Earth Science Data Systems
Policy for Open Source Software Governance

Purpose
Governance Model Guidelines
Introduction
What is a Governance Model?
The Value of Open Source
The Challenges of Open Source
Internal vs External
The Governance Model
Roles and Responsibilities
Support Infrastructure
Code Repository
Documentation
Issue Tracking
The Earthdata Code Collaborative (ECC)
Communication
Mailing Lists
Decision Making Process
Contribution Process
Metrics
Technical Case Study - Earthdata Search
Technical Case Study - OODT
Governance Model Template
General Overview
Roles and Responsibilities
Project Manager
Project Management Committee
Committer
Contributor
User
Support Infrastructure
Code Repository
Documentation
Issue Tracking
Communication
Decision Making Process
Contribution Process
Code
Documentation
Design Work
Purpose
This document provides guidelines and a template for developing a governance model for any Open Source Software (OSS) project within the NASA Earth Science Data Systems (ESDS) Program ecosystem. It has been developed and shaped by a number of stakeholders across the ESDS community with the aim of providing a consensus of how OSS Governance should be addressed.

Governance Model Guidelines

Introduction
Congratulations! If you're reading this, then you have most likely received approval for an open source license on your software project. Whereas that license is the legal framework that will enable collaboration on your project - between centers and with the public, the governance model is the social framework. It is the fundamental element in building a community around your project so that you can gain all of the benefits that an open source license makes possible.

There are two parts to this document; the first is the Governance Model Guidelines (GMG), which you are reading now, and the second is the Governance Model Template (GMT), which is designed to make it easy for you to develop your own project governance model and to provide criteria for ensuring it is successful. These guidelines provide context for the decisions that you will need to make, and the template contains clearly delineated [guidance] on the commitments you will need to make and populate for your specific project. Together the two items provide you with a roadmap for developing a governance model, which will then be the procedures by which you manage the community surrounding your open source project.

What is a Governance Model?
The governance model for an open source software project defines the way in which the project is managed, and provides details for contributors (and potential contributors) on the roles that are available to them and how they may become involved. Making this information easily available and understandable is a critical component of building a community around your project and thereby leveraging the benefits that open source brings. This document lays out a governance model that balances the hierarchical constraints, code review standards, and legal obligations that come with being a NASA project, while providing maximum flexibility for building and managing this community in a way that is relevant to your specific project.

The Value of Open Source
A properly implemented open source software project has the potential to benefit the original developer of the software as well as a broad base of additional users. In addition, open source development requires transparency and openness, which is in alignment with NASA’s values.

The primary benefit to the original developers of an open source software project is to have a passionate community assisting with the development, improvement, and maintenance of the
software. This shared responsibility lowers development costs and improves both stability and functionality. In addition, many developers find the open source development model to be more productive and efficient than traditional methods. This derives from the transparency and openness that all developers on the project have to the code base, the roadmap for project development, and the list of current issues.

As others reuse your software for their own needs, you will find that not only do they benefit from your work, but the project itself will improve in response to their bug reports and feature requests. This is a ‘virtuous cycle’ which makes the software more attractive and helps to build a larger user base.

In addition, open source is in alignment with ESDIS goals of encouraging software reuse - both for the benefits described above but also in order to make best of use of public funds spent on software development. Naturally, these benefits do not happen with every open source project but having a governance model will greatly improve the likelihood that it will happen for your project.

**The Challenges of Open Source**

At this point, your largest challenge in gaining the benefits of open source is making certain that you put the infrastructure in place for it to be a successful project. That's what this document is about. By combining your own expertise and knowledge of your software project with the guidance in this document, you will be in a good place to gain those benefits outlined in the “The Value of Open Source” section.

The second challenge is growing community around the code you write. Depending on your project, you may have an already established community that you will just need to organize, or you may be a solo developer or team who is hoping to build a community around your newly-released project. Either way, the larger the community that you are able to build, the greater your benefit from releasing your project as open source. Use this governance model as a foundation on which to build your efforts, and the following questions as the initial steps to guide your community building:

1. Who else might be interested in your software? Don’t just think of individuals, think of existing communities, sectors, and industries.
2. Where do these people meet or discuss things? Do they have trade shows, meetups, or online forums?
3. Are you already part of these communities? If not, how do you join them?
4. Are there high-profile or well-respected individuals within these communities that you can contact directly?
5. How can you publicize your release to these communities?

