

An analysis of Edward Kit's *Software Testing in the Real World - Improving the Process*

By Tammy Hoganson

On occasions when I've had an opportunity to interact with my testing and quality colleagues at various conferences, seminars, and meetings, one thing that I've often noticed is a certain air of "preaching to the choir." The people who are attending the conference or event obviously "get it;" that's why they're there. But I can't count how many times I've heard a tester rather wistfully say, "I wish <my manager/the development director/my CTO/someone with more influence than me> was here to hear this. Then I could make more of an impact on my organization's quality issues." I've even said it myself.

Yes, a quality worker's job is a little less stressful if decision-makers have knowledge of current industry practice. But if you can't talk these individuals into tag-teaming a quality or testing conference with you, you might consider passing along a copy of Edward Kit's "Software Testing in the Real World - Improving the Process."

Above and beyond its education value, one of the book's greatest attributes is its focus on reality. Most of us don't need another guru selling us yet another 'revolutionary' testing model, tool, name-brand methodology or supposed silver bullet. This book doesn't promise to create a testing utopia. It contains

no dogmatic arguments about whom or what is "right." Instead, it offers concrete, practical advice about what an organization can do to improve their testing process today, in the midst of day-to-day chaos. It presents information about the current state of industry practice, and allows the reader to assess which techniques might work in their company at any particular point in time. It makes suggestions about how to increase your own influence, instead of feeling so dependent on others'.

Readers looking for "geek-level" nitty-gritty detail will have to look elsewhere. Weighing in at less than 250 pages including appendices (shorter than some executive summaries I've read!), the book manages to highlight our most critical testing issues and their potential ramifications to our counterparts in development, the business unit, and the executive suite. At the book's foundation is the concept of improving the testing process - emphasizing that 'testing' tasks are integrated throughout the product development cycle, and that it's not just testers who test.

The book is organized in four basic sections: Software Testing Process Maturity, Framework for Test Process Improvement, Testing Methods, and Managing Test Methodology. This review highlights some issues within these sections that strongly resonate for me.

Software Testing Process Maturity (Chapters 1-3)

"Test groups that operate within organizations having an immature development process will feel more pain than those who don't."

I think one of the biggest services Kit provides to the testing community with this book is that he exposes how dependent testing teams are upon others who are involved in a product development cycle. A quality worker's most critical need is information, and organizations having immature development processes rarely produce the software engineering artifacts - such as requirements documents - that enable testers to do their jobs with the desired level of professionalism and rigor. Testers working within the type of organization he refers to in the quotation above feel acutely dependent upon developers in particular: for providing timely, accurate information, for resolving defects correctly, for producing the builds that drive our workload, for making real (in bits and bytes, anyway) the very product we work on. In this environment, you learn very quickly that information is likely to come through word of mouth. Making a buddy in the development organization is a very productive move.

Another great education point Kit makes is that testing has evolved into a full profession, having its own technical body of knowledge and professional certification opportunities. Unfortunately, many hiring managers still aren't aware of this and continue to staff the testing function with "junior developers" and present the assignment as a steppingstone into a "real" development job. Kit's inclusion of this topic increases awareness that testing professionals *choose* to channel their technical skill into verification and validation rather than creation. It's a career choice; they're not marking time until something 'better' comes along.

Bravo to Kit for gently rebuking academic institutions for their historic lack of attention to testing and quality topics in Comp.Sci., MIS and Software Engineering curricula. This learning gap directly impacts product quality once these students enter the workforce, especially in organizations still struggling to establish basic software engineering processes.

Kit does a great job of illustrating both the promise and limitations of testing tools. As he states, tools have indeed "come into their own." Classic capture/playback tools are more robust than ever, and the load and performance tools released in most suites' web toolkits provide much-needed capabilities. But such tools should not be perceived as the next silver bullet. There is still a serious under-appreciation for the amount of process maturity that needs to be in place to get even minimal benefit from such tools. Remember – they're only "automatic" after someone (usually a quality worker) analyzes, designs, codes, tests, then releases the suite as an executable entity.

Framework for Test Process Improvement (Chapters 4-6)

At the beginning of Chapter 4, Kit urges the reader to take a practical perspective on the topic of process improvement.

As a practitioner, I have found that this is not just a very good recommendation, but a political and psychological necessity. Testing process improvement is especially challenging because most of us have to work within an existing framework and culture, and most of our development and managerial co-workers need at least some instruction as to why we're recommending what we're recommending. When web release cycles are measured in weeks, rather than months or years, some product managers see little incentive to take the learning curve 'hit' required to learn to do something a new way – even if it will save time in the long run. You must work with what's in place, and do what you can given the needs and constraints of the project.

"The quality of requirements is an important indicator of the maturity of an organization."

In my experience, one of the biggest areas of misunderstanding is recognizing (and valuing) the skills that it takes to be a good tester. To the surprise of many developers and technical managers, they aren't the same skills it takes to be a good developer, though of course there is some overlap. The main distinction I find is in mindset, which Kit discusses. Testers hunt errors, and to a large extent must assume that a product WON'T work as it's supposed to. The tester must work as a skeptic, as a devil's advocate, in order to add value. This mindset tends to balance out a developer's natural confidence, reducing risk for the organization by illuminating errors as early in the process as possible. A tester's contribution to the team is to provide diplomatic, objective visibility into the product – good news and bad.

