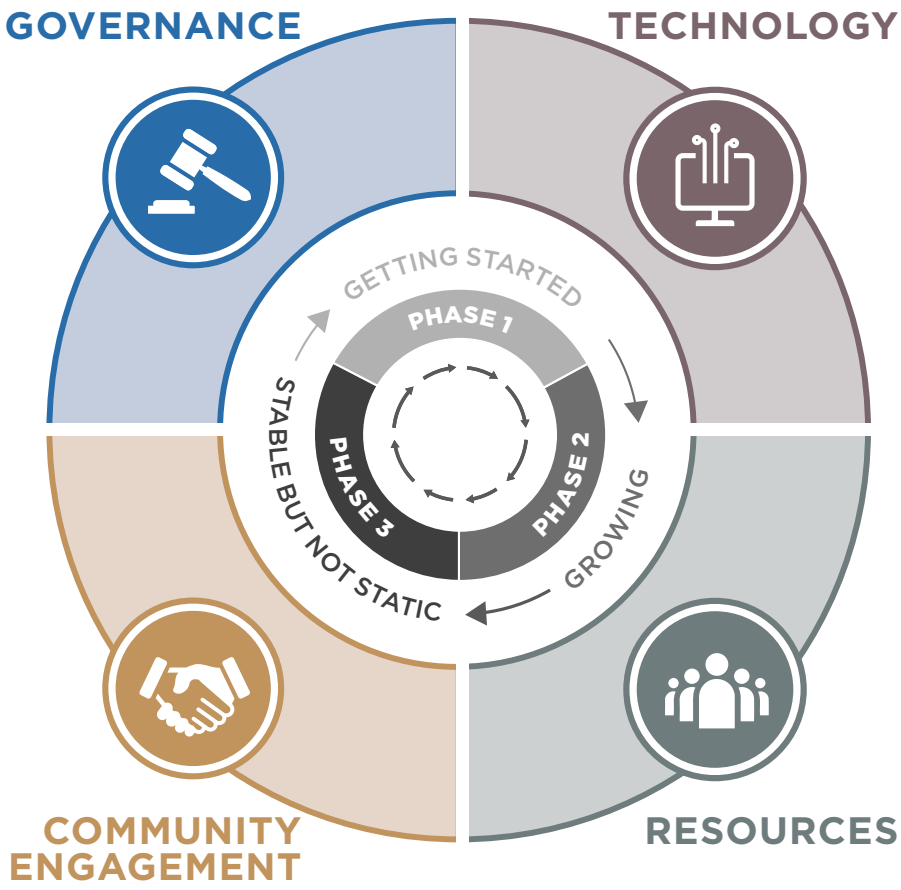
A Guidebook for Programs Serving Cultural and Scientific Heritage

Sustainability Wheel



GOVERNANCE

Phase I: Establishing
Working with original engineers, project staff, or organization. Go to page 11.

Phase II: Stabilizing
Functional but limited in one or more aspects. Go to page 12.

Phase III: Evolving
Strong management structures; not necessarily formal governance. Go to page 13.

TECHNOLOGY

Phase I: Laying the Groundwork
In design, pre-release or early beta testing phase; small set of early adopters. Go to page 20.

Phase II: Expanding and Integrating
Have more than one public release. Go to page 21.

Phase III: Preparing for Change
In production, well-adopted, supported. Technology stack stable. May be looking to next generation. Go to page 22.

RESOURCES

Phase I: Creating Consistency
Funded by single organization, grant-funded or volunteer operated. Go to page 26.

Phase II: Diversification
Distributed resourcing; meeting expenses, small number of revenue streams. Go to page 27.

Phase III: Stable, but not Static
Diverse staff support and income streams; focused on long-range strategy. Go to page 28.

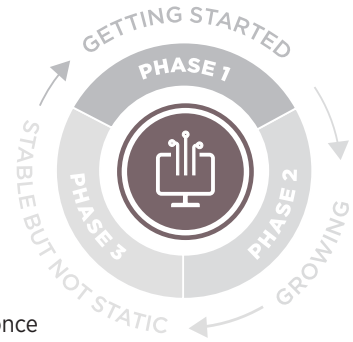
COMMUNITY ENGAGEMENT

Phase I: Getting Beyond Initial Stakeholders
Focused on primary stakeholders; lack of engagement with broader communities. Go to page 32.

Phase II: Establishing CE Infrastructure
Determining how to facilitate engagement that works for community. Go to page 33.

Phase III: Evolving CE
Established infrastructure to enable engagement. Go to page 35.

Facet: Technology



Phase I: Laying the Groundwork

Core Goal

Turn an idea for an application into a viable product that serves the needs of the community.

