Editor: Warren Keuffel ■ wkeuffel@computer.org

Adaptive Practices for Software Excellence

Deependra Moitra

Adaptive Software Development: A Collaborative Approach to Managing Complex Systems *by James A. Highsmith III, Dorset House Publishing, New York, 2000, 0-932 633-40-4, 389 pp., US $44.95.*

If fast-paced, high requirements volatility, and uncertainty characterize your development environment, and you can't figure out how to succeed, then *Adaptive Software Development* by James Highsmith can help. It offers an innovative approach grounded in the theory of complex adaptive systems, and it's both well researched and timely.

Its real attraction, however, is that it provides a holistic approach to software development and management—*adaptively*, of course—and covers the crucial aspects of learning in software production environments.

Today, developers struggle to cope with the conflicting demands of producing quality software and delivering it in Internet time. As a result, that body of knowledge commonly described as software development's "best practices" is continually evolving to meet the needs of its users. The first two reviewers discuss books whose authors, Jim Highsmith and Kent Beck, address the conundrum of how to reconcile quality and speed. As with all new development concepts, you'll find more enthusiasm and anecdotal success stories than measurable supporting data, but if nothing else, these books—and our reviewers' evaluation of them—deserve your attention for challenging the establishment.

Rounding out our trio of reviews is an evaluation of an important new book from Ron Kenett and Emanuel Baker that will be of interest to those developing software quality programs.

And now a plea from your Bookshelf editor: if you have particular expertise in the area of usability and would like to contribute a review for an upcoming issue focused on that topic, please contact me by 15 August 2000.

—Warren Keuffel, Bookshelf Editor

Goals and Content

Highsmith begins by stating that his four goals are to

- provide an alternative to the popularly held belief that optimization is the only solution to increasingly complex problems;
- offer an approach based on adaptive principles and introduce associated frameworks and models for implementation;
- show the necessity and benefits of power of collaboration to succeed; and
- introduce adaptive management.

In the first two chapters, Highsmith introduces complex adaptive systems and the various components and models of adaptive software development. The next four chapters explain the various components of adaptive development—speculating on direction, adaptive cycle planning, collaboration, and learning—and address various aspects of adaptive project planning and execution. The treatment in these chapters is powerful and though-provoking, though readers familiar with methodologies such as the Dynamic System Development Methodology (www.dsdm.org) will find some of the ideas presented familiar to them; nevertheless, certain sections are especially interesting and hence noteworthy.

Chapter 3 presents the concept of a product mission profile, which helps identify the primary driver for the product's success in the market and set the project's focus. The chapter includes a product mission profile matrix with four project variables (scope, resources,

0740-7459/00/$10.00 © 2000 IEEE

quality, and schedule), which can help readers establish priority in their own projects. Similarly, Chapter 4's detailed explanation of adaptive cycles and thorough guidance on adaptive planning techniques is very useful. A hypothetical cycle-planning example is an added attraction. In addition, I particularly liked the discussion in Chapter 6 on *customer focus group* reviews. Highsmith explains CFG reviews, highlights their usefulness, builds a strong case for conducting them, and offers guidance on how to successfully conduct them.

Chapters 7 through 11 focus on the cultural infrastructure needed to perform adaptive development and describe in detail adaptive management practices. Chapter 9 introduces yet another important practice—workstate life-cycle management. Highsmith contrasts this with the traditional workflow-based approach and explains why the workstate-oriented mindset is suitable for fast-paced development.

A Few Glitches

Highsmith has produced an excellent book that will surely help those who peruse it seriously. It should help software engineers, managers, trainers, and consultants alike. Experienced software professionals will particularly be able to relate to Highsmith's approach.

However, some details presented in the book seem unnecessary, especially those that are primarily built on the contributions made by others. For example, Chapter 7 spends too much time on material already well-covered in two seminal books, *The Innovator's Dilemma* (Clayton Christensen, Harvard Business School Press, 1997) and *Inside the Tornado* (Geoffrey Moore, HarperBusiness, 1995); references to these would have sufficed. Also, many chapters require a philosophical bent of mind; readers must considerably stretch their minds to comprehend and digest the material.

In many chapters, the analogies to mountaineering are extreme and, at times, boring. In addition, many paragraphs are accurate in their own ways but do not connect well with the following paragraphs or even the book's theme. Furthermore, Highsmith should have included success stories or data based on real projects that have benefited from the adaptive development approach.

Interestingly, Highsmith, like many others, often attempts to wage a war against the CMM. However, he appears to have his own understanding of and assumptions about it. I found his comparison to and argument against the CMM unnecessary and not well supported.

