I know this is all wrong, but fear it might be right - Alain Désilets

The idea for this panel came to me from my recent exposure to the TikiWiki development community. This is an Open Source project that seems to take a very wiki way to software development. It has a very large and vibrant community of developpers, many of which are not professional developpers.

As a strong proponent of the wiki-way, I thought this sounded like a great idea at first. But when I actually became an active participant in this community, every fiber in my professional developer brain started screaming bloody murder! But the more I learned about this community, the more I felt uneasy about the whole thing. I increasingly got the feeling that this is one of those things that, to paraphrase Dick Gabriel, "I **know** is wrong, but **fear** might be right".

The Tiki community is doing it all wrong!

As a developer with 25 years experience, and strong practitionner and evangelist for Agile and User-Centered development, I **"know"** the value of a strong discipline grounded in practices like Test Driven Development, Continuous Refactoring, good code crafmanship, and User-Centered thinking. Yet, when I look at Tiki Wiki, what I see is an apparent bloody mess. The project has no automated testing, and as far as I know, there isn't even a manual testing plan. Recently, a small group of Tiki developers put together an automated testing framework, but it is based on a Record and Playback approach which I know from experience will lead to brittle and unintelligeable tests. The code is a complete mess and is a case-book example of the "Big Ball of Mud" anti pattern. There are no classes, no modules, and very few functions. The basic building block is the script, and each of them are mostly monolithic, and receive their arguments from CGI arguments. The only good thing about it is that there is a clear separation between presentation and backend, using Smarty templates for the presentation. Looking at the user interface, I see that Tiki is very feature rich, but those features seem to have been developed with little if any thought about who will be using them, how, and for what purpose.

In short, Tiki is a bloody mess.

But might they be doing it right?

At the same time, it's difficult to argue with success.

In spite of everything that the Tiki dev community is doing "wrong", Tiki is a very popular wiki engine, and probably has the largest feature set of any wiki engine, or even Content Management System. The development community is very large, and very active, and more importantly, very diverse. Many of the "developers" on the Tiki team are closer to end users than developers.

The more I thought about this, the more I realized that my reactions to the "fly by night" nature of Tiki development were very similar to the reactions that scholars and professional writers have when they are first exposed to Wikipedia: they look, tend to (at least initially) frown at Wikipedia and see it as a bunch of amateurs who are doing it "all wrong". Yet, we know that there is something **terribly right** about Wikipedia, and that the only reason why it seems wrong is that it is using a completely new paradigm to

content authoring.

Could it be that there is also something **terribly right** about the Tiki way to support community development?

Looking at automated testing for example. While Record and Playback automated testing tends to yield tests that are brittle and hard to understand, it does have the advantage that pretty much anyone (not just developers and testers) can generate tests that way. Thus, a Record and Playback approach might allow a large community to quickly generate a relatively exhaustive suite of automated tests, which sure would beat the current situation of no tests at all. As far as code architecture goes, could it be that the flat, monolithic structure is actually easier to modify by novice programmers than a beautifully crafted object oriented cathedral? Could it even be that the messiness of the code results in more contributions by acting as an invitation to fix it? On the UI front, could it be that lack of cohesiveness and polishing is just the "small" price you have to pay in order to get a hugely feature rich system like Tiki?

Let's talk about this

So instead of brushing away Tiki-style community development as "amateurish", I find myself very much sitting on the fence and pondering questions like these:

- What are the advantages and disadvantages of this kind of approach
    - For example, there has been long periods of time between Tiki releases (more than 18 months). Could it be that the situation is spiraling out of control?
        - marclaporte: That could be related. Releases in Tiki are much more work & coordination because all the code/features are bundled (vs the we'll-release-the-core-and-let-3rd-party-developers-update-their-stuff-later approach). But mostly, the release-when-it's-ready (at the macro-level) was a huge (non) decision and it was a failure/huge risk for the project. Lesson learned, as part of Tiki model, we now release twice per year.

- What kinds of projects lend themselves to this sort of approach
    - Obviously, you wouldn't want to build a flight control system that way. But between a project like Tiki and a flight control system, there is a lot of stuff. Which of that stuff COULD be done in a community way?
        - marclaporte: Stuff for which you have a large community, and a culture or participation. Again, the fact that everyone has access to Tiki core (almost 400 people with commit access) plays a part.

- Could we get the best of both worlds?
    - Is it possible to allow a large community to contribute code in this wiki way, yet make it possible for others to "refine" this raw material into something better and cleaner? If so, what kinds of architectures and processes would be needed to support this kind of thing?
        - marclaporte: Yes, I think there would be places to improve. The same way wikis empower content creators to become webmasters, senior developers could improve various aspects of Tiki to make junior developers and theme designers more efficient.
            - One such improvement:

For commits to the stable branch (Ex.: 3.1 -> 3.2 -> 3.3), they must be approved by the Quality Team. For the next major version (ex.: 3.x to 4.0), all contributors commit directly to the main code base. We feel this is a great balance of

- having lots of innovation and low coordination overhead when developing the future version.
- while having minor versions with as few regressions as possible.

So if you need a lot of stability you should skip .0 releases (which really, you should avoid for any project) and have an upgrade pattern of 3.1 -> 3.2 -> 4.1 -> 4.2 -> 5.1, etc

So, I am very much interested in talking about this with anybody who cares!
ML: I think you should read ~~www.marclaporte.com/TikiSucks~~ moving to model and think about how the all-in-one design, vs the more common small-core-and-tons-of-3rd-party-plugins/exensions modularity approach used by bitweaver and others.

Related:

- Software Development the Wiki Way

Alias

- I know this is all wrong but fear it might be right
- I know this is all wrong, but fear it might be right