Remember that communities grow through mutual respect and contributions. If you are not yet part of a community but think they would be interested in your software, you will have to gain
their respect through your contributions. The way that you approach and engage with these communities makes a big difference in how they receive your project.

Finally, there are some technical considerations that will make it easier to develop an active community who uses and helps to maintain your software. The ESDS Software Reuse Working Group published a guide called “Reuse Readiness Levels” that will help you understand the value of each of the characteristics and more clearly evaluate your readiness to make your software available to others for reuse. They defined nine characteristics to evaluate on:

- Documentation
- Extensibility
- Intellectual Property Issues
- Modularity
- Packaging
- Portability
- Standards Compliance
- Support
- Verification and Testing


Internal vs External
The division between internal and external is one of the most difficult concepts for veteran developers that are used to working in a closed source environment to adjust to within an open source one. For a traditional NASA software project, there is no separation that needs to be made - it’s all internal and any issue that arises with the software, be it in the codebase or specific to a single implementation can be handled the same way. However, when developing in collaboration with a community, this distinction becomes very important. If the software has a bug, it needs to be reported and resolved through the external community. If there is an issue with a specific implementation of the software inside of NASA, e.g. there are hardware problems or compatibility issues with another proprietary piece of software, then that needs to be resolved via internal communication channels.

The Governance Model
The governance model that you are developing is based largely on the model that the Apache Software Foundation uses. This model was deemed most relevant to ESDS for a number of reasons, including its success in a wide diversity of project sizes and types, the flexibility in governance that it leaves to individual projects, and the fact that the Apache Foundation has, similar to NASA, certain legal obligations that it requires its projects to meet. See here for more details on the Apache model: https://www.apache.org/foundation/governance/pmcs.html.
This document assumes that your main code base and development efforts will happen on a public repository, that your community will include (or grow to include) members of the public, and that your project will be managed by its community. Without these assumptions being met, it is difficult to see the value of licensing the project as open source.

Within NASA, there will also need to be support infrastructure established for private communication and documentation of the specific implementation(s) of the software.

**Roles and Responsibilities**
The governance model defines the roles that individual community members may fill within your project. It is important to remember that everyone that touches the project is a member of the community, including your internal development team.

These roles and their responsibilities are detailed in the Governance Model Template and include the Project Manager, the Project Management Committee, Committers, Contributors, and Users.

**Support Infrastructure**
The point of providing infrastructure for your open source project is to enable collaboration between community members and provide support for end users and contributors. Each of the tool types is required to build a healthy community.

**Code Repository**
This is the place where your code will be stored so that anyone can access it. Github is the recommended choice. NASA has a public Github repository, as do some Centers. Visit [http://code.nasa.gov](http://code.nasa.gov) to find the most current information.

**Documentation**
We recommend that you use a public wiki tool for managing your documentation. Along with your Github repository, you will get wiki capabilities, but if you have more complex needs you will want to investigate other options as well.

For internal (private) implementations of software, you should use the Earthdata Code Collaborative tools: [https://ecc.earthdata.nasa.gov/](https://ecc.earthdata.nasa.gov/).

**Issue Tracking**
Publicly, you need a good issue tracker. Again, Github provides this capability as part of your repository but if you are used to something else or have complex need, you will need to investigate other options.

**The Earthdata Code Collaborative (ECC)**
ESDS provides a suite of tools for shared development projects called the Earthdata Code Collaborative (ECC). It is expected that your initial hosting will be with the ECC, unless you already have alternative resources established. Depending upon how your project develops, it
may then move on to other resources, but this provides at a minimum an incubation environment to simplify your early development stages and, at a maximum, all of the tools that you will need.

