

A Process That Is Not

Hakan Erdogmus

TikiWiki, or Tiki in short, is a notable piece of software. It's open source, big, successful, and widely used. Nothing too special about that. But Tiki also embraces Erik Raymond's "bazaar" model (www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar) in the extreme. It's not driven by a handful of core developers, or supported by an ecosystem of third-



party contributions that plug in to the core. It doesn't have any systematic quality controls: no design reviews, no testing, no real gatekeepers, no architecture workshops, no imposed architecture, nothing. It doesn't have an ad hoc steering committee making quasi-binding decisions about what's important and what's not, or who's allowed to touch what, or veering the project strongly in this direction or that. Granted, at any given time, it does have its visionaries, marketing czars, technical leaders, maxims of development, best practices, and so on that provide continuity, visibility, stability, and motivation. But most remarkably, what Tiki has is a large, unrestricted contributor base, none of whom enjoys special commit privileges. Yet Tiki works, despite the software and the antiprocess used in developing it.

Did I just call it an *antiprocess*? Apologies! Of course, there's a process: it's just not what we who care about process and preach its virtues think should prevail in a serious initiative. It represents the odd data point, where our honorable assumptions about the correlation between process and success all go down the drain.

Tiki (<http://tikiwiki.org>) is ubiquitous, multi-purpose, feature-rich collaboration software with

a wiki engine. It's used in Web sites, both commercial and nonprofit, in various ways: as a plain wiki, a wiki on steroids, a content management system, a groupware application, a Web application, a Web portal, an issue-tracking system, a form generator, a knowledge base, and combinations thereof. A close colleague, Alain Desilés, first brought Tiki to my attention. Alain is a contributor himself, and was amazed at how the Tiki approach works on the scale that it does. To reassure those who are unfamiliar with Tiki that it's not marginal software with marginal success, here are some facts.

Tiki Facts

Tiki was first released in 2002 and has been under active development for seven years. It now has more than a million lines of PHP code and more than a thousand features and configuration options. With over 200 active source-code contributors, Tiki has one of the largest open source teams in the world. Tiki contributors are among the top 2 percent of all project teams on Ohloh (an open source directory recently acquired by SourceForge). It's reported that the Tiki code base typically receives over 20 commits a day. When I checked on 5 September, it had had 18 commits over a two-day period, most of them fixes, refactoring, and merges. That's still quite a lot of activity.

Tiki has over 700,000 downloads from SourceForge alone (not counting installs through Web hosting services). But is there any evidence that anybody important cares about Tiki? Yes. Mozilla Firefox has adopted it for their official support site. It's included in the Fantastico script installation library, a standard control panel application offered by most Web hosting services. Tiki is also one of

Mission Statement:

To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

Kudos and Thanks

In this issue, editors Rebecca Wirfs-Brock and Robert Glass wrap up their two hugely popular columns: Design and Loyal Opposition. I want to thank Rebecca and Bob for writing thought-provoking articles and recruiting excellent guest authors every so often. We will miss their columns, but we're pleased both are staying with us as members of the *IEEE Software Advisory Board*.

Also joining the Advisory Board are two new members: Ayse Basar Bener and Douglas R. Vogel. Ayse switched to academia in 2002 after a 15-year career in the finance and banking industries. She held senior executive positions leading large IT initiatives and managing IT operations before joining the faculty at Bogazici University, Istanbul. Her current research focuses on empirical software engineering and involves close interaction with industry. Doug is chair professor of information systems at the City University of Hong Kong and an Association for Information Systems Fellow. He began his professional career as a software engineer in the aerospace industry and later served as a general manager in the computer manufacturing industry. Doug's interests bridge the business and academic communities on multiple continents and address interpersonal communication, group problem solving, cooperative learning, multicultural team productivity, and knowledge sharing.

the top 50 most popular applications on Freshmeat. A Google search for "inurl:tiki-index.php" returns over 37 million hits, with every distinct hit indicating a possible live install. Is this enough?

Observations at TikiFest

Upon Alain's invitation, I attended a Tiki coding spree, a TikiFest, on a sunny spring day in Ottawa. Seven dedicated developers with laptops were gathered around a large table in the bright downtown offices of Code Factory, a nonprofit organization supporting the local development community. Five were working alone. Alain was paired with a developer from Germany who was in Ottawa for a larger occasion. The room was eerily quiet for such an event, disturbed only by the occasional whispering from the pair and the odd question thrown at the group. There were no stickies on the walls. The white board was sparse, with some scribbling on it. I had imagined a TikiFest room to look like a project war room, like the ones you would see in an agile environment. It did not look like that.

