Software Engineering For Embedded Systems Methods, Practical Techniques And Applications Expert Guide | 1ac3b33ecd6510f4a4995ea50a033696

This is a textbook for graduate and final-year-undergraduate computer-science and electrical-engineering students interested in the hardware and software aspects of embedded and cyberphysical systems design. It is comprehensive and self-contained, covering the basics from the hardware to the software, with particular emphasis placed on software. It is intended to serve as a textbook for courses in embedded systems and cyberphysical systems. The book is divided into three parts: The first part covers the fundamentals of embedded systems, including a brief introduction to computer architecture and operating systems. The second part covers the design of embedded systems, including microprocessor design, embedded Linux, and real-time operating systems. The final part covers applications of embedded systems, including examples from the automotive, aerospace, and consumer electronics industries. The book is intended to be used as a textbook for courses in embedded systems, and is also suitable for self-study by practitioners in the field.
programmers write the most efficient code possible, whether that is measured in processor cycles, memory, or power. It starts with an introduction to using the tool chain, covers the importance of knowing the embedded architecture before optimization, then moves on to cover a wide range of optimization techniques. Techniques are presented which are valid on all programmable and non-programmable systems, including loop transformations, the use of code generation, memory management, and much more. The book also includes case studies that demonstrate how these techniques can be implemented, the rationale for choices made, and design guidelines and trade-offs. This textbook introduces the concept of embedded systems with exercises using Arduino Uno. It is intended for advanced undergraduate and graduate students in computer science, computer engineering, and electrical engineering programs. It contains a balanced discussion on both hardware and software related to embedded systems, with a focus on co-design aspects. Embedded systems have applications in Internet-of-Things (IoT), wearables, self-driving cars, drones, cyberphysical systems, and robotics. The hardware chapter discusses various microcontrollers (including popular microcontroller hardware examples), sensors, amplifiers, filters, actuators, wired and wireless communication topologies, schematic and PCB designs, and much more. The software chapter describes OS-less programming, bitmath, polling, interrupt, timer, sleep modes, direct memory access, shared memory, mutex, and smart algorithms, with lots of C-code examples for Arduino Uno. Other topics discussed are prototyping, testing, verification, reliability, optimization, and regulations. Appropriate for courses on embedded systems, microcontrollers, and instrumentation, this textbook teaches budding embedded system programmers practical skills with fun projects to prepare them for industry products. Introduces embedded systems for wearables, Internet-of-Things (IoT), robotics, and other smart devices; Offers a balanced focus on both hardware and software co-design of embedded systems; Includes exercises, tutorials, and answers. The chapter introduces the automotive industry, which is unlike any other, as it is driven by its rigorous planning, architecting, development, testing, validation and verification. The physical task of writing embedded software for automotive applications versus other application areas is not significantly different from other embedded systems, but the key differences are the quality standards which must be followed for any development and test project. To write automotive software effectively, the engineer must be involved in the complex and interwoven environment and be aware of the differences and commonalities between the automotive submarkets. They must be familiar with the applicable quality standards and why such strict quality controls exist, along with how quality is tested and measured, all of which are described in this chapter with examples of the most common practices. This chapter introduces various processes to help software engineers write high-quality, fault-tolerant, interoperable code such as modeling, autocode and advanced trace debugging, supported by the industry standards AUTOSAR and ISO26262, as well as the more traditional standards such as AEC, OBD-II and MISRA. This chapter provides some guidelines that are commonly used in embedded software development. It starts with principles of programming, including readability, testability, and maintainability. The chapter then proceeds with discussing how to start an embedded software project, including considerations for hardware, file organization, and development guidelines. It then moves on to development guidelines, which include the importance of a syntax coding standard. The chapter concludes with descriptions of variables and definitions and how they are typically used in an embedded software project. A recent survey stated that 52% of embedded projects are late by 4-5 months. This book can help get those projects in on-time with design patterns. The author carefully takes into account the special concerns found in designing and developing embedded applications specifically concurrency, communication, speed, and memory usage. Patterns are given in UML (Unified Modeling Language) with examples including ANSI C for direct and practical application to C code. A basic C knowledge is a prerequisite for the book while UML notation and terminology is included. General C programming books do not include discussion of the contraints found within embedded system design. The practical examples give the reader an understanding of the use of UML and OO (Object Oriented) designs in a resource-limited environment. Also included are two chapters on state machines. The beauty of this book is that it can help you today, which Design Patterns within these pages are immediately applicable to your project Addresses embedded system design concerns such as concurrency, communication, and memory usage. Examples contain ANSI C for ease of use with C programming code. The software architecture of embedded