The ECC provides a GIT-based platform to host and collaborate on development of tools, services, or service endpoints. This service is available to EOSDIS-affiliated developers and DAACs. ECC provides versioning and code control via a GIT repository with Atlassian Stash integration. Similar to “Github”, Stash provides “clone”, “fork”, and “pull request” capabilities to ECC. Additionally, all projects within ECC can use Jira for issue tracking and Confluence to manage a project-specific wiki. By unifying powerful tools within a sleek and simple user-interface, ECC facilitates discovery and collaboration on tools and services across EOSDIS and with other partners. The ECC can be found at https://ecc.earthdata.nasa.gov/.

Communication

Mailing Lists
Mailing lists are the main way that ideas are communicated and because you are building a geographically diverse community, they should be considered the official communication for all project discussions and decisions.

Public Development: This is the main email list for all members of the community to share information, discuss proposals, solve problems, and determine the future direction of the project. This is also the official location for decision making. Any proposal regarding the governance or development of the project should be submitted here where it can be discussed and voted upon by the community. While only members of the Project Management Committee have official votes, unofficial voting on this list (eg, a response email with an “agree,” “yes,” or “+1” in it) makes it easy to discern community opinion. Additionally, having a public vote of this nature has sufficient legal backing behind it and holds up to jurisprudence.

Private Development: While use of it should be minimized, occasionally there are sensitive development issues that need to be discussed. This list includes all Project Management Committee members and all Committers on the project, and may be used for items such as security concerns that are not yet known to the public.

Project Management Committee Private: While it should be minimized, this list is for items that the Project Management Committee will need to discuss in private, like inviting community members to be committers or to join the Project Management Committee.

Decision Making Process
Beyond what is described in the Template, it bears clarifying that while the Project Management Committee is ultimately responsible for the project and what it does, it’s primary responsibility is to facilitate the process and make certain that all members of the community are treated fairly and with respect.
**Contribution Process**

Another difficult thing when transitioning from traditional NASA development models to open source is making certain that your internal developers understand the contribution process and are engaging with it in the same way as any public community member would be expected to. This is particularly important when you are just building your community because it sets the tone for how things will work as the community develops around you. If your internal development team doesn’t follow the procedures laid out in the governance model, that will become obvious to external contributors as they begin to engage with your project and it will damage your community building efforts in the long run.

**Metrics**

In order to assess the vitality of a project, the Project Management Committee should capture the following metrics on a monthly basis and make them available, at a minimum within the ECC:

- Number of code contributions, broken down by relevant divisions such as:
  - funded contributions, contributions from government employees, and contributions from the public
  - accepted vs rejected
- Length of time to commit submitted contributions to the code base
- Number of community members, including how many PMC members, committers, contributors, and users - and their division in terms of government or public
- Number of emails sent on the public email list

These numbers will not be useful in terms of judging a project’s viability as compared to other projects, but over time will provide insight into the development of the project, the health of its community, and its long term sustainability as an open source project.
Technical Case Study - Earthdata Search

In order to provide a tangible example of how to utilize the Earthdata Code Collaborative (ECC) in combination with public open source software development, Patrick Quinn, the development lead for Earthdata Search provided this case study of their experience:

Earthdata Search started as a private ECC project, using a Stash repository, a JIRA issue tracker, Bamboo builds and deployments, and a Confluence wiki. When we received open source approval, we wanted to continue using most of our ECC tools but also make our source code publicly available on GitHub. We needed to work GitHub into our existing processes, in particular:

1. Scrum-based agile development. We make extensive use of JIRA’s agile tools and prefer them to GitHub’s offerings
2. Verification before merge. We ensure code passes all build checks before merging into the baseline. We need to be able to do this for both internal and external contributions.
3. Continuous deployment through Bamboo. Bamboo automatically deploys our code to our SIT environment when our build checks pass.

