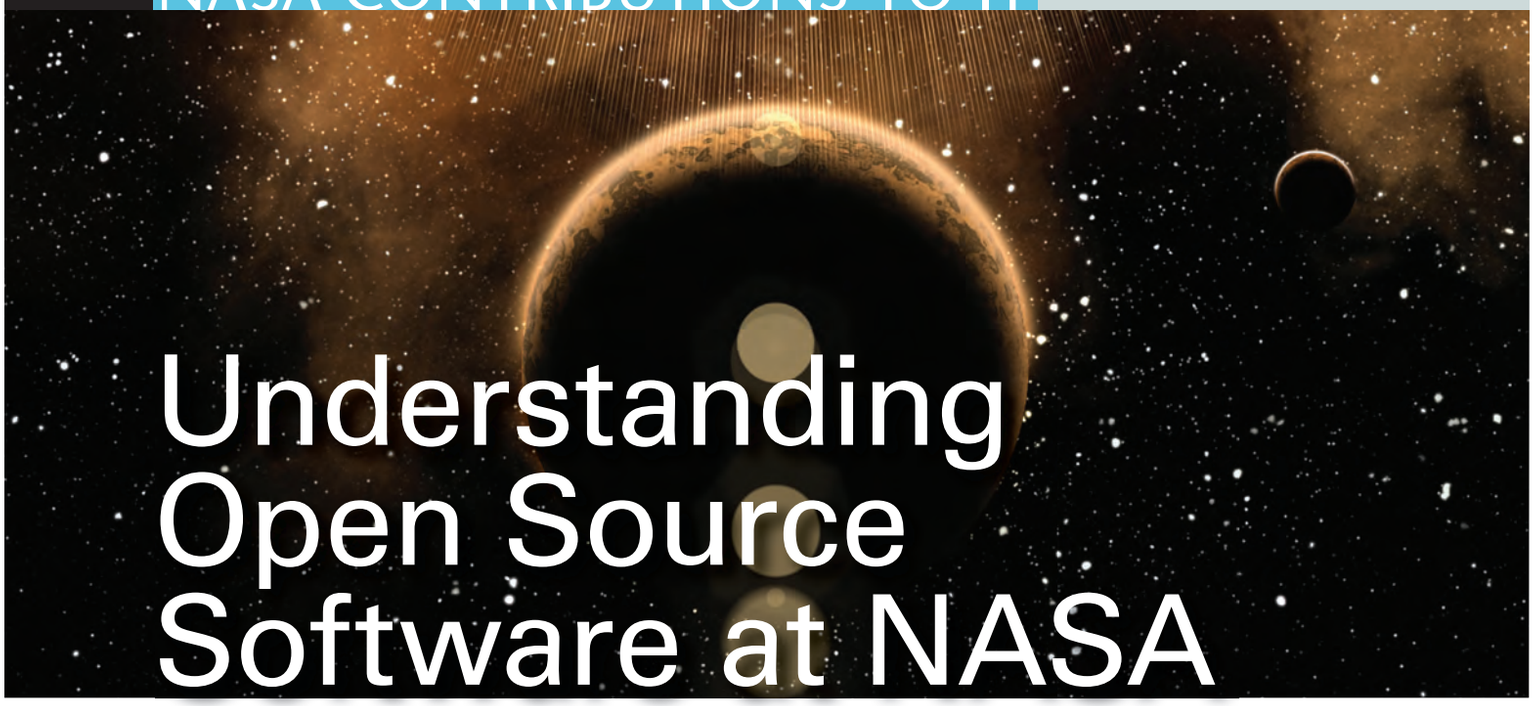

Understanding Open Source Software at NASA

Chris A. Mattmann, Daniel J. Crichton, Andrew F. Hart, Sean C. Kelly, Cameron E. Goodale, Paul Ramirez, and J. Steven Hughes, *NASA Jet Propulsion Laboratory*
Robert R. Downs, *Center for International Earth Science Information Network*
Francis Lindsay, *NASA Goddard Space Flight Center*

To provide a framework for comparing and understanding open source software at NASA, the authors describe a set of relevant dimensions and decision points that NASA and other government agencies can use in formulating an open source strategy.

The phrase "open source" has steadily gained traction at the National Aeronautics and Space Administration (NASA) over the past decade. The agency, as a whole, has evolved from developing specific, single-community software projects—for example, spacecraft and instrument control software—to developing and deploying reusable software systems with broad-based applicability, such as the Apache Hadoop software framework,[1] the Earth Science Information Partners (ESIP) open data representation standards, and general-purpose compression libraries for data formats. In fact, one of the most well-known open source compression libraries, zlib, is maintained by Mark Adler, who works at the NASA Jet Propulsion Library (http://zlib.net).

Although open source software is becoming mainstream, NASA offers comparatively few examples of its projects. The open.nasa.gov site, a recent effort to promote the agency's open source software and the current US White House administration's Open Government Initiative,[2] links to two of the agency's center-specific open source sites, opensource.gsfc.nasa.gov (46 projects total at the NASA Goddard Space Flight Center) and opensource.arc.nasa.gov (23 projects total at the Ames Research Center).

