IS TIKI THE RIGHT SOLUTION FOR MY PROJECT? AND WIKISUITE?

Who should read this article?

Anyone making technology choices for organizations, and wondering how to make the best long term choice, and to avoid issues such as vendor lock-in, ending up with yet more systems that don't adapt to emerging needs and become legacy systems, etc.

In 2009, the IEEE Computer Society published an editorial entitled "A Process That is Not" on the software development model/process of Tiki Wiki CMS Groupware. A copy of the article is shown below, along with follow-up comments and an update on how the project has fared since the article was written.

What is the IEEE Computer Society?

"The IEEE Computer Society is the premier source for information, inspiration, and collaboration in computer science and engineering. Connecting members worldwide, the Computer Society empowers the people who advance technology by delivering tools for individuals at all stages of their professional careers. Our trusted resources include international conferences, peer-reviewed publications, a robust digital library, globally recognized standards, and continuous learning opportunities." The website is at Computer.org.

What is Tiki Wiki CMS Groupware?

Tiki Wiki CMS Groupware (also known as TikiWiki or Tiki) is an Open Source web application that has been developed by a worldwide community of contributors since 2002. Tiki is the Free/Libre/Open Source Web Application with the most built-in features:
https://tiki.org/FLOSS-Web-Application-with-the-most-built-in-features.

Why was this article originally written?

The IEEE Computer Society strives to understand what works and what doesn't work. They were trying to comprehend how a community Open Source project could be so successful without having a more centralized structure and the deep pockets of a large corporation or a government.

Why are we following up a decade later?

The article concludes wondering "how much longer its purely organic philosophy is sustainable". The evolution of the project provides clear answers to this question. Please read the article keeping in mind it **was written in 2009**, over a decade ago. And after reading the article, please proceed to our follow-up notes.

Editor in Chief: Hakan Erdogmus ■ Kalemun Research ■ hakan.erdogmus@computer.org

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, multipurpose, 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 Desiléts, 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.

 0740-7459/09/$26.00 © 2009 IEEE

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 access 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

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.

The article above is from 2009. And it concludes wondering "how much longer its purely organic philosophy is sustainable".

Tiki in numbers

What	Stat	Reference
Number of code commits	79625 as of 2022-01-30	https://sourceforge.net/p/tikiwiki/code/79625/
Number of software code committers	As of 2025-11-18, 446 individuals from multiple organizations	https://openhub.net/p/tikiwiki
"Cost to develop" (value, according to Open Hub)	Over USD$10 million	https://www.openhub.net/p/tikiwiki/estimated_cost
Number of messages on the developer's mailing list	Over 50,000	https://sourceforge.net/p/tikiwiki/mailman/tikiwiki-devel/
Number of wiki pages of documentation	Over 2000	https://doc.tiki.org/tiki-listpages.php
Verifiable downloads	Over 1.3 million	https://sourceforge.net/projects/tikiwiki/files/stats/timeline?dates=2002-10-07+to+2022-01-30 Also note that, nowadays, users download .zip files less often, and more frequently get Tiki using installers, source code, Docker, etc., which are not counted in the stats from SourceForge.net.
Registered users on tiki.org	Over 25 000	https://tiki.org/Community

See the list of new features and releases since 2009. LTS means "Long Term Support".

- 2009-05: https://doc.tiki.org/Tiki3 LTS
- 2009-11: https://doc.tiki.org/Tiki4
- 2010-06: https://doc.tiki.org/Tiki5
- 2010-11: https://doc.tiki.org/Tiki6 LTS
- 2011-06: https://doc.tiki.org/Tiki7
- 2011-11: https://doc.tiki.org/Tiki8
- 2012-06: https://doc.tiki.org/Tiki9 LTS
- 2012-12: https://doc.tiki.org/Tiki10
- 2013-07: https://doc.tiki.org/Tiki11
- 2013-11: https://doc.tiki.org/Tiki12 LTS
- 2014-08: https://doc.tiki.org/Tiki13
- 2015-05: https://doc.tiki.org/Tiki14
- 2016-04: https://doc.tiki.org/Tiki15 LTS
- 2016-11: https://doc.tiki.org/Tiki16
- 2017-07: https://doc.tiki.org/Tiki17
- 2018-01: https://doc.tiki.org/Tiki18 LTS
- 2018-11: https://doc.tiki.org/Tiki19
- 2019-06: https://doc.tiki.org/Tiki20
- 2020-03: https://doc.tiki.org/Tiki21 LTS
- 2020-11: https://doc.tiki.org/Tiki22
- 2021-08: https://doc.tiki.org/Tiki23
- 2022-03: https://doc.tiki.org/Tiki24 LTS
- 2022-12: https://doc.tiki.org/Tiki25
- 2023-08: https://doc.tiki.org/Tiki26
- 2024-07: https://doc.tiki.org/Tiki27 LTS
- 2025-01: https://doc.tiki.org/Tiki28
- 2025-09: https://doc.tiki.org/Tiki29