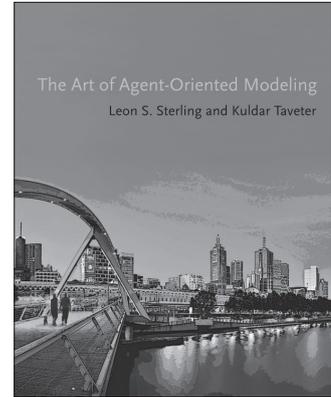
Hovering around the seven developers was Marc Laporte, Tiki's project administrator since 2003. We started talking. I asked about his vision for Tiki. Marc wants Tiki to evolve into a sort of general-purpose Web-based operating system, which we could install locally and ac-

cess through a browser. Good. I was more interested in how the project was managed.

I asked about TikiFests and what the community hoped to achieve in these events. Marc said they've had about 35 TikiFests over the past five years. Somebody initiates the event to work on a specific feature, refactoring, or release. The organizer advertises it. The events are open, and any number of people can attend. Sometimes they're collocated with other, larger happenings, like WikiSym (<http://wikisym.org>). The philosophy is similar to that of an unconference, such as the BarCamp gatherings ("Unconferences Catch On with Developers," *IEEE Software*, Nov./Dec. 2008).

The Tiki Way

The Tiki project has a soft organization with no central authority yielding power. Marc summarized it as bootstrapping the "Wiki way" of working to develop software collaboratively (*The Wiki Way: Quick Collaboration on the Web* by Bo Leuf and Ward Cunningham, Addison-Wesley, 2001). For software development, this philosophy implies collaboration at a larger scale than usual. And Marc is behind Tiki's "recruit early, recruit often" strategy, which encourages open participation by as many people as possible. The strategy applies indiscriminately to code, documentation, ideas, translation, and whatever else needs

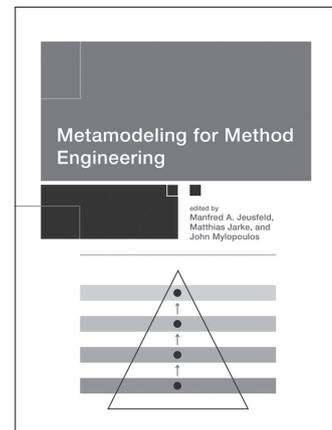


The Art of Agent-Oriented Modeling

Leon S. Sterling and Kuldar Taveter

"In *The Art of Agent-Oriented Modeling* readers will find an answer: a thorough description of all the ideas behind agent-oriented software engineering and a new approach to modeling that can fit many different methodologies. Far from being a painful set of definitions and procedures, it will be a pleasure to read." — Maurizio Martelli, Università di Genova

Intelligent Robotics and Autonomous Agents series
408 pp., 141 illus., \$38 cloth



Metamodeling for Method Engineering

edited by Manfred A. Jausfeld, Matthias Jarke, and John Mylopoulos

A practical guide to method engineering based on metamodeling, with theoretical foundations and case studies, suitable for classroom use or as a reference for practitioners.

Cooperative Information Systems series
424 pp., 154 illus., \$55 cloth

To order call 800-405-1619 • <http://mitpress.mit.edu>
Visit our e-books store: <http://mitpress-ebooks.mit.edu>

EDITOR IN CHIEF

Hakan Erdogmus

hakan.erdogmus@computer.org

EDITOR IN CHIEF EMERITUS:

Warren Harrison, Portland State University

ASSOCIATE EDITORS IN CHIEF

Computing Now: Maurizio Morisio, Politecnico di Torino; maurizio.morisio@polito.it

Design/Architecture: Uwe Zdun, Vienna Univ. of Technology; zdun@infosys.tuwien.ac.at

Development Infrastructures: Martin Robillard, McGill University; martin@cs.mcgill.ca

Distributed and Enterprise Software:

John Grundy, University of Auckland; john-g@cs.auckland.ac.nz

Empirical Results: Forrest Shull, Fraunhofer Center for Experimental Software Engineering, Maryland; fshull@fc-md.umd.edu

Human and Social Aspects: Helen Sharp, The Open University, London; h.c.sharp@open.ac.uk

Management: John Favaro, INTECS; john@favaro.net

Processes and Practices: Frank Maurer, University of Calgary; maurer@cpsc.ucalgary.ca

Programming Languages and Paradigms:

Laurence Tratt, Bournemouth University; laurie@tratt.net

Quality: Annie Combelles, DNV/Q-Labs; annie.combelles@dnv.com

Requirements: Neil Maiden, City University London; cc559@soi.city.ac.uk

Ann Hickey, University of Colorado at Colorado Springs; ahickey@uccs.edu

DEPARTMENT EDITORS

Bookshelf: Art Sedighi, SoftModule

Career Development: Philippe Kruchten, University of British Columbia

Design: Rebecca J. Wirfs-Brock, Wirfs-Brock Associates

Loyal Opposition: Robert Glass, Computing Trends

On Architecture: Grady Booch, IBM

Pragmatic Architect: Frank Buschmann, Siemens

Requirements: Neil Maiden, City University London

Software Technology: Christof Ebert, Vector

Tools of the Trade: Diomidis Spinellis, Athens Univ. of Economics and Business

User Centric: Jeff Patton, consultant

Voice of Evidence: Forrest Shull, Fraunhofer Center for Experimental Software Engineering

