ApacheCon North America 2017

From dev@ to user@ via the Apache Way

Lessons on the front lines of the ASF Incubator
Podling Progress

During the incubation of Podling Streams I've observed four general areas where a project must develop to become self-sufficient.

Vision
Code
Documentation
Team

All are essential, none can be neglected, but this list is not comprehensive.

For each area we'll discuss a few key dimensions of maturity, why each is important, and characterize phases of progress.

I'll offer some examples/anecdotes from my experience with Streams, and some tactics for improving I learned during my crash course in the apache way.
Vision

Mission

What are the project's goals?

Framing

How well is the mission explained?
Vision

Mission

What are the project's goals?

Projects entering incubation all start from a unique place. We see fully baked platforms, off-shoots of other projects, single purpose apps, a grab-bag of barely related features, or just an idea with little or no implementation.

Regardless the project needs a mission to grow and flourish.
When proposing a project, the fledgling PPMC may think they have a bulletproof plan. However, months or years in there may be competing visions for where the project is going.
Vision

Mission

Unknown

Uncertain

The project team will certainly disagree on the many aspects of implementation. They will probably also disagree on the project's scope - what the project should try to do, and what it should avoid doing. These discussions are part of the process of refining the Mission and carving out a niche in the OSS ecosystem.
Vision
Mission

Unknown

Uncertain

Explicit

It's important to ultimately determine the larger goals, scope, and limits of the project and state them explicitly. You'll achieve more capability-wise if the team's time and attention are focused on the most unique things you do. Explicit mission helps hone and maintain this focus.
Vision

Framing

How well is the mission explained?

It's essential that the vision can be summarized in a single sentence. Leave technology and solutions out of the vision.
Vision
Framing

Complicated

When you are trying to solve a multitude of problems, it's likely that aspects of your solution will tend to leak into the vision. By all means enumerate everything the project does, highlighting how your project is distinct from alternative and complementary projects, but don't get carried away on your landing page or in your initial user flow.
Vision

Framing

Complicated

Rambling

Your project's reason for existing will take more than a sentence, probably more than a page to describe. If it seems to a novice or an outsider to be a collection of bullet points rather than a compelling narrative, it will not resonate as well.
Vision

Framing

Complicated

Rambling

Coherent

Structure the essentials of your 'vision' content into a narrative with pictures and real-world examples, or into an outline / multi-page site with a strong index, and place pictures and real-world examples throughout the tree of content. Also, work to ensure that every piece of the vision described has a working module (even if rough) in the code base.
Code

Licensing

Is the code legit Apache?

Functionality

Does the code work?

Synergy

Is it one code base or many?

Comprehensibility

Can noobs understand it?
Code

Licensing

Is the code legit Apache?
Code
Licensing
Illegitimate

Out of the gate many projects do not meet minimum code standards for an Apache release.
Getting a release through the IPMC vote is a significant milestone in itself. Getting your license house in order for graduation is another.
Code Licensing

Illegitimate

Releasable

Auditable

Your project's source code should be demonstrably free of any licensing issues.
Code

Functionality

Does the code work?
Code

Functionality

Broken

There are apache projects (not just in incubator) that go extended periods of time with compile and/or test phase failures in master. To an outsider, this reflects very poorly on the project, regardless of root cause or duration.
Project code is officially working when there are unit tests in every module that get run routinely, and that standard is applied to all new code submissions.
Projects with solid code have automated continuous integration and address build issues brought to light immediately. They probably have integration tests that check end-to-end workflows for proper function. Features that aren't being maintained or aren't keeping up are scrutinized and sometimes deleted.
Code
Synergy
Is it one code base or many?
Fragmented

While a project is being assembled initially, not all the parts fit all the time. Groups of modules will be incompatible with other groups. The same function may be implemented in multiple modules. Interfaces aren't yet stable, or don't exist although they should.
During incubation you'll find modules you don't need and interfaces you desperately need. You'll go through many refactorings which hit tens and hundreds of files. You'll eventually get all the code connected, with few orphaned modules or fully stand-alone classes. Until you get there, the interest you generate in the project has a huge hurdle to overcome to convert a new user or committer.
An integrated code base has exhaustively considered interfaces, with predictable class and package names, and patterns for configuration, logging, instantiation, testing, etc., that are highly versatile and common across the project. This is worth doing. Smart projects are more fun to dig into because they teach you new tricks.
Code

Comprehensibility

Can noobs understand it?
Code

Comprehensibility

Confusing

Has anyone checked out a new code base and had the feeling of landing on an alien planet? You get glimpses of the familiar but on the whole you have no idea where, what, or why anything is the way it is.
Only once you come to understand the reasoning behind both design and implementation do you develop intuitions that are useful in diagnosing a problem or adding a feature. But projects can take steps to seem less foreign, as least to an apache committer.
While each newcomer still has to explore for themselves, there's a noticeable difference between projects where understanding the design and implementation reasoning comes easily and those where it does not. Consistency matters a lot. Try to push your project one step toward consistency in design and implementation with every issue you close.
Documentation

Availability
Are there decent docs?

Utility
How helpful are they?

Recency
Are they up to date?