computing systems is a depiction of the system as a set of structures that aids in the reasoning and understanding of how the system will behave. Software architecture acts as the blueprint for the system as well as the project development. Architecture is an important necessity for safety-critical systems, such as the systems used in autonomous vehicles, aircraft flight controllers, car braking systems or medical devices), but also to lesser-integrity applications. This chapter introduces the concept of software engineering for embedded systems. It discusses some of the challenges that arise, such as the need to optimize the effectiveness of the available test time and budget demands that pragmatic decisions should be made. To complement this multitude of software test techniques there is a similar plethora of test tools available to automate them. These tools are commonplace in the development of safety-critical applications, but elsewhere not everyone has the budget, time, or indeed a need to purchase or use these tools. Of course, the providers of these tools can help the purchasing teams and every one of them, so how can a limited budget best be allocated? And where no budget exists, how can similar principles be applied to provide confidence that the finished item is of adequate quality? In addressing these issues not only are the concepts behind the techniques presented, but also some “case study” software code examples to drill a little deeper and illustrate how some of them are implemented in practice. Agile software development is a set of software development techniques based on iterative and incremental development, where different requirements and solutions can be provided to the customer for review at reasonably short intervals. Agile development supports adaptive planning, evolutionary development and delivery, and a time-boxed iterative approach. The goal of agile is rapid and flexible response to change. Agile is a conceptual framework which promotes interactions throughout the development cycle. Applying agile to embedded software projects introduces some unique challenges, such as more difficultly effectively testing evolving software, but also the corresponding hardware may not be available in time, less freedom to make changes, due to the fact that the corresponding hardware change may have an unacceptable high cost, and less ability for “learn as you go” approaches, considering the hardware construction may demand a more upfront style of planning and design. This chapter will introduce agile software development and show how to apply these techniques to an embedded system. Authors of two of the leading authorities in the field, this guide offers readers the knowledge and skills needed to achieve proficiency with embedded software. Linux continues to grow as an operating system of choice in many embedded systems such as networking, wireless, and base stations. In this chapter we look at possible uses of
Read Free Software Engineering For Embedded Systems Methods Practical Techniques And
Read Free Software Engineering For Embedded Systems Methods Practical Techniques And Labs Of Varying Lengths And Levels Of Difficulty And Supporting Website Including All Sample Codes Software Tools And Links To Additional Online References Software Engineering For Embedded Systems: Methods, Practical Techniques, and Applications, Second Edition provides the techniques and technologies in software engineering to optimally design and implement an embedded system. Written by experts with a solution focus, this encyclopedic reference gives an indispensable aid to tackling the day-to-day problems encountered when using software engineering methods to develop embedded systems. New sections cover peripheral programming, Internet of things, security and cryptography, networking and packet processing, and hands on labs. Users will learn about the principles of good architecture for an embedded system, design practices, details on principles, and much more. Provides a roadmap of key problems/issues and references to their solution in the text Reviews core methods and how to apply them Examples demonstrating timeless implementation details Short and to-the-point case studies show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs This chapter provides information to successfully organize and manage any embedded software project or program. It introduces quality systems, the OSI model of architectural software into stacks, several software development models and ways in which teams may be organized, and the integration of testing and simulation with schedule, costs, incremental software development, resources, quality, and customer satisfaction fully addresses all the work and activities of any project or program. The natural progression of software development from its concept through its life-cycle until release is discussed. Tools are presented for successful planning and execution of resource management, risk management, problem solving, and the traceability of work extending from requirements to respective engineering responses to testing against hardware specifications. Embedded systems often have one or more real-time requirements. The complexity of modern embedded software systems requires a systematic approach for achieving these performance targets. An ad hoc process can lead to missed deadlines, poorly performing systems and cancelled projects. There is a maturity required to define, manage, and operate on multiple real-time performance requirements. Software performance engineering (SPE) is a discipline within the broader systems engineering area that can improve the maturity of the performance engineering process. SPE is a systematic, quantitative approach to constructing software systems that meet performance objectives. SPE is a software-oriented approach; it focuses on architecture, design, and implementation choices. It focuses on the activities, techniques, and deliverables that are applied at every phase of the embedded software development life-cycle, especially responsiveness and scalability, to ensure software is being architected and implemented to meet the performance requirements. In the first part of the second edition, the authors present an introduction to real-time systems and embedded systems. This chapter explains what a