Beyond adding and improving features, Tiki has proceeded with major architectural enhancements to progressively take advantage of modern technology.

Back in 2002, Tiki used CVS, the popular software of the day for source control management, and then moved to SVN, and later Git.

> *Internet Explorer peaked during 2002 and 2003, with about 95% share. Its first notable competitor after beating Netscape was Firefox from Mozilla, which itself was an offshoot from Netscape.*
> *Source: https://en.wikipedia.org/wiki/Internet_Explorer*

For a sense of history, we saw the rise and fall of Flash and Java Applets. In 2020, Chrome and Firefox:

- have a combined market share of over 75% and are collaborating openly to improve browser standards
- offer a new major version every 6 to 8 weeks (IE 7 was released in 2006, after IE 6 in 2001)

Since Tiki started

- Tiki went from a basic PHP 4 application to leveraging possibilities from successive versions of PHP. And we are now moving to PHP 8.
- From HTML 4 to 5, to a living standard. From home-grown CSS to CSS pre-processors, with Bootstrap 3 and then Bootstrap 4 and later 5 (mobile-first front-end framework).
- For the database encoding: From Latin1 to UTF-8 to utf8mb4.
- From barely any JavaScript to rich interactivity with the addition of jQuery in 2009 and Vue.js in 2020.

In 2002, dial-up was still very much a thing!

> *As of the end of March 2003, 31% of home Internet users had a high-speed connection at home.*
> *Source: https://www.pewresearch.org/internet/2003/05/18/broadband-adoption-at-home/*

Tiki went from having a mobile version for WAP phones (in 2003) to jQuery Mobile (an alternative output), to today's responsive technologies. See the evolution at https://doc.tiki.org/Mobile.

In 2020, Tiki added PWA support.

> *A progressive web application (PWA) is a type of application software delivered through the web, built using common web technologies including HTML, CSS and JavaScript. It is intended to work on any platform that uses a standards-compliant browser. Functionality includes working offline, push notifications, and device hardware access, enabling creating user experiences similar to native applications on desktop and mobile devices.*
> *Source: https://en.wikipedia.org/wiki/Progressive_web_application*

Tiki has always prioritized code re-use. And its dependency management has evolved from CVS modules to svn:externals to the current 150+ projects via Composer: https://doc.tiki.org/Composer. And then, in 2021 to Composer v2.

From only end-user testing to continuous integration and an array of various tests via GitLab pipelines: https://gitlab.com/tikiwiki/tiki/pipelines, https://gitlab.com/tikiwiki/tiki/-/blob/master/.gitlab-ci.yml.

In 2002, installing and managing Tiki instances was a very manual operation. Then, in 2008, Tiki Remote Instance Manager was born, as a combination of shell and PHP scripts to install, update, backup, restore and monitor (check security of) a large number of Tiki installations (instances). Along with system configuration features added in 2011, this has facilitated more complex projects which require multiple environments (ex.: development, testing, acceptance, production, fail-over, etc.). Later, a web interface was added. In 2018, the script was revamped/modernized to be fully in PHP (making it more portable to different operating systems) using The Symfony Console Component. And the script was renamed to "Tiki Manager".

Most of the *.tiki.org sites (powered by Tiki, of course) are automatically updated daily from a stable branch, and a test upgrade to the latest code (trunk/master) is done daily, permitting the testing of upcoming enhancements with realistic data, and the detection of regressions. As the developers use the Tiki software itself to communicate, collaborate, track issues and so forth, the software is tested and improved as it is used (this is sometimes referred to as "eating our own dogfood" — making not just for others, but to use in the development process, which is also an indicator of enthusiasm).

In 2019, Cypht Webmail replaced Tiki's home-grown solution.

