process of finding and fixing coding errors, including how to: –Inspect variables and data structures –Understand segmentation faults and core dumps –Know why your program crashes or throws exceptions –Use features like catchpoints, convenience variables, and artificial arrays –Avoid common debugging pitfalls Real world examples of coding errors help to clarify the authors' guiding principles, and coverage of complex topics like thread, client-server, GUI, and parallel programming debugging will make you even more proficient. You'll also learn how to prevent errors in the first place with text editors, compilers, error reporting, and static code checkers. Whether you dread the thought of debugging your programs or simply want to improve your current debugging efforts, you'll find a valuable ally in The Art of Debugging.

DebuggingThe 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware

ProblemsAMACOM

*Surpasses archaic debugging practices. *Introduces advanced debugger topics such as customization, optimization and extension. *Serves as a valuable resource for developing and deploying rock-solid Perl applications. *There is no direct competition for an advanced and comprehensive debugging book.

Is something not working? Maybe you can debug it! Learn about the codes all around us in Debugging: You Can Fix It! Sing along as you learn to Code It!

A single error in a line of code can cause a computer program to go haywire or stop working entirely. Luckily coding has a process for dealing with errors: debugging. Debugging consists of finding and fixing errors in code. You don't have to work as a coder to develop your debugging skills, though. The simple activities in this book help readers develop their logic skills and an eagle eye for spotting errors, both of which are indispensible in coding and helpful in many other areas as well.

The First In-Depth, Real-World, Insider's Guide to Powerful Windows Debugging For Windows developers, few tasks are more challenging than debugging–-or more crucial. Reliable and realistic information about Windows debugging has always been scarce. Now, with over 15 years of experience two of Microsoft's system-level developers present a thorough and practical guide to Windows debugging ever written. Mario Hewardt and Daniel Pravat cover debugging throughout the entire application lifecycle and show how to make the most of the tools currently available–-including Microsoft's powerful native debuggers and third-party solutions. To

help you find real solutions fast, this book is organized around real-world debugging scenarios. Hewardt and Pravat use detailed code examples to illuminate the complex debugging challenges professional developers actually face. From core Windows operating system concepts to security, Windows® Vista™ and 64-bit debugging, they address emerging topics head-on–and nothing is ever oversimplified or glossed over!

This book describes an approach and supporting infrastructure to facilitate debugging the silicon implementation of a System-on-Chip (SOC), allowing its associated product to be introduced into the market more quickly. Readers learn step-by-step the key requirements for debugging a modern, silicon SOC implementation, nine factors that complicate this debugging task, and a new debug approach that addresses these requirements and complicating factors. The authors' novel communication-centric, scan-based, abstraction-based, run/stop-based (CSAR) debug approach is discussed in detail, showing how it helps to meet debug requirements and address the nine, previously identified factors that complicate debugging silicon implementations of SOCs. The authors also derive the debug infrastructure requirements to support debugging of a silicon implementation of an SOC with their CSAR debug approach. This debug infrastructure consists of a generic on-chip debug architecture, a configurable automated design-for-debug flow to be used during the design of an SOC, and customizable off-chip debugger software. Coverage includes an evaluation of the efficiency and effectiveness of the CSAR approach and its supporting

infrastructure, using six industrial SOCs and an illustrative, example SOC model. The authors also quantify the hardware cost and design effort to support their approach.

“Mario Hewardt’s Advanced .NET Debugging is an excellent resource for both beginner and experienced developers working with .NET. The book is also packed with many debugging tips and discussions of CLR internals, which will benefit developers architecting software.” –Jeffrey Richter, consultant, trainer, and author at Wintellect “Mario has done it again. His Advanced Windows Debugging (coauthored with Daniel Pravat) is an invaluable resource for native code debugging, and Advanced .NET Debugging achieves the same quality, clarity, and breadth to make it just as invaluable for .NET debugging.” –Mark Russinovich, Technical Fellow, Microsoft Corporation The Only Complete, Practical Guide to Fixing the Toughest .NET Bugs Advanced .NET Debugging is the first focused, pragmatic guide to tracking down today’s most complex and challenging .NET application bugs. It is the only book to focus entirely on using powerful native debugging tools, including WinDBG, NTSD, and CDB, to debug .NET applications. Using these tools, author Mario Hewardt explains how to identify the real root causes of problems—far more quickly than you ever could with other debuggers. Hewardt first introduces the key concepts needed to successfully use .NET’s native debuggers. Next, he turns to sophisticated debugging techniques, using real-world examples that demonstrate many common C# programming errors. This book