Characteristics

Programs in Phase I are in the design, pre-release, or early beta-testing phase of software development. These programs may have no users yet, or a core of committed early adopters or beta testers. New development may also be based on newer or unproven technology, require staff training, and may exhibit considerable technical or resource challenges.

Concerns and Roadblocks

Programs in the early phases often suffer from the need to be all things to all people – in order to get funding, they often promise the moon to sponsors. This leads programs in the early phases to be very susceptible to scope creep. A focus on trying to cram in every last feature may leave critical elements behind, such as testing, documentation, and community building. It can also be difficult to accurately assess the amount of time new development will take in a new environment.

Moving Forward: Objectives

- **Understand core community needs**

  OSS for cultural and scientific heritage is often developed in response to a specific institutional or community need. Programs should evolve from working within a single organization to gathering input and feedback from the broader community. This feedback can help define community-based functional needs, influence the architectural approach, and help refine core needs that require coordinated development. Programs can gain community confidence by articulating a broader vision; regularly releasing small, solid updates that allow funders and stakeholders to visualize the bigger picture; communicating how feedback influences development; and by focusing on overall quality.

- **Continue to gather data**

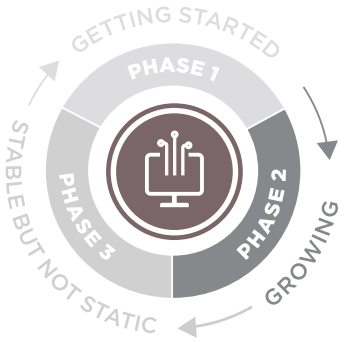
  A community needs analysis does not end once a program moves from design to development. Reach out directly to users. Continue to have conversations with the end users of applications. While it may be too early to ask for input on software improvements or new features and functionality, community members can provide valuable feedback and engagement by assisting with testing and documentation.

> Early openness with stakeholders and other investors will provide a good foundation.

- **Communicate process and progress with stakeholders**

  Museums generally do not let people view exhibits until they are completely installed. Archivists prefer to process a collection before making it available to researchers. Until fairly recently, scholarly data was not made available until the journal article was published. Contrary to these approaches, the best OSS development is open and transparent. Program staff need to counteract the tendencies of subject matter experts to play things close to the vest during design and development. By using an open code repository, public bug tracking and regular releases, OSS developers can inspire confidence and engage stakeholders. This kind of transparency may be somewhat counter to the culture of wanting to present completely finished work, but early openness with stakeholders and other investors will provide a good foundation for opening up the program to the wider community in future phases.

Facet: Technology



Phase II: Expanding and Integrating

Core Goal

Refine the application: identify and strengthen areas that are working well, identify gaps that can be filled with new features and functionality, and phase out elements that are not working.

Characteristics

Phase II programs have had more than one public release, developed a formal release process that includes a numbering system or other method for identifying major and maintenance releases, and the application is being used in production outside of the founding organizations. Programs are generally adding new features and functionality to their software packages and exploring integrations with related applications.

Concerns and Roadblocks

Once an application has been developed and released, it can sometimes be difficult to evaluate it with an objective point of view. Making the decision to deprecate or redesign features that took several sprints to design and develop can be complicated, especially if the features were championed by important project stakeholders. Programs that do not engage with their communities at this phase run the risk of developing features the community does not care about, and can be seen as only serving their own interests.

Long-lived OSS programs spend as much effort on the process of producing code as they do on producing code itself.

Moving Forward: Objectives

- **Engage the community**

  Community involvement in the requirements gathering and functional specification process is paramount. Sitting down, either physically or virtually, with the people who use the application frequently can provide development teams with a clearer view of what is working, what features and functionality are most heavily used, and how the application may be improved or expanded to better fit user needs.

- **Grow thoughtfully**

  Once an application has been released and a community of users begins to grow, the program team must learn to balance community feedback and interest in exciting new features with maintaining stable, up-to-date, and well-documented software. Programs that can communicate clearly about architecture and infrastructure can form a common understanding with the community of the importance of backend maintenance and support. It is also important during this phase to cultivate the community of developers and committers (with commit rights) outside of the core organization and stakeholders. Outside contributors add not only valuable code to the application, but also new perspectives that keep the program from becoming an echo chamber.

- **Consider integration over new development**

  We have communities and we are a community. There are many organizations working to develop open source solutions to address cultural and scientific heritage problems, and it may be that one of the problems an OSS program needs to solve has already been tackled by other members of our community. Leveraging existing open source solutions can not only add functionality, but also open up a program to a new set of users, developers, and stakeholders. Instead of using scarce resources to develop new functionality which may or may not be ancillary to the software's core purpose, explore if integrations with existing platforms with appropriate functionality can serve this function. It may be possible to increase the sustainability of the core product, especially if these ancillary platforms have significant user communities, development communities and strong governance. This leveraging of other communities allows the program to grow in functionality and potentially serve new audiences without having to necessarily invest a large amount of resources.

