Books

Doing IT Right: Technology, Business, and Risk of Computing, Harold Lorin, Manning Publications Co., Greenwich, CT, 1996. 402 pp. (ISBN 1-884777-09-0).

Doing IT Right is a timely book that takes a dead aim at what Hal Lorin calls the "empty center between management and technologists" in IT (information technology) shops. This center is currently dangerously empty at a time when most organizations are floundering in "architectural vacuums." The computer industry is in desperate need of IT practitioners with skills to effectively create, manage, and deploy information systems that take advantage of the new distributed technology. The development laboratories need the IT catchers, and vice versa. So if you're an IT professional with some time on your hands, this is one key book that you'll want to read.

The book starts by describing the chaos most of us face when we attempt to make distributed systems work at the intergalactic level: How do we make the disparate pieces work together? Where do we start? Is there a reliable foundation we can build on? These are some of the tough questions we constantly face in our working lives. Of course, the answers only get tougher with every new wave of technology that gets thrown at us. Just when we think we know how to build systems with clients and database servers, we get hit with three-tiered architectures, TP (transaction processing) monitors, the Internet, and now distributed objects. There seems to be no end in sight for the weary IT professional.

If chaos gives you ulcers, then you'll find Hal Lorin's book to be a gentle antidote. It will guide you through technology labyrinths and uncharted terrain. From the very start, you'll get the feeling that you're in the hands of a guide who is intimately familiar with the terrain and has been there before. The book even guides you through the Wittgensteinian problem of "knowing what you do not know." It covers a broad range of useful topics—including operating system choices, communication protocols, client/server computing, distributed system standards, project planning, tool selection, and how to calculate the real costs of system ownership.

This book covers more ground than most, which should not come as a surprise if you are familiar with its author, Hal Lorin. Hal—for those of you who do not know him—is one of the early distributed systems visionaries. In the early 1970s, Hal Lorin and his colleague James Martin declared that they had seen the future of computing, and it was fully distributed. At that time, they were both affiliated with the Systems Research Institute (SRI)—an IBM think tank—where they wrote, lectured, and experimented with distributed computing.

In the very early 1980s, I had the good fortune to attend a ten-week SRI seminar taught by Hal Lorin and others. It was an intense and uplifting experience. It gave me a chance to reevaluate what I was doing and where I was going. At the time, the vision of a PC-centric client/server world was almost heresy—PCs were just toys, nobody would dream of using them to create mission-critical systems. I did, and was instantly branded a heretic. During my heretic years, Hal's books, lecture notes, and occasional Stanford classes became the beacons I used to navigate the uncharted waters of developing client/server systems. I survived and was able to coauthor a few books of my own on the subject.

So what do SRI and Hal's course have to do with this book? SRI is no longer in existence, and you probably don't have ten weeks to spare. So the next best thing to a live SRI course is to read Hal's new book. It gave me a jolt similar to the ones I got from the SRI classes. Hal comes to life in his books, just as he does in his lectures. You won't get bored. The book is filled with vintage Hal insights and jokes; it seems Hal only gets better with time.

So the good news is that your guide to *Doing IT Right* is one of the real *bona fide* visionaries of the dis-

tributed computing field. During these chaotic times we need all the 20/20 vision we can get. Even ex-heretics, like myself, are amazed at how fast client/server technology grew. It has become the new computing paradigm. In a few short years, we have gone from being heretics to becoming mature members of the computing establishment. Sadly, we will repeat the mistakes of history if we don't learn from them. Again, this is an area where Hal's new book comes in handy. He does very well when it comes to crossing the great generational divide. Hal understands the old and the new paradigms of computing and can skillfully translate between them.

So what's missing from this book? Unfortunately, it does not fully cover some important areas that could benefit from Hal's insights—including distributed objects, components, data warehousing, TP monitors, groupware, and systems management. Perhaps Hal left some of this material for *Doing IT Right, Part 2.* You can vote for a sequel by buying this book.

This book lives up to its title. It teaches us something about doing IT right. The book contains 14 semi-autonomous chapters. Read the areas that interest you first, then come back for more. It is best to absorb Hal's work in small increments, with time between to ponder his insights.

Robert M. Orfali
Object Technology Support and Consulting
IBM San Jose
California

The Essential Distributed Objects Survival Guide, Robert Orfali, Dan Harkey, and Jeri Edwards, John Wiley & Sons, Inc., New York, 1996. 604 pp. (ISBN 0-471-12993-3).

There is nothing more pleasing than reading a book by informed and engaged authors who have a point of view, can distinguish between facts and their own vision, and warn you when they have gotten on the soap box.

This is a genial, intelligent, complete book with an attitude. The successor to the *Client/Server Survival Guide* by the same authors, this book establishes objects as a key tool for client/server technology and discusses the emerging technologies and issues. It is essentially an extraordinarily complete and informing survey of the world of object and component models—SOM/DSOM vs COM, and OLE** vs CORBA** (or, System Object Model/Distributed SOM vs Component Object Model**, and Object Linking and Embedding vs Common Object Request Broker Architecture). However, the authors do not fear expressing informed viewpoints to the extent that they are helpful. There is a fast path through the material for managers and other casual readers, but the entire book is necessary for anyone who wishes to be considered an intelligent professional in this time.