real-time kernel is and what services it provides the product developer, and explains some of the internals of a kernel. A kernel is a component of an RTOS. In this chapter, we'll look at task management, interrupt handling, scheduling, context switching, time management, resource management, message passing, priority inversions and much more. Real-time Operating Systems: Methodologies, Principles, and Applications, Second Edition describes the techniques and technologies in software engineering to optimally design and implement an embedded system. Written by experts with a solution focus, this encyclopedic reference gives an indispensable aid to tackling the day-to-day problems encountered when using software engineering methods to develop embedded systems. New sections cover peripheral programming, Internet of things, security and cryptography, networking and packet processing, and hands on labs. Users will learn about the principles of good architecture for an embedded system, design practices, details on principles, and much more. Provides a roadmap of key problems/issues and references to their solution in the text Reviews core methods and how to apply them Examples demonstrating timeless implementation details Short and to-the-point case studies show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs. This book integrates new ideas and topics from real time systems, embedded systems, and software engineering to give a complete picture of the whole process of developing software for real-time embedded application. You will not only gain a thorough understanding of concepts related to microprocessors, interrupts, and system boot process, appreciating the importance of real-time modeling and scheduling, but you will also learn software engineering practices such as model documentation, model analysis, design patterns, and standard conformance. This book is split into four parts to help you learn the key concept of embedded systems: Part one introduces the development process, and includes two chapters on microprocessors and interrupts—fundamental topics for software engineers; Part two is devoted to modeling of software architectures and Part four covers software implementations, with a focus on POSIX-compliant operating systems. With this book you will learn: The pros and cons of different architectures for embedded systems POSIX real-time extensions, and how to develop POSIX-compliant real time applications How to use real-time UML to document system designs with timing constraints The challenges of real-time UML to cross-domain and inter-task communications (shared memory, messages, data objects, message queues, pipes, signals) How to use kernel objects (e.g. Semaphores, Mutex, Condition variables) to address resource sharing issues in RTOS applications The philosophy underpinning the notion of "resource manager" and how to implement a virtual file system using a resource manager The key principles of real-time scheduling and several key algorithms Implementation of the latest UML standard (UML 2.4) Over 20 design patterns which represent the best practices for reuse in a wide range of embedded systems Development of real-time operating systems which have been tested in GNEA---an operating system widely adopted in industryOne of the most important considerations in the product life-cycle of an embedded project is to understand and optimize the power consumption of the device. Power consumption is highly visible for hand-held devices which require battery power to be able to guarantee certain minimum usage/idle times between recharging. Other main embedded applications, such as medical equipment, test, measurement, media, and wireless base stations, are very sensitive to power as well - due to the need to manage the heat dissipation of increasingly powerful processors, power supply cost, and energy consumption cost - so the fact is that power consumption cannot be overlooked. The responsibility for setting and keeping power requirements often falls on the shoulders of hardware designers, but the software programmer has the ability to provide a large contribution to power optimization. Often, the software engineer has on the power consumption of a device is overlooked or underestimated. The goal of this chapter is to discuss how software can be used to optimize power consumption, starting with the basics of what power consumption consists of, how to properly measure power consumption, and then moving on to techniques...
for minimizing power consumption in software at the algorithmic level, hardware level, and data-flow level. This will include demonstrations of the various techniques and explanations of both how and why certain methods are effective at reducing power so the reader can take and apply this work to their application immediately. When designing an embedded system, special care must be taken when you design the user interface. For simple devices, simple text, command buttons, and LEDs are adequate. For more complex systems, full graphical user interfaces and touch panels are required. User interface design focuses on the following key areas: (a) the design of interfaces between different software components, (b) the design of interfaces between the software and other nonhuman producers and consumers of information, and (c) the design of the interface between a human and the computer. This chapter will focus on the process, guidelines, human factors and techniques required to design an effective user interface. Embedded Systems Architecture is a practical and technical guide to understanding the components that make up an embedded system’s architecture. This book is perfect for those starting out as technical professionals such as engineers, programmers and designers of embedded systems; and also for students of computer science, computer engineering and electrical engineering. It gives a much-needed “big picture” for recently graduated engineers grappling with understanding the design of real-world systems for the first time, and provides professionals with a systems-level picture of the key elements that can go into an embedded design, providing a firm foundation on which to build their skills. Real-world approach to the fundamentals, as well as the design and architecture process, makes this book a popular reference for the daunted or