  Invest in testing, documentation and training. Long-lived OSS programs spend as much effort on the process of producing code as they do on producing code itself. Robust and efficient testing, documentation, and training (both of developers and end users) are critical to scalability and sustainability.

Facet: Technology

Phase III: Preparing for Change

Core Goal

Determine how the core application's technology stack and functionality will serve the future needs of the community; plan ahead for expansion, integration, re-architecture, or retirement.

Characteristics

Phase III applications are in production, well-adopted, and well-supported. Design and development of the core technology stack is stable, with few changes to the application's architecture with each release. Programs typically have a stable supply of developers and committers, and a published and predictable release schedule. Program staff in this phase are generally looking to the next generation of the application. The existing application may be nearing the end of its useful life due to changing market circumstances or require a technology overhaul to bring the code up to date with new technology or community needs.

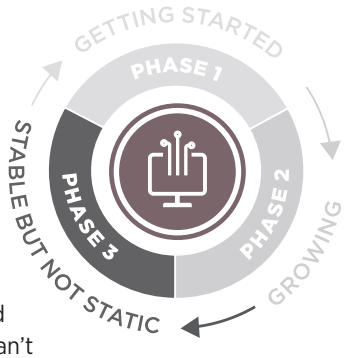
Concerns and Roadblocks

Some community members may feel comfortable with the current platform, it is stable and has been proven as a production-ready application for some time. For others, Phase III can feel like a return to the drawing board. New communities and stakeholders or technology obsolescence may require re-architecting or retiring elements of an application. Program staff must balance the needs of stakeholders invested in and comfortable with earlier versions with the need for significant refresh and potential expansion to new communities.

Moving Forward: Objectives

- **Reassess community needs**

  The demand for software re-architecture or retirement must come from stated community requirements, balanced with the community's ability to support and keep up with change. Program staff must ask themselves how re-architecture or retirement will serve the community. Are there things users would like to accomplish but can't with the current architecture? Are things fine the way they are but underlying technology is sunsetting and must be replaced? Is there an opportunity to migrate current users to an OSS application built on newer technology? Users of OSS for cultural and scientific heritage rely on these applications to care for information held in the public trust, and must be part of any decision-making process that would affect their ability to create, maintain, and preserve that information.



- **Plan for evolution**

  Once the need for change has been identified, the community needs to review whether incremental improvements to the OSS application are sufficient or whether a complete refactoring and re-architecture is required. If the core requirements that inspired the original development of the application still exist, but the language, libraries, or hardware platform used to create the application are obsolete, it may make sense to refactor or re-architect the application. It is sometimes the case, however, that requirements have evolved, and at the time of refresh, additional functionality or a fundamental restructuring is needed. Thinking ahead rather than waiting for crises allows program staff to get buy-in from the community, secure necessary funds, and develop transition and migration plans for existing implementers.

- **Document an exit strategy**

  Sustainability is not synonymous with perpetuity. There are cases where a program has been successful, but served its purpose, and should be gracefully retired. Programs that no longer meet the needs of their communities or have been supplanted by alternatives may need to develop plans to communicate the end-of-life decision to the community and organize support or migration services for remaining users.

> Sustainability is not synonymous with perpetuity.

Technology Resources and Tools

- Dombrowski, Quinn. "What Ever Happened to Project Bamboo?" *Literary and Linguistic Computing*, Volume 29, no. 3 (2014): 326–339.
- Fogel, Karl. *Producing Open Source Software: How to Run a Successful Free Software Project.* Beijing: O'Reilly, 2009. http://producingoss.com/.
- Ries, Eric. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* New York: Currency, 2017.
- Rosenberg, Scott. *Dreaming in Code: Two Dozen Programmers, Three Years, 4,732 Bugs, and One Quest for Transcendent Software.* New York: Three Rivers Press, 2008.

Software documentation examples:

- "Avalon Media System Documentation.:" Avalon Media System. Accessed 1 February 2018. http://www.avalonmediasystem.org/documentation.
- "Koha For Developers." Koha Community. Accessed 1 February 2018. https://koha-community.org/get-involved/for-developers/.
- "Samvera: Developers." Samvera Community. Accessed 1 February 2018. https://wiki.duraspace.org/display/samvera/Developers.

Technology Case Studies

Guidebook case studies provide first-hand accounts from forum participants about their program's work toward sustainability. Technology case studies are from the Fedora and LOCKSS programs.