enables you to Make practical use of postmortem debugging, including PowerDBG and other “power tools” Understand the debugging details and implications of the new .NET CLR 4.0 Master and successfully use Debugging Tools for Windows, as well as SOS, SOSEX, CLR Profiler, and other powerful tools Gain a deeper, more practical understanding of CLR internals, such as examining thread-specific data, managed heap and garbage collector, interoperability layer, and .NET exceptions Solve difficult synchronization problems, managed heap problems, interoperability problems, and much more Generate and successfully analyze crash dumps A companion web site (advanceddotnetdebugging.com) contains all sample code, examples, and bonus content.

This book offers a comprehensive practical guide to SAP ABAP for ABAP professionals. Part I of this two-part series lays the groundwork with ABAP basics. Readers will learn fundamental methods and procedures for everyday ABAP use— for example, how to download files from SAP directories to workstations. Dive into the SAP Data Dictionary and how it works. Get detailed information on effective debugging techniques and how to use the SAP Debugger. Clarify when it is best to use standard SAP tables vs. Z-tables. Get expert developer tips and tricks including how to navigate ALV grid lists. Understand the documentation programs available to you and how to use them. Obtain useful reference lists of SAP transactions and SAP database tables. By using practical examples, tips, and screenshots, the author brings readers up to speed on the fundamentals of SAP

ABAP. - How to get the most out of SAP ABAP - Guide for understanding and using the SAP Data Dictionary - Beginner and advanced debugging techniques - Expert ABAP development techniques

Debugging has always been a costly part of software development, and many attempts have been made to provide automatic computer support for this task.Automated debugging has seen major develoments over the last decade. Onesuccessful development is algorithmic debugging, which originated in logic programming but was later generalized to concurrent, imperative, and lazy functional languages. Important advances have also been made in knowledge-based program debugging, and in approaches to automated debugging based on static and dynamic program slicing based on dataflow and dependence analysis technology. This is the first collected volume of papers on automated debugging and presents latest developments, tutorial papers, and surveys.

Pearce's book is specifically about debugging in the programming language VB.NET for every application type, from Windows Forms to ASP.NET to XML Web services.

Object relationships in modern software systems are becoming increasingly numerous and complex, and program errors due to violations of object relationships are difficult to detect. Programmers need new tools that allow them to explore objects in a large system more efficiently and to detect broken object relationships instantaneously. Such tools incorporate approaches used in such areas as data visualization, pattern matching and extraction, database querying, active databases, and rule-based programming. The query-based debugging approach developed by the author of this book is another powerful yet efficient tool to be added to the developer's tool chest. Advanced Debugging

Methods presents practice and tools for debugging computer programs. This book proposes new powerful approaches that simplify the daunting task of debugging complex software systems. Although debugging has been addressed in numerous research papers, many of its methods have yet to be explored in a book-length format. This book helps to fill this gap by presenting an overview of existing debugging tools with motivating examples and case studies, as well as presenting new, state-of-the-art debugging methods. Advanced Debugging Methods will be of use to software developers looking for tools to be applied in cutting edge practice; system architects looking at the relationship between software design and debugging; tools and programming language researchers looking for new ideas in run-time tool implementation as well as detailed descriptions of advanced implementations; and university professors and graduate students who will use this book as supplementary reading for graduate courses in programming tools, language implementation, and advanced object-oriented systems. Advanced Debugging Methods is also a handy reference of currently existing debugging methodologies as well as a springboard for cutting-edge research to simplify the difficult task of debugging and to facilitate the development of more robust software systems.

This book provides comprehensive coverage of verification and debugging techniques for embedded software, which is frequently used in safety critical applications (e.g., automotive), where failures are unacceptable. Since the verification of complex systems needs to encompass the verification of both hardware and embedded software modules, this book focuses on verification and debugging approaches for embedded software with hardware dependencies. Coverage includes the entire flow of design, verification and debugging of embedded software and all key

approaches to debugging, dynamic, static, and hybrid verification. This book discusses the current, industrial embedded software verification flow, as well as emerging trends with focus on formal and hybrid verification and debugging approaches.

