Editor: Warren Keuffel ■ wkeuffel@computer.org

# Getting Serious about Open Source

**Terry Bollinger**

**Succeeding with Open Source** *by Bernard Golden, Addison-Wesley, 2004, ISBN 0-321-26853-9, 272 pp., US$39.99.*

I like a book that takes on a difficult topic and does its best to address that topic's most worrisome questions. I also like a book where the author starts out not as a zealot with a point to prove or an ax to grind, but as a skeptic who warms to his topic only after immersing himself in it. I'll also give the thumbs-up to a book that "gets real" and provides practical, easy-to-apply advice that's readily accessible to a broad range of software developers, managers, and users. Finally, I'm intrigued by any book to which my overall reaction is, "Wow, a small company that takes this book to heart could end up with some pretty serious competitive advantages." All these describe my reaction to *Succeeding with Open Source,* a fascinating and informative new book by Bernard Golden. Golden devotes much of his book to showing readers how to apply his methods, and his objective approach to a difficult topic is welcome.

## The controversy

The book's controversial topic is certainly no secret—it's right there in the title. Open source or "free" software is a phenomenon that gives software decisions managers nightmarish visions of a freight train barreling down a track, a track to which they and their businesses just happen to be firmly tied. They can see it coming, and as much as they'd like to wish it away or pretend it's no big deal, the fear remains. Will open source software squash their proprietary software business and leave them jobless? Will a group of fanatical free-software zealots who don't care one lick about the right to create and sell new software products push them out of business? Or, as others suggest, is open source simply the next wave of competitive advantage—a freight train that, if you can just free yourself from the proprietary railroad ties that bind and blind you, can carry you leaps and bounds ahead of your less visionary business peers?

In responding to such troublesome visions, Golden appears to take his cue from the following bit of advice from Gene Kranz (played by Ed Harris) in the movie *Apollo 13*: "Let's work the problem, people. Let's not make things worse by guessing."

That's good advice. Wherever Golden got it, I'm glad he took it. I can summarize his pragmatic approach as follows: "Wow, open source looks like a badly underused software resource with great potential. What barriers keep people from using it, and is there some straightforward way to remove those barriers?"

## Facing the fear

The barriers Golden discovered will seem familiar to anyone in software engineering. In short, the major roadblocks to using open source and proprietary software products are the same: immaturity of products and of the processes that create them. Open source products are created using decentralized and volun-



0740-7459/04/$20.00 © 2004 IEEE

tary processes that are so radically different from those of traditional software firms that they engender an almost reflexive barrier of fear. Golden quickly moves beyond this barrier and starts asking familiar and surprisingly productive questions. He notes that just like traditional software, open source software products vary enormously in code quality, support, testing, documentation, and training availability. From this simple starting point, he asks the obvious question, "How do I uncover and evaluate these qualities for open source applications, which use far less centralized development processes than traditional software?"

## Open Source Maturity Model

Golden's approach is familiar. He defines a list of elements to evaluate, explains why these elements are important, and discusses how they differ from proprietary software. He then combines the elements into an Open Source Maturity Model, complete with easy-to-use templates to help users collect the necessary data. Golden's OSMM uses the same concept of maturity that we apply to people: when the chips are down, will the person—or application—be there to help?

Golden's OSMM provides a consistent, persuasive framework that companies can use to evaluate and compare open source opportunities. The OSMM uses five primary elements that are universal to any software product, whether proprietary or open source:

- Product quality
- Availability and quality of support
- Security and testing
- Documentation
- Availability and quality of training

The OSMM then adds four secondary elements that focus more on open source products' and processes' unique aspects, such as whether any traditional companies support the product. These secondary elements include availability of commercial distributions, availability of professional services, press coverage, and analyst coverage.

Golden does not leave the OSMM as a hypothetical model. He demonstrates its use on JBoss, a popular open source implementation of Sun's J2EE Application Server standard. If Golden's book did nothing more than document the resulting evaluation of JBoss, it would have provided a valuable service. By combining a tempting suite of features with an open source origin, JBoss perplexes many decision-makers. Golden's example not only provides useful information about JBoss, but also helps explain why certain aspects of open source evaluation must go beyond traditional measures. Overall, Golden's OSMM points the way for the industry as a whole to begin integrating open source software and methods into mainstream software development and acquisition.

## Catching the train

Golden's premise is that once you get beyond the fear factor, open source software becomes just another way to move ahead competitively. Far from seeing open source as a death knell of proprietary software, Golden views it as a new, complementary resource that helps companies control skyrocketing infrastructure costs and explore new markets more nimbly.