the inexperienced: if in doubt, the answer is in here! Fully updated with new coverage of FPGAs, testing, middleware and the latest programming techniques in C, plus complete source code and sample code, reference designs and tools online make this the complete package. Visit the companion web site at http://booksite.elsevier.com/9780123821966/ for source code, design examples, data sheets and more. A true introductory book, provides a comprehensive get up and running reference for those new to the field, and updating skills: assumes no prior knowledge beyond undergrad level electrical engineering. Addresses the needs of practicing engineers, enabling it to get to the point more directly, and cover more ground. Covers hardware, software and middleware in a single volume. Includes a library of design examples and design tools, plus a complete set of source code and embedded systems design tutorial materials from companion website. Optimization metrics for compiled code are not always measured in resulting execution clock cycles on the target architecture. Consider a modern cellular telephone or wireless device which may download executables over a wireless network connection or backhaul infrastructure. In such cases, it is often advantageous for the compiler to reduce the size of the compiled code which must be downloaded to the wireless device. By reducing the size of the code needed to be downloaded, savings are achieved in terms of bandwidth required for each wireless point of download. Optimization metrics such as the memory system performance of compiled code are other metrics which are often important to developers. These are metrics correlated to the dynamic run-time behavior of not only the compiled code on the target processor, but also the underlying memory system, caches, DRAM and buses, etc. By efficiently arranging the data within the application or, more specifically, the order in which data and corresponding data structures are accessed by the application dynamically at run-time, significant performance improvements can be gained at the memory-system level. In addition, vectorizing compilers can also improve performance due to spatial locality of data when SIMD instruction sets are present and varying memory-system alignment conditions are met. An embedded system is a computer system designed for a specific function within a larger system, and often has one or more real-time computing constraints. It is embedded as part of a larger device which can include hardware and mechanical parts. This is in stark contrast to a general-purpose computer, which is designed to be flexible and meet a wide range of end-user needs. The methods, techniques, and tools for developing software systems that were successfully applied to general purpose computing are not as readily applicable to embedded computing. Software systems running on networks of mobile, embedded devices must exhibit properties that are not always required of more traditional systems such as near-optimal performance, robustness, distribution, dynamism, and mobility. This chapter will examine the key properties of software systems in the embedded, resource-constrained, mobile, and highly distributed world. The applicability of mainstream software engineering methods is assessed and techniques (e.g., software design, component-based development, software architecture, system integration and test) are also discussed in the context of this domain. This chapter will overview embedded and real-time systems. This Expert Guide gives you the techniques and technologies in software engineering to optimally design and implement your embedded system. Written by experts with a solutions focus, this encyclopedic reference gives you an indispensable aid to tackling the day-to-day problems when using software engineering methods to develop your embedded systems. With this book you will learn: The principles of good architecture for an embedded system Design practices to help make your embedded project successful Details on principles that are often a part of embedded systems, including digital signal processing, safety-critical principles, and development processes Techniques for setting up a performance engineering strategy for your embedded system software How to develop user interfaces for embedded systems Strategies for testing and deploying your embedded system, and ensuring quality development processes Practical techniques for optimizing embedded software for performance, memory, and power Advanced guidelines for developing multicore software for embedded systems How to develop embedded software for networking, storage, and automotive segments How to manage the embedded development process Includes contributions from: Frank Schirrmieger, Shelly Grettlein, Bruce Douglass, Erich Stygler, Gary Stringham, Jean Labrosse, Jim Trudeau, Mike Brogjoli, Mark Pitchford, Catalin Dan Udma, Markus Levy, Pete Wilson, Whit Waldo, Inga Harris, Xinxin Yang, Srinivasa Addepalli, Andrew McKay, Mark Kraefling and Robert Oshana. Road map of key problems/issues and references to their solution in the text. Review of core methods in the context of how to apply them. Examples demonstrating timeless implementation details Short and to-the-point case studies show how key ideas can be implemented, the rationale for choices made, and design guidelines and trade-offs. This book provides a good opportunity for software engineering practitioners and researchers to get in sync with the current state-of-the-art and future trends in component-based embedded software research. The book is based on a selective compilation of papers that cover the complete component-based embedded software spectrum, ranging from aspect-oriented techniques to mass customization. Methodology aspects covered by the authors include functional, temporal, and non-functional specification, validation, verification, and component architecture. As tools are a critical success factor in the transfer from academia-generated knowledge to industry-ready technology, an important part of the book is devoted to tools. This state-of-the-art survey contains 16 carefully selected papers organised in topical sections on specification and verification, component compatibility, component architectures, implementation and tool support, as well as non-functional properties.