Use this collection of best practices and tips for assessing the health of a solution. This book provides detailed techniques and instructions to quickly diagnose aspects of your Azure cloud solutions. The initial chapters of this book introduce you to the many facets of Microsoft Azure, explain why and how building for the cloud differs from on-premise development, and outline the need for a comprehensive strategy to debugging and profiling in Azure. You learn the major types of blades (FaaS, SaaS, PaaS, IaaS), how different views can be created for different scenarios, and you will become familiar with the Favorites section, Cost Management & Billing blade, support, and Cloud Shell. You also will know how to leverage Application Insights for application performance management, in order to achieve a seamless cloud development experience. Application Insights, Log Analytics, and database storage topics are covered. The authors further guide you on identity security with Azure AD and continuous delivery with CI and CD covered in detail along with the capabilities of Azure DevOps. And you are exposed to external tooling and trouble shooting in a production environment. After reading this book, you will be able to apply methods to key Azure services, including App Service (Web Apps, Function Apps, and Logic Apps), Cloud Services, Azure Container Service, Azure Active Directory, Azure Storage, Azure SQL Database, Cosmos DB, Log Analytics, and many more. What You Will Learn Debug and manage the performance of your applications Leverage Application Insights for application performance management Extend and automate CI/CD with the help of various build tools, including

Azure DevOps, TeamCity, and Cake bootstrapper Who This Book Is For Application developers, designers, and DevOps personnel who want to find a one-stop shop in best practices for managing their application's performance in the cloud and for debugging the issues accordingly

The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to: * Understand the system: how perceiving the ""roadmap"" can hasten your journey * Quit thinking and look: when hands-on investigation can't be avoided * Isolate critical factors: why changing one element at a time can be an essential tool * Keep an audit trail: how keeping a record of the debugging process can win the day The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-

detection war stories, the book shows readers how to: * Understand the system: how perceiving the ""roadmap"" can hasten your journey * Quit thinking and look: when hands-on investigation can't be avoided * Isolate critical factors: why changing one element at a time can be an essential tool * Keep an audit trail: how keeping a record of the debugging process can win the day The rules of battle for tracking down -- and eliminating -- hardware and software bugs. When the pressure is on to root out an elusive software or hardware glitch, what's needed is a cool head courtesy of a set of rules guaranteed to work on any system, in any circumstance. Written in a frank but engaging style, Debugging provides simple, foolproof principles guaranteed to help find any bug quickly. This book makes those shelves of application-specific debugging books (on C++, Perl, Java, etc.) obsolete. It changes the way readers think about debugging, making those pesky problems suddenly much easier to find and fix. Illustrating the rules with real-life bug-detection war stories, the book shows readers how to: * Understand the system: how perceiving the ""roadmap"" can hasten your journey * Quit thinking and look: when hands-on investigation can't be avoided * Isolate critical factors: why changing one element at a time can be an essential tool * Keep an audit trail: how keeping a record of the debugging process can win the day Debugging Linux Systems discusses the main tools available today to debug 2.6 Linux Kernels. We start by exploring the seemingly esoteric operations of the Kernel Debugger (KDB), Kernel GNU DeBugger (KGDB), the plain GNU DeBugger (GDB), and JTAG debuggers. We then investigate Kernel Probes, a feature that lets you intrude into a kernel function and extract debug information or apply a medicated patch. Analyzing a crash dump can yield clues for postmortem analysis of kernel crashes or hangs, so we take a look at Kdump, a serviceability tool that collects a system dump after

spawning a new kernel. Profiling points you to code regions that burn more CPU cycles, so we learn to use the OProfile kernel profiler and the gprof application profiler to sense the presence of code bottlenecks. Because tracing provides insight into behavioral problems that manifest during interactions between different code modules, we delve into the Linux Trace Toolkit, a system designed for high-volume trace capture. The section “Debugging Embedded Linux” takes a tour of the I/O interfaces commonly found on embedded hardware, such as flash memory, serial port, PCMCIA, Secure Digital media, USB, RTC, audio, video, touch screen, and Bluetooth, and provides pointers to debug the associated device drivers. We also pick up some board-level debugging skills with the help of a case study. The section “Debugging Network Throughput” takes you through some device driver design issues and protocol implementation characteristics that can affect the horsepower of your network interface card. We end the shortcut by examining several options available in the kernel configuration menu that can emit valuable debug information. 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.