There is general consensus in the industry that the next generation of systems will extend the client/server architecture into three-tier client/server systems with increased server-to-server interaction and interdependence. There is difference of opinion about the role of each tier—the nature of the end point appliance, whether the second tier is "shared business objects" or "the data warehouse," and if that matters. There is also general consensus in the industry that distance will disappear as a differentiating force in computer culture, that the Ethernet world and the Internet world will merge into each other.

This book takes up these issues from a distributed object point of view. It builds on the theme of "intergalactic" client/server, but introduces a particular vision of the generation to come. It is an object vision in which object-oriented (OO) concepts exist deeply in the structure of systems, and object-to-object interaction is the basis of distributed computing. Computer models based on structured query language are dismissed and the prediction is made that TP monitors (transaction processing monitors) will become Object Request Broker (ORB) -based. Current groupware middleware, the authors claim, is too proprietary and unscalable to survive the needs of Internet intergalactic computing.

Part 1 of the book proposes and develops this view that the underlying paradigm that will define client/server in the future is the distributed object. Distributed objects offer plug and play components, interoperability, portability, coexistence with legacy applications, and self-managing mobile entities. The next (third) wave of computing, following client/server systems based on database managers (second wave), which followed those based on file systems (first wave), will be based on distributed objects.

In support of this argument the authors offer an elegant tutorial on object concepts and how they mature into concepts of components and plug and play. The need for components derives from the increas-

ing complexity of software and the increasing costs of moving programs from one version to another. *Components* are defined as marketable entities that can be used in unpredictable ways with other components from different sources. They are extended objects with the features of shrink-wrapped software. The authors provide some attributes that components must have to be usable and trustworthy. At the end of the trail of development are "business objects" that enable enterprises to model the behavior of their processes with objects.

Part 2 of the book is a nearly 200-page discussion of CORBA. CORBA is presented as the most important and ambitious middleware project of the industry. Its importance lies in the fact that it defines middleware that offers both a unifying metaphor for older applications and a foundation for future development. It is, in effect, a sort of software bus that allows components to discover each other and interact.

The discussion of CORBA includes the Interface Definition Language that defines a component's interfaces with potential clients and permits objects of various languages to interact with ORB. ORB Object Services are described in the context of the Object Management Group (OMG) architecture. The technical discussion flows freely and easily, and is as readable as anyone could want. It is amusingly illustrated throughout and the text is highlighted by discussions that focus on any doubts you may have about what the authors are saying. Difficulties, disappointments, and delays (these authors are in the middle of the game) are acknowledged, but the convincing argument for the paradigm is sustained. DCE**RPC (Distributed Computing Environment remote procedure call) and CORBA II are faced head-on, and the concerns about the role of DCE are discussed frankly.

The most amazing thing about this almost 200-page treatment is that it "goes down like warm muffins." The narrative style is friendly, informative, and yet complete. It handles necessary complexity as well as any narrative can. At the end of it, the reader feels a comfortable understanding of CORBA and a sense that additional details will come easily into the robust framework the authors have built.

Part 3 brings us into the broad sunlit uplands of business objects and components. The notion of frameworks within collections of objects that interact and cooperate is presented in appropriate detail. The chapters introduce and comment on various *de facto* standards for business objects and frameworks: OpenDoc**, OLE, Taligent, OpenStep, and Newi**. Frameworks are compared to alternative concepts of program construction-OO class libraries and procedural APIs and are found to be particularly useful, because they provide the control flow of an application, minimize how much code has to be written, and reduce maintenance costs dramatically.

A chapter is given to an elegant definition and description of business objects. A splendid discussion of the role of objects in systems management is provided with particular attention to the Tivoli Management Environment** (TME). TME is the *de facto* standard for CORBA ORBs. Another chapter discusses the compound document concept, setting the stage for a description of OpenDoc and other frameworks. Microsoft's OLE is described and its challenge to OMG and CORBA is frankly assessed. Parts 4 and 5 discuss OpenDoc and OLE in considerably greater detail and continue to compare, contrast, and evaluate.

The book ends by returning to its promise that objects are the root technology of the future of client/server. It discusses COM/OLE and CORBA/OpenDoc interoperability; the nature of linking gateways; and concludes with a discussion of the practicalities of working with component suites.

These authors are clearly presenting a point of view based on two premises—that objects will become the underlying building block of software and that the CORBA version of the object world will become the dominant object culture. The first premise is likely right. Despite the disappointments already experienced and the unbearable hype already heard, it is likely that objects represent a software organizing principle that offers the encapsulation, predictability, and modularity we have been aiming for since the beginning of software. The second premise is less sure in a world where Microsoft is Microsoft and many competitors seem to lack the will to resist conforming to Microsoft standards.

Regardless of how the market works out, those who want to be in it will profit enormously from this book.

Harold Lorin
The Manticore Consultancy and
Senior Adjunct Professor,
Hofstra University
New York

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Microsoft Corp., Object Management Group, Open Software Foundation, Inc., Apple Computer, Inc., Integrated Objects, or Tivoli Systems Inc.