A Solid Investment

Overall, *Adaptive Software Development* is a useful contribution to the changing profession of software engineering. It attempts to integrate markets, organization, development, and customers and offers a thought-provoking framework of concepts, practices, and guidelines to sail through today's volatile software product development environment. I am sure the software soldiers and their commanders operating in turbulent times stand to gain from this book.

Deependra Moitra is the General Manager of Quality at Lucent Technologies, India Product Realization Center. Contact him at d.moitra@computer.org.

Taking Common Sense to the Extreme

Diomidis Spinellis

eXtreme Programming Explained: Embrace Change *by Kent Beck, Addison Wesley Longman, 2000, 0-201-61641-6, 190 pp., $29.95.*

Many software projects face vague and changing requirements: a pressing time schedule, stringent quality objectives, high staff turnover rates, and demands to deliver value at a low cost. Extreme programming, as described in *eXtreme Programming Explained*, by Kent Beck, challenges many conventional software engineering doctrines and, turning development on its head, offers a lightweight methodology for effectively delivering software while controlling time, cost, quality, and, refreshingly, the project's scope.

Eliminating the Elaborate

The theoretical basis for XP is an options-based pricing model of software development. Because software is developed in an environment where technological, business, and human risks abound, investing for the future through careful architectural design, elaborate code structures that support all possible future requirements, and matching documentation can be a waste of effort. Changing requirements or technology might render parts of the system obsolete.

Kent Beck's insight lies in discounting elaborate design for future needs in favor of today's simplicity. He suggests deferring important design decisions until a particular feature is needed. According to Beck, you should develop the most important features first, rapidly providing the customer with the functionality required and minimizing the risk of total project failure. As code evolves, you can always refactor it in response to new needs. Surprisingly, Beck takes for granted that current state-of-the-art technology and programmer skills will support these continuous design changes.

Beck's XP

XP takes a holistic, value-driven view of the development process, focusing on four key values: *communica-*

tion, *simplicity*, *feedback*, and *courage*. It encourages communication by having the developers collectively own all the code and work in pairs on a single machine. XP's simple design evolves through constant refactoring—guided by a suitable metaphor and implemented in accordance to common coding standards—and obviates the need for extensive documentation. XP encourages programmers to develop only the functionality needed and to plan on changing the design to accommodate new features.

To provide feedback during development, XP programmers write unit tests before coding activities and run tests after design changes or integration. They also have a customer representative on site who can write function tests. Finally, courage complements the other three values by letting team members throw code away or improve system-level design and by encouraging managers to embrace change.

A Guide to Reading the Book

Beck devotes separate chapters to management, planning, development, design, and testing; however, he fails to discuss in depth several practical implementation aspects. A complete case study might have helped readers understand how to plan a large project and the pragmatics of pair programming. More importantly, a case study could teach readers how unit and functional tests evolve in a large or GUI-based system. Beck takes for granted the use of object-oriented technology (including object-oriented databases) when discussing design refactoring and thus fails to give this important implementation constraint and its implications the attention it deserves.

On a more positive note, Beck offers advice on how to adopt XP (by gradually implementing XP practices) and retrofit it into existing projects (which is difficult but not impossible). He also outlines the life cycle of an XP project and discusses various, roles, problems, limitations, and exploitation issues. He frankly admits XP's limitations, and I recommend that skeptical readers read the respective chapter first to avoid distracting themselves by trying to apply XP to areas for which it is not suited.

Beck advises against using XP in an environment whose culture would not tolerate it: in large teams (a 10-person team is probably the upper limit), applications where changes incur large overheads, systems with long compile or test cycles, applications that do not allow easy testing, and physical environments that separate people (XP teams are encouraged to work cooperatively in a single room; it is unclear how constant interruptions and pair working allow deliberate concentration). Again, I advise skeptical readers to read the penultimate chapter early on—it describes how to apply XP to existing contractual work practices, albeit not for open source development.

Although not always entirely convincing, Beck proposes how to adapt fixed price, outsourcing, and time and materials contracts for XP (not the other way around). An eclectic annotated bibliography (which will doubtlessly provide incentives for more reading), a glossary, and an index (with too many second-level entries and too few first-level ones) complement a generally well-structured and well-written book.

Overall, A Beneficial Read

XP is clearly not for everyone. However, most projects currently being developed under the pressures of Internet time can surely benefit from its approach, and they are not the only candidates. I have often found myself following parts of the XP approach while programming—I'm glad that Beck formalized it into one coherent methodology encompassing the complete software lifecycle. I will certainly try to experiment with XP in a team development context, and I am sure I will not be alone.