In 2021, Tiki started the move to the PSR-12 Extended Coding Style standard, to be more in line with the global PHP community.

In 2022, the community added native support for Markdown (CommonMark), which has now become a de facto standard for text-based syntax.

And 2022 culminated with Tiki25, the biggest Tiki release ever.

In 2023, a new build system was introduced for JavaScript and CSS dependencies, to facilitate the management of increasing number and sophistication of JavaScript dependencies: https://dev.tiki.org/The-Tiki-27-plus-Build-System

So our world has massively changed since Tiki's first release in 2002. And Tiki has evolved, relentlessly pursuing the goal of being the best possible Open Source Web Application. The Tiki development model has proven its efficiency and adaptability. During this period, thousands of web application projects have been born, and most do not survive past a few years. For those that do survive, very few enjoy a comparable level of sustainable success. So the question is *why?*

Tiki contributors will argue that it has a more efficient development model than any comparable web application. This is detailed at https://pluginproblems.com/.

Trackers are the "database" component of Tiki. Using Trackers, you can create forms and manage data. Combining with advanced wiki usage provides a no-code/low-code platform which permits to cover various use cases with a common code base: https://TikiTrackers.org/Database-Web-App-Builder. Tiki profiles permit to collaborate on recipes/configurations.

Another key point of Tiki's success is the time-based, "Release early, Release often" approach with Long Term Support (LTS) versions, to balance the needs of stability and innovation: https://tiki.org/Versions. This is especially important for the public sector and the enterprise.

Other applications that have a similarly large feature coverage (Odoo, Drupal, etc.) all have one or many major risks associated:

1. Not Open Source
2. VC funding
3. Fragmented ecosystem

In contrast,

1. Tiki is fully Free/Libre/Open Source. And there is no Contributor License Agreement so Tiki is sure to always stay Free/Libre/Open Source.
2. No VC funding is involved. There is an ecosystem of service providers, and a non-profit association owns the assets such as the Tiki trademarks: https://tiki.org/Tiki-Software-Community-Association
3. Tiki also means "**T**ightly **I**ntegrated **K**nowledge **I**nfrastructure", and is the Free/Libre/Open Source Web Application with the most built-in features: https://tiki.org/FLOSS-Web-Application-with-the-most-built-in-features.

Here is a video interview from 2010 where two very experienced Open Source contributors debate Marc Laporte, a major Tiki contributor: https://tiki.org/article319-Watch-Tiki-s-Marc-Laporte-on-FLOSS-Weekly Simon Phipps (one of the interviewers) later became President of the Open Source Initiative (OSI).

The ongoing success of Wikipedia.org has demonstrated what can be created when many contributors collaborate together in a minimal and fairly flat structure. The Tiki project has applied this model to creating software itself, with developers encouraged to join in and freely add their code fixes and feature enhancements, and the more active members of the Tiki community monitoring the code commits of new participants.

Further pursuing this "Wiki Way" collaborative spirit, Tiki spawned WikiSuite, which was founded in 2011. By 2016, WikiSuite had become, and still is, the most comprehensive and integrated Open Source enterprise solution: https://wikisuite.org/The-most-comprehensive-and-integrated-Open-Source-enterprise-solution. WikiSuite's cost to develop is well over US$50 million: https://wikisuite.org/50-million

While Wikipedia is the broadest unified body of knowledge, WikiSuite is the most comprehensive and integrated Open Source enterprise solution. WikiSuite's mission is to "empower organizations to better manage, secure and use their information, to become data-driven, and achieve their optimal performance". The Tiki community succeeded by federating with other Open Source communities with complementary feature sets. Notably, Jitsi Meet for video-conferencing, Syncthing for file synchronization and backup, MeshCentral for remote computer management, Virtualmin for server administration and Xibo for Digital Signage. Each project remains fully independent, and the WikiSuite community facilitates innovation, interoperability and visibility.

Join the Open Source community and be in control of your data!

Related links

- https://wikisuite.org/blogpost14-Budgeting-practices-for-major-software-initiatives-by-Waldo-Jaquith-18F-at-the-Michigan-Senate-Appropriations-Committee
- Cigref (An enterprise consortium) published a report “Open source, an alternative to major IT providers”: https://www.cigref.fr/cigref-report-open-source-alternative-major-it-providers-en-2018
- How CMS architecture affects dev communities - A case study of WordPress, Drupal, Tiki Wiki and XOOPS