But is this view correct? Does open source really increase competition and build new markets, or is it actually a woolly-clothed wolf that will end up annihilating software innovation for all time?

There's evidence that Golden's view is indeed prescient. For example, Google, now a multibillion-dollar public company, combined the opportunity of open source (all Google servers use GNU/Linux) with proprietary innovation (their famous page-count ranking system). At the other end of the size spectrum, open source is creating a frenzy of feature and price competition for low-cost Web hosting, as you can see by looking for low-cost Web hosting companies at FindMyHosting.com. Finally, leading-edge information technology startup companies are increasingly packaging their innovations as easy-to-add appliances for augmenting the security, storage, or analysis capabilities of legacy networks and facilities. In an informal survey, every such appliance I encountered used GNU/Linux. Clearly, something interesting is going on there.

My suggestion is to take Golden's implicit advice. It's time to stop the fuss and start working the problem, and reading *Succeeding with Open Source* is a good way to do just that. I recommend it highly, and hope that readers will be able to gain that critical advantage they need in their new or existing businesses.

**Terry Bollinger** is a principal information systems engineer at the MITRE Corporation. Contact him at terry@mitre.org.

# Anticipating Change: How to Change the Organization, Starting with Yourself

**Mike Barker**

**Quality Software Management, Volume 4: Anticipating Change** *by Gerald M. Weinberg, Dorset House, 1997, ISBN 0-932633-32-3, 504 pp., US$50.95.*

Does your organization develop software as well as it possibly can? Does it manage that work as well as possible? Are the results and process high quality? If so, you might not need to read Gerald Weinberg's book, *Quality Software Management, Volume 4: Anticipating Change*.

However, if you want changes but don't know where to begin or how to get the changes to start or stick, you'll find it a useful compendium of advice. It begins with a hard look at why change isn't easy and recommends a model built around human reactions to change:

- Stage 1: Late status quo (I'm happy where I am.)

*Critical point 1: The intrusion of the foreign element*

- Stage 2: Chaos (Oh no, I have to change.)

*Critical point 2: The transforming idea*

- Stage 3: Integration (Alright. Let's get this settled.)
- Stage 4: New status quo (The return to stasis.)

From there, Weinberg turns his attention to personal-change artistry and provides a series of challenges to let you practice what he's preaching. Next, he discusses organization and planning that provide a basis for actually carrying out change as opposed to simply talking about it. Last, he lays out his vision of what needs to change and how.

The book isn't a cookbook with recipes for successful change management, but it does provide a lot of advice about things you'll want to consider when trying to accomplish change. Reading it won't make you a master change artist, but it will provide a framework, exercises, and places for you to start on that journey. You might not want to read the book in order, but I'd recommend at least reading Part One to give you a basic understanding of the change model Weinberg uses. From there, you can look up sections in the table of contents that interest you.

While you're looking through the book, you'll see references to Weinberg's software cultural patterns. He describes them in "Appendix C: Software Engineering Cultural Patterns," but throughout the book, he refers to them only by number and name. Here's a summary:

- Pattern 0, Oblivious Culture: "We don't even know that we're performing a process."
- Pattern 1, Variable Culture: "We do whatever we feel like at the moment."
- Pattern 2, Routine Culture: "We follow our routines (except when we lose our nerve)."
- Pattern 3, Steering Culture: "We choose among our routines based on the results they produce."
- Pattern 4, Anticipating Culture: "We establish routines based on our past experience with them."
- Pattern 5, Congruent Culture: "Everyone is involved in improving everything all the time."

The metaphor for the Congruent Culture is worth noting: "The Starship Enterprise: When going somewhere, we can go where no one has gone before, we can carry anything, and we can beam ourselves anywhere, but this is all science fiction."

Sometimes I think you can measure a book's richness by the difficulty you have determining its "point." Horst Remus in *Computing Reviews* says this volume "presents a recipe for a quality software engineering organization." Warren Keuffel in *Software Development* says it "addresses how to create an environment conducive to implementing the software engineering culture he describes in the first three books of the series." The reader reviews at Amazon.com have a wide set of descriptions of what this volume does for you.

If you're looking for a down-to-earth manual on software development, software quality, or project management, this probably isn't the right book for you. However, if you're grappling with how to improve software development and especially how to improve managing software development, then this might be exactly right for you.