Diomidis Spinellis is an assistant professor in the Department of Information and Communication Systems at the University of the Aegean. Contact him at dspin@aegean.gr

Software Process Quality: Flying Coach

Robert Bruce Kelsey

Software Process Quality: Management and Control *by Ron S. Kenett and Emanuel R. Baker, Marcel Dekker, New York, 1999, 0-8247-1733-3, 241 pp., $150.*

When you book an economy coach flight across the country for $150, you know what to expect. You'll cover a lot of ground but your sightseeing will be restricted to what can be seen from 30,000 feet. And, if you're lucky, you'll get a snack that will stave off hypoglycemia until you land. But if you plunk down $150 for a book, even at the inflated prices in airport bookstores, you expect to walk away with an indispensable, perhaps even seminal work in the field, not an in-flight magazine. Caveat emptor—*Software Process Quality* falls somewhere in between.

The View from 30,000 Feet

The authors derive their material from graduate courses Ron Kenett delivered at Tel Aviv University. As a primary text in a survey course, it has much to recommend it. It describes the components of a software quality program, discusses the various standards that you can use in developing such a program, and at least touches on all aspects of development from

requirements management to configuration management to metrication for continuous improvement. What sets this work apart from the typical "how to" texts available is the authors' commitment to present their practical techniques in the context of a quality management program.

The first third of the book is devoted to describing the tenets of quality management and the benefits of using continuous improvement models such as the CMM. Echoes of Walter Shewhart, J.M. Juran, Kaoru Ishikawa, and W. Edwards Demming abound, and some of the material reads like a synopsis of the study guides for the Certified Quality Manager exam from the American Society of Quality (in which Kenett is a senior member). The authors begin with an explanation of the Quality Journey and why quality councils, strategic quality planning, and process management are crucial to its success. Next, they describe and compare ISO 9000/9000-3, CMM 1.1, and SPICE as continuous improvement models. With the rationale and models in place, the authors turn their attention to the requirements for process control: quality measurements, quality attributes for software and support for documentation, software reliability, and reviews and inspections.

An In-flight Snack, Not a Meal

Software Process Quality is the latest release in the *Computer-Aided Engineering* series from Marcel Dekker, Inc. The intended audience for the series, according to series editor Mark Coticchia, includes students, educators, and practitioners. Unfortunately, a book that meets the needs of students usually falls short of meeting the needs of practitioners. Kenett and Baker's effort is no exception. Their stated goal is to provide a "methodology for establishing the current status of a software development process and laying out a reasoned plan for process improvement" (p. xi). The program they present contains nothing new or terribly controversial, and in that respect a neophyte software quality manager could pick up this book and use it to build a tolerable quality system. The problem isn't what the authors present; it's what they chose to ignore.

Omissions and inconsistencies are particularly troublesome in the chapters on software measurement, software quality, and software reliability. For example, the authors present only raw incidence measures and ignore the various diagnostic measures available. They use lines of code in their defect density measure without explaining how different product types and development platforms can affect the measure. They present four reliability models but never mention the caveats throughout the literature about the conditions under which reliability estimations lose their validity. They spend several pages on how to perform a CMM-based assessment, but give both test methodologies and configuration management short shrift and devote no space to reuse.

Still, the Service is Good

Nevertheless, the topics that Kenett and Baker *do* cover are presented clearly and concisely, making this book a handy introduction to assessments, inspections, reviews, and requirements analysis. If you want to learn how to set up the infrastructure to support a continuous improvement program, *Software Process Quality* makes an excellent travel guide for your Quality Journey. Educators should consider this book a viable candidate for a software quality course as long as they are prepared to make up for some of the text's deficiencies either in lectures or through additional assigned readings. And as a handbook for practitioners, it presents a practicable—but not thoroughly defended or explained—methodology for implementing a cohesive software quality program.

Robert Bruce Kelsey is a member of the IEEE and the American Society for Quality. Contact him at robertbrucek@netscape.net.

IEEE Software

How to Reach Us

Writers
For detailed information on submitting articles, write for our Editorial Guidelines (software@computer.org), or access computer.org/software/author.htm.

Letters to the Editor
Send letters to

Letters Editor
IEEE Software
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
cbaltes@computer.org

Please provide an e-mail address or daytime phone number with your letter.

On the Web
Access computer.org/software for information about *IEEE Software*.

Subscription Change of Address
Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Be sure to specify *IEEE Software*.

Membership Change of Address
Send change-of-address requests for the membership directory to directory.updates@computer.org.

Missing or Damaged Copies
If you are missing an issue or you received a damaged copy, contact membership@computer.org.

Reprints of Articles
For price information or to order reprints, send e-mail to software@computer.org or fax +1 714 821 4010.

Reprint Permission
To obtain permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at whagen@ieee.org.