Aesthetics
Are they visually appealing?
Documentation

Availability

Are there decent docs?
Documentation

Availability

Missing

You need a project website from day one. You can get away with boilerplate copied out of the proposal and links to your code, but not for long. If your website sucks, expect to hear about it. Take that criticism as seriously as any. The developers and users you want to recruit by-and-large will not tolerate of missing or crap documentation.
You've achieved documentation level one when your website contains details about everything your project is and does, your README(s) go beyond licenses and one sentence summaries, and your project's web site and other artifacts rank toward the top of searches for your project and code-related keywords. However, the docs you have will not be the docs they want, most of the time. Documentation assumes certain use cases and motivations, and those will be in flux for some time. You'll need to keep creating, updating, and reorganizing docs constantly.
Your project's documentation should anticipate everything that a user will want to know, and present the right information in response to the expected search terms. Devs are going to google your packages, classnames, and exceptions - you want them to land on properties you control. JavaDocs are good for that, but unless your team annotates your code the page might as well be blank. Many techies will chance upon your project via GitHub, so you'll need to create README(s) there and they ought to be helpful.
Documentation

Recency

Are they up to date?
Documentation

Recency

Ancient

Sometimes incorrect or stale documentation is worse than no documentation. Documentation that goes stale even a few versions may be so wrong that a newbie fails and gets frustrated trying to run the software at all.
Even small oversights can lead to problems and reflect poorly on the project.
Documentation Recency

Ancient

Dated

Current

Try to make updating documentation standard practice during code reviews, and perform a survey for incorrect content leading up to and during the release process.
Documentation

Utility

How helpful is the documentation?
Documentation

Utility

Irrelevant

Does your documentation actually tell the world why they should use the project? Help a new user get up and running seamlessly? Explain the line between what it does out of the box and what it can do for power-users who code to its interfaces and internals?
Documentation

Utility

Irrelevant

Helpful

Docs are your first line of defense and your best chance to win advocates. Assuming your code can provide user value quickly, your docs explain exactly how to make that happen.
The parts of your code geared toward other programmers likewise need marketing - when and why are your libraries the best choice? Why does the system work the way it does? Why is it awesome? What improvements are on deck and why?
Documentation

Aesthetics

Are they visually appealing?
Ugly

Some of the tools we use daily are just ugly. We get used to it, but isn’t it a breath of fresh air when you find a new tool that isn’t? You don’t want new users thinking ‘ick’ as they encounter your project’s brand, you want them thinking ‘oooh’.
Don't settle for just bare bones Apache CMS site. Pick a platform that lets you pick from and then tweak an existing theme with header menus at a minimum. Create pages that explain the purpose of the project holistically, with pictures as well as text. Write an FAQ. Getting Started. Example(s).
Documentation

Aesthetics

Ugly

Acceptable

Engaging

Invest the time in themes, stylesheets and platforms to make your website visually appealing. It affects your reputation, even if you think it shouldn’t. Project utility aside, human persons are simply more likely to share links to pages we enjoyed looking at. As it happens, web crawlers care as well.
Team Presence

Is anybody there?

Coordination

How is work getting done?

Planning

How are you deciding what to do?

Experience
Team

Presence

Is anybody there?
Team

Presence

Absent

This is certain to happen at some point. Everyone's busy, disconnected, or unmotivated. Find a specific topic to spur discussion, or goal to galvanize commits. Then find another.
Even with best intentions from all, email threads and pull requests can take longer to wrap up than we’d often like. Sometimes prompts offline can keep things moving. If that doesn’t work, call lazy consensus.
Team Presence

Absent

Distracted

Engaged

Activity renews interest. Keep the focus on short, medium, and long term milestones. And always be recruiting for new participants.
Team

Coordination

How is work getting done?
Participation is great, but what to do when team members are working in silos, or at cross-purpose? When everyone wants to get code merged fast instead of get good code merged first?
Team Coordination

Isolated

Integrating

To break out of these patterns, try to reach a quorum about where to focus next, starting with the upcoming release. Create an agile board, order your backlog, and group issues into upcoming release sprints. Assess difficulty of features, debate assumptions and implications before coding.
Discusses the implications of each feature request and develops an appropriate implementation and testing approach up-front. An integrated team treats every new feature as an evolution of pre-existing patterns, or an opportunity to implement a design that can been carefully considered.
Team

Planning

How are you deciding what to do?
Random

Code submissions come in ad-hoc, aren't associated with tickets, or are associated with low quality tickets. Email threads mostly debate why things are the way they are whether the ought to change.
Team Planning

Random

Tactical

Teammates are collaborating to get tickets resolved, and get releases out the door. Decisions are made on list, written up, and then shift into persistent reference materials. When questions come up, sharing links is suitable to answer repeat questions. Tickets contain enough details of the work to be done that they can be picked up and attempted by anyone.
Sprint objectives and release dates are planned strategically in advance. Deliberately group changes into major, minor, and maintenance releases depending on the degree of change needed to implement them correctly, rather than rapidly.
Team

Experience

Specifically, with the Apache Way
Team

Experience

Novice

Maybe your team has great coders, but they don't know anything about open source.
Team

Experience

Novice

Learning

The core team will have to teach them, patiently, with a mix of public and private communications. It's perfectly normal for learning moments to occur on-list, but some messages are best delivered, or best received, without an audience.
Add folks to the PPMC who already understand the Apache Way, even if they aren't likely to commit as heavily as others. The number of mishaps and missteps will shrink and observers will learn and emulate best practices.
Thank You!

people.apache.org/~sblackmon

sblackmon@apache.org