**If you're grappling with how to improve software development and especially how to improve managing software development, then this might be the right book for you.**

Don't expect to read it once and run right out to apply the Weinberg system, because it isn't that kind of a book. But it is the kind of book you might read several times, finding different insights as you work through the exercises and challenges. And when you've read a section, you'll probably find yourself thinking about it and the questions it raises for a while—that's what change is all about.

**Mike Barker** is a visiting scholar in the Graduate School of Information Science at Nara Institute of Science and Technology in Nara, Japan. Contact him at mbarker@computer.org.

## Software Project Management for Nondevelopers

**Philipp K. Janert**

**Waltzing with Bears: Managing Risk on Software Projects** *by Tom DeMarco and Timothy Lister, Dorset House, 2003, ISBN 0-932633-60-9, 196 pp., US$27.95.*

You can organize a software project in one of two ways: either around a technical lead, who's a developer (or former developer), or around a project manager, who might have a technical background but generally isn't qualified to inject technical content. Both will face different challenges and have differing approaches. Although the technical lead might not write actual code, he or she will nevertheless be intimately involved in all aspects of the software being implemented. He or she might understand the project's internal problems quite well but might not pay enough attention to such factors as the relationship with customers and other stakeholders, the project's overall goal as it's expressed in the business case, and even the purpose that the project is ultimately sup-

posed to fulfill. The project manager, on the other hand, might be all too aware of the business case and project goals but will be challenged to appreciate or even acknowledge the technical constraints and complexities inevitable in a typical software project. *Waltzing with Bears* is intended mainly for this kind of person.

Tom DeMarco and Timothy Lister spend about the first quarter of their (short!) book arguing that software projects are risky and that the only promising approach is to take risk management seriously. Software professionals need no convincing of this—and it's more than a bit surprising that *anyone* might. After all, unmet schedule and quality promises are the overwhelming rule in software development, from shrink-wrapped application software to nationwide toll-road systems. However, for someone steeped in a culture that measures success by the aggressiveness of your goals, and where saying "it simply can't be done in the time allotted" marks you as an inefficient whiner, the realities of software development might be hard to get used to. After all, nothing is impossible for the person who doesn't have to do it.

As a key concept, DeMarco and Lister introduce what they call a "risk diagram": a curve showing the relative probability of completing a project in a certain amount of time. Clearly, the curve is bell-shaped, with a long tail and a shortest-possible completion time, before which the probability of success is essentially zero. The authors then recommend numerous ways to estimate this curve (for example, by using other projects' postmortems to discover likely risks, estimating each identified risk's impact and likelihood, and encouraging honest communication about perceived risks). They even provide a little downloadable Excel spreadsheet model to obtain the overall risk diagram from a set of quantified individual risks.

The authors also advocate an incremental approach to delivery, both as a way to track project status and as a risk mitigation strategy. Again, active software professionals won't find much new here; the treatment remains very high level. For instance, the authors don't discuss the subtle issue of how to design meaningful acceptance tests to ensure a subsystem's completion and how to conduct them formally.

Finally, DeMarco and Lister explore cost-benefit analysis, strongly recommending that a project's benefits should be spelled out with as much precision as the project's costs. The risk to be managed here is that if you can't measure your project's success, it might as well have failed. Here one also finds the amazing observation that, in the authors' experience, "the one common characteristic among death-march projects is low expected value." After all, if the value proposition for these projects had been more positive, they would have been planned, budgeted, and staffed properly. I'm not sure that my experience bears this out (incompetent management, even on high-expected-value projects, is much more common than one would expect), but their comment is thought-provoking nevertheless. These considerations aren't part of actual project management but deal with bigger business issues such as project inception, approval, and sponsorship.

As the last section confirms again, *Waltzing with Bears* isn't a technical book, and it isn't meant for technical people. It is (and reads like) a business book, meant for business-oriented managers with little software development experience. The advice offered here to such readers is sound, and the book is short and entertaining enough to be read on an airplane.

What is surprising and (depending on your perspective) more than a little unsettling is that the individual developer never comes into view. From the project manager's viewpoint, the project is apparently simply an engine on the path to greater glory. Accordingly, his or her prime worry regarding the developers is that one might fail, the way a radiator hose fails on a car, by quitting and looking for a more fulfilling work environment. This is the most curious observation I took away from *Waltzing with Bears*, the latest book by the authors of *Peopleware*.



***Waltzing with Bears*** **isn't a technical book, and it's not meant for technical people. It's a business book, and as such, it offers sound advice for business-oriented managers.**

**Philipp K. Janert** is a software project consultant and maintainer of the beyondCode.org Web site. Contact him at janert@ieee.org.