Given that there's always more demand for testing services than can be accom-

modated in most organizations, it's essential that a company strategically deploy their quality workers. Active risk management helps an organization use its testing resources where they're most urgently needed.

Planning is critical. Software project management is an area in which so many organizations continue to experience problems. To paraphrase Kit, without adequate planning, you can't know what to do, how to do it, how long it will take, how many people are needed, or what it will cost. (Would you personally bankroll a project so poorly planned?) In less mature organizations, testers and test teams tend to be particularly impacted by poor planning because it so often results in missed milestones, slipped internal schedules, and late delivery of the product for testing. But rarely does the product's actual ship date change. This significantly reduces test coverage and increases project risk.

Testing Methods (Chapters 7-12)

Kit joins other software practitioners in his recommendation that we make more aggressive use of 'low-tech' verification methods such as checklists. (See the checklists he helpfully includes in the appendices, and reuse, reuse, reuse!) A sexy tool might be more fun, but it certainly carries more overhead, cost and risk.

Inspections and reviews are, of course, proposed as a high leverage activity. Unfortunately, many organizations don't seriously consider the idea because of their perceptions about the amount of rigor and/or overhead it might require. The reality is that you don't have to do full-blown Fagan inspections to get some level of benefit. If even one additional person looks at a work product, you create an opportunity to find defects that might not otherwise be discovered.

Tucked away in Chapter 7, "Verification Testing," (p. 65) are, in my opinion, some of the most significant sentences in the

entire book: “*The quality of requirements is an important indicator of the maturity of an organization. If there is no lifecycle, if there is no agreement that there will be a requirements document, and there’s no agreement on who writes it, what should be in it, or even what are the frameworks for acceptable requirements, development will be difficult and proper testing will be extremely difficult.*” [Emphasis added.] The requirements document (or whatever it’s called in your organization) is at the foundation of process maturity and test success for a number of reasons. Here are a few that come immediately to mind. (1) With a requirements document, there is least the POTENTIAL to verify early - to less expensively discover defects before they become part of the code base. (2) There’s a vehicle (other than word-of-mouth) for information exchange – remember, the more information that is available, the more value a tester can provide. (3) More comprehensive test planning can be performed. Testers use specific techniques to maximize the opportunity to find defects at each phase of the development cycle, but their arsenal is limited when information is sketchy.

Speaking of test planning, another critical point Kit makes is that it’s essential to prioritize tests during your test planning activities, because, as we all know, you never have enough time to test everything. Make sure you have a clear understanding of the level of ‘quality’ that is expected in context with the other goals that the product is attempting to meet. This time-to-market slant may feel a bit like a cop-out to trained quality professionals, but that’s the reality that some of us find ourselves in. Provide the best coverage you can with the resources you have available.

Managing Test Methodology (Chapters 13-15)

Reorganizations are tiring, especially for testing or quality teams who often must do a fair amount of ‘educating’ before successfully influencing co-workers to work with them instead of against them. The book proposes seven potential organizational structures for the testing function, with the strengths and weaknesses of each approach provided as food for thought. Kit recommends assessing your organization objectively, then determining which structure might work in context with your organization’s unique technical and cultural issues, and

its business needs and goals.

One of the things I like most about this book, and others of its ilk (Karl Wiegers’ *Creating a Software Engineering Culture* comes to mind) is that the focus is on practicality, on what can be accomplished now. It exhorts readers to make informed decisions for themselves after assessing their unique reality. It doesn’t preach to the choir as much as enlighten the congregation – and in a way that doesn’t insult them or make them feel defensive.

Common sense ISN’T so common, and that’s what keeps me reaching for this book time and time again. It’s so reasonably priced that you can give individual copies to your favorite influential developer, an open-minded manager, and your CTO. It just may be the best training investment you make this year.

➤ *Tammy Hoganson has 12 years’ experience at West Group as a developer, tester, quality supervisor, instructor, architect and change agent. You can reach her by email at tammy.hoganson@westgroup.com*

Graduate Programs in Software Engineering, IT & IS

at the University of St. Thomas is celebrating 15 years of providing quality software development educational opportunities to the Twin Cities & MN.

Looking to enhance your career with an advanced degree in IT or IS?

Graduate Programs in Software Engineering, IT & IS offers **master’s** programs & **graduate certificate** programs, **evenings & Saturdays**.

Are you interested in learning more about software?

Try the **Mini Master of Software Design & Development** series - 12 different software development topics one evening a wk; the **Mini Master of IT & IS** series - 8 different software topics one evening a wk; or a short course in C++, JAVA, Objects, UNIX, Visual Basic, Software Project Management, etc.



651.962.5500

-

www.GPS.stthomas.edu

-

gradsoftware@stthomas.edu

University of St. Thomas admits students of any race, color, creed and national or ethnic origin.