Advice for alternatives

If you are building something like WikiSuite, here is some advice.

WikiSuite started in 2011. Since then, some alternatives have surfaced, evolved and some have died. Looking back on this period, here are some lessons learned from the successes and failures of us and others.

Do your proper due diligence

You will invest a crazy number of hours in this. It is a wise investment to do thorough research into all the alternatives. Is there really nothing out there which corresponds to your vision? Could you work with an existing project and help take it to the next level?

Be in this for the long run

End users will have all their data in this. You need to do this having in mind to support it for the next 20+ years

And thus, you need to plan for the long run, and be mindful of technical debt

Think about the project as a whole

It's not just the code. You'll be running a full-fledged organization.

Here is an example to illustrate the various roles and responsibilities in a large software project: https://tiki.org/Teams

Recommend reading:

- https://producingoss.com

Don't underestimate the integration challenges

Think of the flow from the end user perspective as they move from feature to feature. For example, for WikiSuite, we have been working with components to migrate to Bootstrap to facilitate user experience harmonization.

Don't underestimate the size / scope of such a project

Please see Constructive Cost Model COCOMO

Think beyond installing

Many of the alternatives focus on being a platform where it's easy to install already mature software. This is good but not sufficient. You need to have a plan for integrating users, groups and permissions, search, etc.

Know when to build vs integrate

Developers want to jump in and code. If you build it, it will be more what you want and better integrated. If you integrate, you'll get faster results, but some overlap and integration issues.

Be careful: it's not just building it, it's supporting it for the next 20+ years

Pick all major components before you commit to them

WikiSuite was announced in 2011. There were several iterations before getting to today's list of components. Sometimes, changing one component has a cascading effect. Once you have users and data in one system, it will be very difficult to have them follow you in major changes.

Get involved in your components and make sure they succeed

Think ahead 10 years. It is likely that some of the components you have picked will run into difficulties. Make sure to be involved at a certain level to participate and help them strive. Think and work "upstream first". And these communities are the most likely to join you, as you are offering them complementary solutions.

Deal with the overlap

- Tell end users what to expect. What is supported, what is not.

Be prepared to compromise and accept the trade-offs

Many hard decisions are about trade-offs. Which interests will you sacrifice in order to make things better overall?

For example, in WikiSuite, we don't support Multitenancy. Each project should have its own virtual machine. This has many benefits and drawbacks. We live with that choice.

Be prepared to make such trade-offs.

Respect and enhance standards vs the freedom to innovate

An example of a trade-off: Using established standards offers many benefits, but it can constrain you. And the standards process can be very very slow.

Ex.: Tiki uses SVG-Edit as a Drawing tool. It's great because SVG drawings can also be worked on by another tool. However, it limits us to what is in the spec. To have more features, we later added Draw.io (later renamed diagrams.net) as Tiki Diagram. diagrams.net has way more features, and a better UI, but it doesn't use SVG.

Another trade-off: Supporting IMAP (more compatible) vs JMAP (more features and performance) or both (more

work)?

Be aware of business models

Various areas of an enterprise suite have different dominant business models. For example, e-mail systems tend to look for a subscription model (x$ per account per month) while in business consulting, the model tends to be hourly rates. In a unified system, these two models clash.

- In a subscription model, the service provider wants to interact with the customer as little as possible. It should all "just work".
- In a consulting model, the value is created by the interaction. Ex.: Improving systems and processes to reduces inefficiencies in work practices

And users don't appreciate when a tool goes from Open Source to proprietary SaaS:
https://mindtouch.com/resources/mindtouch-core-and-platform-this-is-the-end-beautiful-friend

Be wary of the viability of a hosting service

A hosted service is super important for adoption. But it's also very risky with a lot of mostly fixed costs. If you offer a free service, put a time limit (ex.: 120 days). So this is plenty of time to try, and attracts clients who intend to eventually pay if they are happy with the service.

Please see:

- https://sandstorm.io/news/2018-08-27-discontinuing-free-plan
- https://sandstorm.io/news/2019-09-15-shutting-down-oasis
- https://indieweb.org/site-deaths

Explain how you are different

It is to be expected to have a handful of major players in any market segment. They have a different drive / vision / DNA. And this will drive different design decisions and diverse outcomes for end users.

For example, in the case of WikiSuite, we explain it at alternatives and we make clear design choices for tons of integrated features because we believe it's the best way to avoid the pitfalls of http://pluginproblems.com/