Secrets of Software Quality: 40 Innovations from IBM, Craig Kaplan, Ralph Clark, and Victor Tang, McGraw-Hill, Inc., New York, 1995. 383 pp. (ISBN 0-07-911795-3).

Secrets of Software Quality describes 40 innovations that helped one of IBM's software development laboratories, IBM Santa Teresa, achieve unprecedented improvement in the quality of its software products. The innovations described were either invented at IBM Santa Teresa (e.g., the excellence council, the center for software excellence, the council system, high-risk module analysis, online reviews and associated support tools, computer-supported team work spaces), or developed and pioneered by other IBM divisions and subsequently adapted by IBM Santa Teresa (e.g., the defect prevention process, cleanroom methodology, design review and code inspections), or were industry "best practices" (e.g., process benchmarking, rapid prototyping, object-oriented design and programming, statistical approaches—use measurement and analysis to guide improvement effort). According to the authors, these improvement approaches were called "innovations" because "they were new ideas to us when we implemented them."

Secrets of Software Quality is heavily oriented toward the Malcolm Baldrige National Quality Award assessment approach. The authors took a balanced approach that integrates leadership, process, and technology (the "iron triangle") as key ingredients of successful software quality management. Furthermore, based on ranges of Baldrige scores and the experiences of the Santa Teresa lab, the authors derived four stages of quality maturity in an organization: awareness, coping, management, and integration. A nice framework was then formed by examining the activities and progress of the iron triangle elements within each stage of quality maturity. The 40 innovations and the book chapters are organized in accordance with this framework. The last chapter provides advice that organizations can use to identify a set of activities for their quality improvement program. The appendix and the accompanying compact disk offer quick and easy ways to perform a Baldrige-style assessment. Thus, this book will be very useful for companies that intend to pursue the Baldrige award or to use the Baldrige assessment approach for quality improvement.

The authors adopted a good format for describing each innovation. The objective and the description of the approach are first given, followed by a discussion of costs, benefits, and risks; a section on implementation advice is also provided. Often, opinions of pros and cons and implementation pitfalls are presented, allowing the reader to reach his or her own judgment with regard to the effectiveness of the approach.

Of the three ingredients of the iron triangle, leadership innovations come through as the strongest items. Indeed the strong leadership, commitment, perseverance, and persistence of the general manager for quality improvement are mentioned numerous times throughout the book. Many leadership innovations described in the book are those executed by upper management. Among several remarkable approaches is the establishment of the Center for Software Excellence, an implementation of the concept of a software "experience factory" by Victor Basili.

The coverage and discussions on process and technology/tools items are generally good. However, after reading through the book, I did not get the impression of a systematic treatment of the software development process, especially from a software engineering perspective. One of the reasons may be the organizational approach of the book (innovation-oriented, grouped by stage of quality maturity, and rightly so), as opposed to a systematic evaluation of the software development process and its improvement.

For example, how does object-oriented design and programming differ from (or resemble) procedural programming in terms of development process and project management? Aren't rigorous design reviews, code inspections, and early test involvement the basic elements of the waterfall development process? Doesn't M. E. Fagan's five-step inspection method—which has been around for two decades—cover all stages from overview to follow-up? Aren't these activities actually "back to the basics" of professional software development? And how about the recent innovations on design reviews and code inspections that appeared in the software engineering literature after Fagan's masterpiece?

Nonetheless, when I overcame the expectations associated with "innovation" with regard to the software development process (apparently the authors use the term loosely) and took the chapters as experience reports about the Santa Teresa team, I truly appreciated the value of the information and discussions.

Perhaps the weak link in the book is that the authors did not quantify the effect of the innovations in concrete, measurable terms. Nor is much specific information given on product or project outcomes (for example, quality level, productivity, cycle time, extent of improvement, etc.) after implementing, for example, such and such, of these innovations. The Baldrige assessment focuses on approach, deployment, and results. The "results" are notably missing in most, if not all of the innovations. In chapter one, impressive results of the Santa Teresa lab in terms of several quality, productivity, and customer satisfaction indicators are presented. However, how and to what extent these 40 innovations contribute to these lab-wide results is a major missing link. Are there other significant variables latent in the equation? Or perhaps, as the authors stated in the preface, "these 40 innovations represent a small fraction of the Santa Teresa team's effort." But then can one rely on these innovations and hope to achieve significant quality improvement like that achieved by Santa Teresa?

Overall, *Secrets of Software Quality* is a welcome addition to modern-day software quality management literature. Its attempt to cover leadership, process, and technology, and to bridge total quality management (TQM) as a management philosophy and actual implementation experiences of the software development process is to be commended. It provides valuable advice for upper management and for companies that intend to pursue quality improvement based on the Baldrige approach. Its discussion of implementation experiences of the software development process also lends useful references and lessons to software development managers, project leaders, and quality professionals.

Stephen H. Kan
Development Quality and
Process Technology
IBM Rochester
Minnesota

Note—The books reviewed are those the Editor thinks might be of interest to our readers. The reviews express the opinions of the reviewers.