Fedora

By David Wilcox
http://fedorarepository.org/

The first public release of Fedora (version 1.0) was made available in 2003. Through a combination of grant funding and community contributions the software matured over time; version 2.0 was released in 2005 and 3.0 in 2008. But like most software projects, a considerable amount of technical debt built up over time as a distributed community continued to build on top of a now-aging codebase, and by 2012 it was time to consider a major project re-architecture. This initiative, dubbed Fedora Futures, focused on five key priorities:

- Improved performance, enhanced vertical and horizontal scalability;
- More flexible storage options;
- Features to accommodate research data management;
- Better capabilities for participating in the world of linked open data; and
- An improved platform for developers—one that is easier to work with and which will attract a larger core of developers.

These priorities represented challenges based on the then-current version of Fedora, but the Fedora Futures initiative also provided an opportunity to re-think the Fedora software based on lessons learned and emerging technologies and standards. Early on, the development team decided to focus on a robust REST-API built on top of an existing open source software platform, thereby reducing the amount of custom code the Fedora community would need to maintain. The API would also be aligned with modern, well-adopted web standards, such as the Linked Data Platform, which would help Fedora move beyond the walls of the library into the world of the web and linked data. These decisions provided great opportunities for the Fedora project and community, but there were also several challenges to overcome.

The biggest challenge of a complete software re-architecture is how to support the existing community of users. Specifically, many institutions were already using Fedora in production, often with client applications that were built based on expectations of functionality that would change in Fedora 4. A considerable amount of community energy has been put into supporting migrations, including tooling, documentation, metadata mapping, and training. However, migrations are often an institutional resourcing problem as they inevitably take considerable, dedicated effort. Supporting migrations continues to be a high priority for the Fedora community as we try to move everyone forward to the latest version of the software.

Fedora 4 has now been in production for over three years, and our focus has shifted toward stability. Ideally, Fedora is a dependable piece of infrastructure that works well and doesn't change very often. To this end, we are committing to a slower release cycle of only one major release per year, and publishing a formal specification of the Fedora REST-API that will provide additional stability for client applications.

"The biggest challenge of a complete software re-architecture is how to support the existing community of users."

Technology Case Studies

LOCKSS

By Nicholas Taylor
https://www.lockss.org/

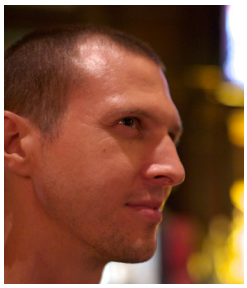


Photo: Ben Chernicoff

For nearly two decades, the Stanford University LOCKSS (Lots of Copies Keep Stuff Safe) Program has supported community-based, distributed digital preservation through its eponymous software. Changes in the larger technical environment in the intervening time have lately prompted a major re-architecture effort, currently underway with substantial funding from the Andrew W. Mellon Foundation, with the goal of bidirectional integration of LOCKSS with the broader ecosystem. This move will support the sustainability of the LOCKSS Program by broadening the communities that are sharing costs to maintain functionality upon which the LOCKSS software depends.

> "The gains to sustainability from the re-architecture have as much to do with community strategy as with technical insight."

The LOCKSS software was originally developed in the nineties, at the inception of web archiving by memory institutions. Like other web archiving applications of this era, e.g., the archival crawler Heritrix and archived web content replay engine Wayback Machine, the LOCKSS software evolved into a complex, monolithic Java application. Significant developments in web technologies in the ensuing two decades motivated technical evolution in web archiving. Though the LOCKSS software confronts similar challenges as the broader web archiving field, its architecture has heretofore incentivized implementing independent solutions.

Recognizing otherwise missed opportunities for alignment with extant community initiatives and the long-term sustainability risk posed by a siloed software stack, we are now modularizing the major functionalities of the LOCKSS software into a set of interoperating web services. This will novelly enable existing open source software to be leveraged as part of a LOCKSS system, reducing maintenance costs and simplifying adoption of new technologies. Conversely, it will also allow for the incorporation of individual LOCKSS software components – e.g., the peer-to-peer data integrity and repair mechanism – into non-LOCKSS systems, unlocking the potential for more flexible integration and a broader impact.

These objectives underscore that the gains to sustainability from the re-architecture project have as much to do with community strategy as with technical insight. We have a strong sense of the need to find, align with, and invest in the broadest possible open source software communities focused on our shared challenges if those challenges are to be addressed both effectively and efficiently. We need to further build, engage, and learn from open source software communities with a stake in the unlocked functionality of the LOCKSS software to maximize the good that it can provide for digital preservation broadly.