When computer programming pioneer Admiral Grace Hopper discovered a moth causing problems in a mechanical computer, the term debugging was born. This book explores how fixing programming and hardware problems has developed into a critical process for computer programmers. The text includes case studies and examples of debugging tools. Readers are challenged to review a simple program with a problem, and to locate the bug. This provides young computer programmers the chance to see debugging skills in use. Photographs and sidebars help readers gain a solid comprehension of debugging skills and practices.

Essays discuss the terminology, etymology, and history of key terms, offering a foundation for critical historical studies of games. Even as the field of game studies has flourished, critical historical studies of games have lagged behind other areas of research. Histories have generally been fact-by-fact chronicles; fundamental terms of game design and development, technology, and play have rarely been examined in the context of their historical, etymological, and conceptual underpinnings. This volume attempts to “debug” the flawed historiography of video games. It offers original essays on key concepts in game studies, arranged as in a lexicon—from “Amusement Arcade” to “Embodiment” and “Game Art” to “Simulation” and “World Building.” Written by scholars and practitioners from a variety of disciplines, including game development, curatorship, media archaeology, cultural studies, and technology

studies, the essays offer a series of distinctive critical "takes" on historical topics. The majority of essays look at game history from the outside in; some take deep dives into the histories of play and simulation to provide context for the development of electronic and digital games; others take on such technological components of games as code and audio. Not all essays are history or historical etymology—there is an analysis of game design, and a discussion of intellectual property—but they nonetheless raise questions for historians to consider. Taken together, the essays offer a foundation for the emerging study of game history. Contributors Marcelo Aranda, Brooke Belisle, Caetlin Benson-Allott, Stephanie Boluk, Jennifer deWinter, J. P. Dyson, Kate Edwards, Mary Flanagan, Jacob Gaboury, William Gibbons, Raiford Guins, Erkki Huhtamo, Don Ihde, Jon Ippolito, Katherine Isbister, Mikael Jakobsson, Steven E. Jones, Jesper Juul, Eric Kaltman, Matthew G. Kirschenbaum, Carly A. Kocurek, Peter Krapp, Patrick LeMieux, Henry Lowood, Esther MacCallum-Stewart, Ken S. McAllister, Nick Monfort, David Myers, James Newman, Jenna Ng, Michael Nitsche, Laine Nooney, Hector Postigo, Jas Purewal, Reneé H. Reynolds, Judd Ethan Ruggill, Marie-Laure Ryan, Katie Salen Tekinba?, Anastasia Salter, Mark Sample, Bobby Schweizer, John Sharp, Miguel Sicart, Rebecca Elisabeth Skinner, Melanie Swalwell, David Thomas, Samuel Tobin, Emma Witkowski, Mark J.P. Wolf

Fiddler is a Web Debugging Proxy platform that monitors and modifies web traffic. This freeware tool enables developers, testers, and enthusiasts to inspect traffic, set

breakpoints, and "fiddle" with incoming or outgoing data. Fiddler includes powerful event-based scripting, and can be extended using any .NET language. FiddlerCore, the core proxy engine underlying Fiddler, is available to integrate into any .NET application. In this book, you'll learn to fully exploit the power of Fiddler to debug traffic from virtually any web-related application, including Internet Explorer, Google Chrome, Apple Safari, Mozilla Firefox, Opera, and thousands more. You'll see how to debug HTTPS traffic, and use Fiddler with popular devices like iPhone/iPod/iPad, Windows Phone, and others. After exploring the hundreds of built-in features, you'll learn to extend Fiddler using the FiddlerScript engine or build your own applications atop the FiddlerCore class library.

Goodyear brings considerable expertise from his web site consulting work for such notable clients as Pricewaterhouse Coopers, Arthur Andersen, and the Home Shopping Network. He fills an information void by covering debugging for either ASP or ASP.NET. By relating numerous examples of real-world problems encountered and their coding solutions, this content will save programmers many hours and dollars.