Beyond those key pieces of process, we wanted to maintain a simple development environment and avoid tool fragmentation. The solution we settled on was to use GitHub instead of Stash as our primary working repository, to avoid needing to sync contributions from multiple places. We maintain a copy of our code in the ECC Stash and our builds sync GitHub changes back to Stash to ensure its copy remains up-to-date, allowing better integration with JIRA and Bamboo. Our builds use Travis CI, which is a continuous integration service available for free to open source projects, however our build steps maintain compatibility with Bamboo. We opted to use Travis CI because Travis CI is able to build GitHub pull requests. Our use of the remainder of the ECC is unchanged. With these pieces in place, the steps from completing code to deploying code are straightforward and work the same regardless of the source of the code:

1. The author of the code (internal or external) commits code to GitHub and opens a pull request against the main Earthdata Search repository
2. The Travis CI server detects the pull request and automatically runs build checks to verify it
3. The maintainers review the code and make comments on the pull request
4. The author makes any necessary changes based on the build and the team’s comments
5. Steps 2-4 repeat until the maintainers are satisfied with the change
6. The maintainers merge the code into the baseline
7. The Travis CI server detects the change to the baseline and runs build checks to verify it, which should pass at this point
8. Upon a successful build of the baseline, the code syncs to Stash
9. Bamboo detects the change in Stash, creates a new deployment artifact, and deploys the artifact to the SIT environment
Note that most steps are automated. Only authoring, code reviews, and merging require any manual intervention. We have been using this process for a couple of months and so far have found it to be simple to work with and very similar to the process we had before the switch to GitHub. We would recommend this approach for teams wishing to work with a repository outside of the ECC, including GitHub, while still being able to take advantage of the great deployment, planning, and documentation tools that the ECC offers.
Technical Case Study - OODT

OODT’s development and open source release pre-dates the ECC but serves as a relevant case study in how an open source project can be integrated into a NASA center’s Software Management Plan. It was developed internally at JPL and upon open source release approval, it entered the Apache Incubation process, becoming a top level Apache project in 2010.

OODT utilizes Apache’s infrastructure, including the code repository, documentation in the Apache Confluence wiki, issue tracking through Apache’s Jira instance, and communication for the community on mailing lists @oodt.apache.org. As part of the Apache Incubation process, OODT received mentorship from veteran Apache members to help build the initial community and ensure that all of the necessary infrastructure and groundwork was laid to give the project the best chance for success. This included establishing the full governance model as well as thorough test and build procedures.

Since OODT continues to be used inside of JPL on multiple projects, it was also necessary to document a Software Management Plan (SMP) that would outline how JPL procedures interoperate with the Apache open source process (this document is included as “OODT SMP 20111122pg-cm.docx”). In brief, each of these projects are responsible for their specific implementation of OODT, as well as contributing their code contributions back to the open source project. By utilizing a standard JPL SMP to explain how the open source project operates, internal adoption of the project became easier and more actionable for those not familiar with the process. The SMP clearly delineates what is the responsibility of the specific project that is utilizing OODT, and what is the responsibility of the open source process that manages OODT.
Governance Model Template

General Overview
This policy details how this project is managed, including full details of how anyone can get involved in it and what they can expect if they should do so. It provides explanations of the different roles and responsibilities, the support infrastructure, the decision making process, and how to make contributions.

Roles and Responsibilities

Project Manager
The Project Manager is the person responsible for the project at NASA and are the official communication channel between the formal structures of the agency and the governance structures of the software project itself. They are a regular member of the Project Management Committee, with the same power and responsibilities as all other members.

Project Management Committee
The Project Management Committee (PMC) is the governing body of the project. It is composed of the Project Manager and other members of the project community.

The responsibilities of the PMC are to:
1. ensure the project is meeting all of its legal requirements and that the defined procedures are followed.
2. manage and protect the community surrounding the project. Most importantly, this means ensuring that there is balanced peer review and collaboration within the project that includes all members of the project community.

With the exception of the Project Manager, all other roles may be filled by community members that have shown their commitment to the project through their contributions, and it is the PMC that extends these invitations. As all members of the PMC are also active members of the larger community, it is part of their responsibility to identify and support individuals whose work improves the project. A tangible way of doing this is to give these individuals more responsibility by inviting them to become a committer or to join the PMC.

