Have you ever wondered how to efficiently manage your iterative software projects in the presence of changes? If so, Managing Iterative Software Development Projects with tell you how. Software requirements and expectations for software products change constantly owing to external factors such as new technologies in the market, new competitors, and a shift in fundamental economic drivers. All these factors motivate the need for iterative software development, which is software development’s response to a fundamental business problem—delivering a product that has value given constant changes.

However, managing iterative development projects isn’t trivial. As Ian Spence and Kurt Bittner point out in Managing Iterative Software Development Projects, the most important variables shaping iterative software development projects are scope, quality, time, and cost. Furthermore, these variables require that we manage projects in a way that maximizes quality and minimizes time to market and project cost.

Organization

Part 1 of the book (four chapters) introduces iterative project management principles, while Part 2 (seven chapters) discusses efficiently planning and managing iterative projects. Spence and Bittner present a process-driven approach. They begin by defining iterative development (chapter 1), then discuss overall project planning concepts (chapter 6) before finally addressing a scalable approach to managing iterative projects (chapter 10). Each chapter builds nicely on the previous ones, and the information connects well. I enjoyed studying the material sequentially.

The book concludes with three appendices. Appendix A briefly introduces use-case-driven development (the basis for many popular iterative and incremental development methods, such as the Unified Process) and its relationship to iterative development. Appendix B presents reusable outlines, templates, and checklists critical for iterative software development. Appendix C is a practical case study that applies iterative development concepts discussed in the book to an ATM banking system.

Strengthening the text

The authors augment the concepts and discussions with clear examples and practical implications for software development. They concisely summarize each chapter separately to remind readers of key ideas. The text is easy to read and accurate, containing technical insights. Ideas and examples are mostly from the real world and are easy to understand and relate to.

I especially liked the meaningful figures and graphs throughout the entire book, which illustrate the authors’ ideas and are ideal for presenting software engineering concepts. For example, figure 2-8 nicely illustrates that an overemphasis in adding functionality to software will degrade its quality. Spence and Bitt-
ner also discuss and justify the figures in the text, which makes for a great read.

Lessons from other disciplines

The book also has many appealing discussions that appear to have been adapted from other disciplines. For example, Spence and Bittner’s feedback control for iterative projects seems to draw from control theory, and the authors formulate and present their theory in a neat fashion. My thoughts on this topic made me realize that using such ideas can help software engineers reuse concepts that have already proven useful in other disciplines. Furthermore, such concept adaptations will help mature the young software engineering discipline.

An excellent resource

When reading this book, I was reminded of George Eliot’s statement: “Iteration, like friction, is likely to generate heat instead of progress.” This book is an excellent tool for generating progress with your iterative software development project, and it will prevent you from falling into “friction” traps.

The book would be a suitable textbook for an advanced fourth-year undergraduate or a graduate course on iterative software development for software engineering students. You could also use the book as a reference for any software development or software management course. Furthermore, it could be an invaluable resource for researchers and practitioners.

Spence and Bittner’s 2002 book, Use Case Modeling (Addison-Wesley), has already guided a generation of analysts and developers in getting requirements right. Iterative Software Development Projects is another valuable contribution, bringing important new information to the software management and quality engineering body of knowledge. The software industry’s success depends on our developing software management techniques to efficiently build iterative software products.

From To-Be-Determined to Must-Have in One Step

Art Sedighi


Is it just me, or has security taken on a life of its own? It has always been important, but it never was the center of attention—just a “to-be-determined” for any security requirement that came across a software designer or network engineer’s desk. These days, it’s a must-have, and it’s our job as software architects and developers to make this requirement a reality.

Maybe the topic of security is so vast because it must fulfill many different requirements. Or perhaps it’s because the topic is still immature. Either way, security is here to stay, and it must be in the forefront of conversation when architecting a new system.

When I started reading Introduction to Network Security, I was struck by Neal Krawetz’s analogy of the computer industry: although “surgeons rarely boast about carving poultry when they were 12, computer programmers who were programming at an early age wear it as a badge of honor.” This couldn’t be truer about how our profession has progressed and expanded in the recent decades.

Krawetz also says that although programming and computer science are self-taught, not everyone learns the profession’s ethical implications. In security, ethical training and technical training must go hand-in-hand. It’s not always about doing the right thing, although “doing things right” is important. Authors usually cover ethical connotations near the end of a book or in an appendix, if at all. Krawetz addresses ethics at the beginning, showing his deep understanding of the topic’s importance.

The book takes a bottom-up approach to network security. It covers theory, of course, but more important is how Krawetz introduces security for each OSI (open systems interconnection) stack layer. You might be interested in the physical layer (wireless security) or simply network configuration with a demilitarized zone, and the organization helps you focus on topics individually. Next, Krawetz covers the Data Link layer and its sublayers, followed by the Network and Transport layers and so on. For each layer, Krawetz first covers the basics (for example, what TCP is, how it works, its packet format, and so forth). He then covers the vulnerabilities of each layer of the stack. This is an atypical approach for a security book, but it gives readers a complete view of both networking and security. For example, denial of service for the Transport layer is different from a DoS attack for the Network layer. Breaking down the topic into sections lets readers grasp the complexities involved in security and threats.

Each chapter contains a general risks section where Krawetz discusses how to reduce each layer’s risks and secure each layer. Throughout the book, he covers the Secure Sockets Layer, Secure Shell, Domain Name System, Simple Mail Transfer Protocol, HTTP/S, Wireless Application Protocol, and other protocols.

Introduction to Network Security is a great book. It covers networking and security—both complex topics—in one breath, breaking down how they relate to make it easy for readers to follow.

Art Sedighi is the CTO and founder of SoftModule. He’s also on the IEEE Software editorial board. Contact him at sediga@alum.rpi.edu.