ADVISORY BOARD

Frances Paulisch, Siemens (Chair)

Pekka Abrahamsson, Univ. of Helsinki

Jennitta Andrea, ClearStream Consulting

Elisa Baniassad,

Chinese University of Hong Kong

Ayse Basar Bener,

Bogazici University

J. David Blaine, consultant

Kaoru Hayashi, SRA

Simon Helsen, IBM Rational

Gregor Hohpe, Google

Steve McConnell, Construx Software

Grigori Melnik, Microsoft

Linda Rising, consultant

Wolfgang Strigel, consultant

Dave Thomas, Bedarra Research Labs

Douglas R. Vogel,

City University of Hong Kong

Markus Völter, consultant

FROM THE EDITOR

doing. Marc seems unconcerned about who contributes and how qualified they are, so long as smart, dedicated, and competent people participate. On his tikiwiki.org user page, he writes, “As Tiki is used more and more, our exceptional dev team always rises to the challenge.”

Another governing Tiki philosophy is strong individual ownership. Each feature is typically adopted by one or more contributors. Experienced contributors band together or help less experienced ones to ensure that Tiki’s core features and cross-cutting functionality continue to work correctly. Marc characterizes the spirit of the community as one “condemned to work together.”

Requirements

Who decides how the application is extended and which features take priority? Nobody in particular. A single, flat wish list exists on the developer portal: anyone can add a feature request to the list, and anyone can pick any item from the wish list and pursue it. Marc points out that this unmanaged approach doesn’t lead to duplication of similar features. The culture enforces a practice of checking what’s already been implemented before embarking on a new pursuit. Such checking is possible thanks to the extensive central documentation (over 1,000 pages), which, again, nobody in particular is responsible for. The documentation is contributed by a large, decentralized body of people. The end result is an application extremely rich in built-in features without much functional redundancy across them. However, this result isn’t accidental: from the get-go, Marc adds, Tiki was intended to be an application with lots of features.

Marc is indeed on record about his unconcern for feature bloat. He writes, “You can add all you want as long as you make it optional and it doesn’t break anything,” and repeats the orientation set by founder Luis Argerich: “With enough eyeballs and adopt-a-feature, this is not a problem. People just activate what they need anyway.”

Design

Tiki has no notable central design. It’s pretty much a monolithic application. Features are directly integrated into the core: no fancy component or plug-in architecture

or capability to support external features. Everything is contributed to a central repository. The user gets everything but can turn individual features on and off, or select a standard profile with the features that best fit the purpose at hand. Marc thinks of modern plug-in architectures as preludes for “dependency hell.” He brags that by avoiding dependency hell, the Tiki project can release everything every six months. Smaller projects with lots of external features can take up to a year for the dependents to catch up: third-party contributions often eventually get abandoned for that reason.

The code base allegedly is structurally stable: “The code base is in many ways very different now than it was six years ago, but the same underlying structure still prevails.” I couldn’t find any documentation on how to navigate this beast on the Tiki developer portal. You’re strongly encouraged to participate in the mailing lists and chat rooms and ask for help, but it pretty much stops there. Apparently, this level of support is enough, given the large number of contributors.

When I ask about how the code’s integrity is preserved, Marc states that code is refactored only after it’s been around for a long time, once the developers know what it’s supposed to do and are convinced that refactoring is needed. So it boils down to a “if it’s not broken, leave it alone” philosophy. None of the pep talk in my previous column about architecting and architecture (“Agile Meets Architecture,” *IEEE Software*, Sept./Oct. 2009) matters here. It’s a different world.

Quality

Quality is also self-regulated by the culture. The centralized code base and documentation are the focal point of all activity. Users need not worry about third-party patches or updates to tens of plug-ins. Contributors need not worry about external dependencies to comply with, except for standard platform components such as MySQL and Apache.

Eric Raymond’s proposition “Given enough eyeballs, all bugs are shallow” is Tiki’s main arsenal. The “recruit early, recruit often” strategy generates the needed eyeballs. “Dogfooding” also helps: the Tiki portal has been running Tiki for some time. Nearly 18,000 registered members make plenty of watchful eyes. A buggy fea-

ture is discovered quickly and gets fixed, if it's popular and used often enough. If it's not, the feature, and the bugs that came with it, are condemned to die a death of disuse. If a new feature interferes with existing popular features, the problematic feature is likely to be removed if it can't be fixed.