Committer
Committers are responsible for reviewing submitted code contributions to make certain that they meet all of the Code Review procedures established for the project. They accept pull requests and incorporate them into the main trunk. This role may be filled by more than one person, and is even a position that an individual from inside the community may be elevated to by the PMC. A member of the PMC must propose the new member, which is then approved by a majority of the PMC members. No-one has the ability to veto a vote.
Are there any legal requirements we should add in here? eg apache requires committers to sign a Contributor License Agreement, to make sure there will never be any issues legally: http://www.apache.org/licenses/#cls

Contributor
Contributors include anyone that provides an input for the project. This includes code, documentation, graphics, designs, or anything else that tangibly improves the project.

User
Users use the software and provide feedback in the form of bug reports and feature requests.

Support Infrastructure

Code Repository
[ Where is it? Who has access to it? ]
Download it. Install it. Use it. Fork it. Fix it. Share it.

Thank you.

Documentation
[ Where is it? ]
Everything you ever wanted to know about [ Project name ]. If it isn’t here, then please help us add it.

Issue Tracking
[ Where is it? ]
Bug reports? Feature requests? Usability issues? Please report them here! If your issue has been reported previously, you may even find a solution to it.

Communication
Public Development: [ Details and how to sign up ]
This is the official communication channel for activities related to the development of this project. If you want to get involved, the first step is to join.

This is where everyone shares information, discusses proposals, solves problems, and determines the future direction of the project. It is also the official location for decision making.

Private Development: [ Details and how to sign up - if you don't have one, delete this section ]
While use of it should be minimized, occasionally there are sensitive development issues that need to be discussed, such as security concerns that are not yet solved and should be made public. This list includes all Project Management Committee members and all Committers on the project.

Project Management Committee Private: [ Details and how to sign up ]
While it should be minimized, this list is for items that the Project Management Committee will need to discuss in private, like inviting community members to be committers or join the Project Management Committee.

Real time chat: [Details if you have one, including “office hours” if that’s relevant]

Decision Making Process
The PMC is responsible for decisions and direction of the project, but often in consultation with the community. In those rare situations where the PMC is forced to make a decision that is unpopular with the community, they will provide justification and explanation for it.

Any member of the project community may make a proposal for consideration. There is no restriction on the subject of these proposals, including the future direction of the software, the way in which the project is governed, and the allocation of available development resources.

A proposal should be submitted on the Contributor email list for the project, where discussion can then follow. It is the PMC’s responsibility to ensure that all member’s ideas are heard and respected and that the discourse is civil. Anyone agreeing or disagreeing with the proposal can “vote” for it by responding to the list with their vote. The votes may or may not include discussion items, but if they are intended as a vote should specify that with something obvious like “agree” or “yes” or “+1” (or their opposites). This voting is all unofficial but makes it easy to discern community opinion for everyone involved.

Contribution Process
Thank you for your interest in contributing to [Project Name]. There are many ways that you can get involved and we’ve tried to provide everything you need to know below to make it easy. This includes writing code, improving the documentation, and doing design work. If you haven’t yet, make sure you sign up on our development email list [link]. Even if you only have advice to share, we’re happy to have you on board!

Code
[Provide introductory information, including how your development cycle works (if you have one), what support resources are available to help developers get up to speed on the codebase, and anything else that might be helpful when first starting to engage with the project.]

Coding Standards: [If you have these, detail them or link to them if they are long]

Code review process: [If it’s simple, put it here, otherwise link to it. It would be nice to include an estimate of how long it normally takes to accept pull requests into the main trunk]

Documentation
We are thrilled to have improvements to our documentation. It is one of the things that software projects are notorious for lagging behind on. We have endeavoured to make this as easy as possible. See here for more details [link to a page in your documentation about how to do it].
Design Work
If design is your thing but you don’t have the coding ability to build it yourself, we are so happy to have you! For starters, join the development email list [link] and share what it is that you’re interested in working on. If you have already mocked up something or have something specific that you want to work on, also submit it as a feature request in the issue tracker [link]. Then it will enter our regular development pipeline.