A spin-off to the "recruit early, recruit often" strategy is to "commit early, commit often." Frequent and timely commits provide fast feedback. The high commit activity for bug fixes indicates that bugs indeed get fixed. Or at least somebody is aware and working on them.

When I inquire about testing, I get a tired look from Marc. He's been asked this question before. He recites the "one-million monkeys" metaphor. He mentions the diligent eyeballs, dogfooding, spirit of collaboration: "No technical problems can survive if you work together." And I've heard that before. Marc states without shame or hesitation that Tiki might have more bugs by standard measures than any other comparable application. However, that's a direct consequence of a deliberate trade-off: a rich set of built-in features over low overall bug density. But the bug density is not uniform with most popular features being also the most stable and highest quality since they're subject to the most scrutiny.

Thanks to a rating system, users know what they're getting on a feature-by-feature basis: the community evaluates each feature continually, and the ratings change over time. So at least the quality is visible on a fine level. This visibility is important for the users, who decide which features to turn on and off.

Marc is admittedly worried about quality when it comes to security flaws. He doesn't mention any specific measures. He tells me that the biggest security flaws were caused by the best developers, a consequence of the best people coding more and taking more risks.

Still no mention of projectwide tests, projectwide testing strategy, or systematically advocated testing practice. Blame my paranoid nature, but I'm not completely at ease.

An Alternative Perspective

My unruly tongue called what underlies the development of Tiki an *antiprocess*. I have already apologized for it, but I can do better.

If you just see the "have-nots," your reaction might be "How could this possibly work?" In reality, and on deeper examination, the Tiki project is still seriously organized as most large open source projects are. It has a vision, a dedicated community, contribution principles, guidelines for newbies, rules of engagement, suggested development practices and patterns, mentoring, a solid central infrastructure, an issue-tracking system, extensive user documentation, a feature list, a user rating system, and so on. And all these parts work together to create something significant that works. You might dismiss Tiki as an application not intended for important tasks. Therefore, you might think its users tolerate the lack of qualities expected of worthier applications. Even if Tiki looks less critical than some other large software systems, with so many users and such wide distribution, it's probably used in many contexts that support critical operations. I don't intend this comment as an advertisement for Tiki. I know that it would be a source of worry for many.

Before I close, let's look at Tiki one last time from the perspective of the seven dimensions in my essay "Essentials of Software Process" (*IEEE Software*, July/Aug. 2008). Tiki's governance approach would

score fairly well in four of those seven dimensions. The Tiki way clearly supports human centricity, pragmatism, empiricism, and experimentation. Value orientation is partially addressed: the Tiki way would probably fair badly in terms of efficiency (as would most open source projects that don't operate under limited resource constraints) but well in terms of end-user value (users ultimately decide what stays in, their wishes are visible, and they get what they collectively want fast). As for the remaining dimensions—technical orientation and discipline—they would be a hard sell for the Tiki way. Still, not so bad for something that I called an antiprocess.

The future of Tiki isn't certain. I don't know how much longer its purely organic philosophy is sustainable. The Tiki way has its caveats, some pretty severe, but Tiki has been around longer than a lot of other software. It works. Many people use it every day. It supports the development of one of the most pervasive pieces of software. Given these facts, the seven essentials I espoused don't appear to be universal, necessary conditions after all. But I knew that, even if I might not have said it before. ☺



UNIVERSITY of WASHINGTON | BOTHELL

COMPUTING & SOFTWARE SYSTEMS

The University of Washington Bothell
Assistant Professor — Software Engineering

The Computing and Software Systems Program at the University of Washington Bothell (UWB) invites applications for a tenure track Assistant Professor position with expertise in Software Engineering to begin fall 2010. All University faculty engage in teaching, research, and service. Areas of research and teaching interest include, but are not limited to: Requirements Engineering, Quality Assurance, Testing Methodologies, Software Development Processes, Software Design Methodologies, Software Project Management, and Collaborative and Team Development.

The Bothell campus of the University of Washington was founded in 1990 as an innovative, interdisciplinary campus within the University of Washington system — one of the premier institutions of higher education in the US. Faculty members have full access to the resources of a major research university, with the culture and close relationships with students of a small liberal arts college.

Required qualifications for the position include an earned doctorate in computer science, software engineering, or another relevant technical field, along with a body of scholarship, or demonstrated promise for future work, that warrants UWB appointment at the rank of Assistant Professor, and demonstrated commitment to excellence in undergraduate and graduate education.

To apply, please send a cover letter, curriculum vitae, a list of at least three professional references including contact information, a statement of teaching philosophy, evidence of teaching effectiveness, and a research plan to css-search@uwv.edu. Review of applications will begin on November 15, 2009; the position will remain open until filled. For additional information, please see our website at <http://www.uwb.edu/CSS/>.

The University of Washington, Bothell is an affirmative action, equal opportunity employer.