The lack of publicized open source software production underscores a deeper issue: the lack of a general framework for understanding the use and production of open source software at the agency level. Specifically, NASA needs policies that outline acceptable open source software licenses, suitable methods for attribution and redistribution, related community development efforts, and legal and intellectual property issues. NASA hasn't universally defined

1520-9202/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

these issues as an agency or at its individual centers.

Our own experience over the past five years in trying to convert the Jet Propulsion Laboratory's (JPL) Object-Oriented Data Technology project[3] away from a center-specific project into a broader, community-based open source effort at the Apache Software Foundation (ASF) demonstrated the need for a framework for understanding open source. When we tried to "open source" our project, we entered uncharted territory regarding software licensing (JPL had its own license, held jointly with the California Institute of Technology, and NASA had another), redistribution (JPL had its own hosting site; we wanted to use Apache), and development modes. The lessons learned navigating these issues motivated our desire to develop an agency-appropriate framework for understanding open source decisions at NASA.

When we tried to "open source" our project, we entered uncharted territory regarding software licensing, redistribution, and development modes.

Open Source at NASA

NASA has a rich history in developing and using open source components and systems. From 1990 to 1994, its open source development activities served as an example for industry. For example, the Beowulf project was instrumental in breaking down the barriers to using open source;[4] it connected a collection of commodity computers using the Linux operating system.[5,6] By 2000, the benefits of open source software development practices at NASA were more broadly acknowledged,[7] and by 2003, NASA had leveraged open source software on high-visibility projects, including the Mars Exploration Rover.[8]

NASA Open Source Agreement

NASA attempted to formally codify its open source principles of transparency, participation, and collaboration in the NASA Open Source Agreement (NOSA; www.opensource.org/licenses/nasa1.3).[9] NOSA is a legal document, akin in many respects to the GNU Public License, although it's less restrictive. The Open Source Initiative has certified NOSA as a valid open source license.

Although NOSA indicates the growing importance of NASA's open source movement, the need for yet another open source license is questionable in light of widely used and well-understood alternatives such as the General Public License (GPL), Lesser GLP, Apache License version 2, and the Berkeley Software Distribution (BSD) license. Furthermore, potential conflicts between the NOSA and GPL licenses contribute to a sense of ambiguity when regarding merging software developed under the different licenses.

World Wind Visualization Software

World Wind is a NOSA-licensed visualization software package distributed by the NASA Ames Research Center that provides an interactive digital globe akin to Google Earth. Users can overlay data on a spherical surface and manipulate the view in real time to create visualizations. In addition to Earth, World Wind supports other major and minor solar system bodies, increasing its utility.

The World Wind software suite debuted in 2004, predating Google Earth by almost nine months.[9] Since that time, it has seen a dramatic transformation from a self-contained desktop application to a lightweight, Java-based development kit that supports a growing ecosystem of plug-ins, applications, and tools. (See the related article in this issue, "Contributions to IT: A View from Ames Research Center.")

Nebula Cloud Platform

Nebula is NASA's answer to an open source cloud computing environment. The Ames Research Center initially developed it in 2008 to increase the use and efficiency of compute resources while reducing energy consumption by establishing infrastructure as a service.[10] Nebula's IaaS is similar to Amazon's Elastic Compute Cloud (EC2) and Simple Storage Service (S3) technologies.

IaaS enables projects large and small to rapidly acquire precisely the compute resources needed for a task and, when the task is complete, return those resources to the general pool. Nebula was

begun after NASA looked into several cloud computing services and determined that the terms of service and security considerations didn't meet NASA's scientific computing requirements.[11]

NASA Open Source Summit

NASA held its first ever Open Source Summit (OSS) at Ames Research Center in March 2011. This two-day event invites participants to share experiences and ideas for the successful adoption and integration of open source software. Participants represented not only NASA but also other US government agencies, including the Department of Defense, and commercial partners, such as Google, RedHat, and Mozilla. The OSS recommendations[12] dovetail with our open source framework dimensions. We presented an early version of our framework at the NASA OSS (see www.slideshare.net/ckleclerc/nasaoss-mattmann) and also at the 2011 Summer ESIP Federation Meeting (see http://wiki.esipfed.org/index.php/Summer_2011_Meeting).

The Open Source Appraisal Framework

Our framework is a cognitive toolkit to help Managers, policy makers, testers, architects and other software system stakeholders better appreciate open source software's place in NASA projects, assess its use, and consider deployment options. Figure 1 shows the three dimensions of this open source software framework.

Licensing Open Source Software

Because reproducing software is so easy, it doesn't follow the distribution model of tangible goods. Rather, software is licensed, such that an individual or organization can use it under a legal contract's terms and conditions. Typically, using the software indicates an acceptance of such terms.

For open source software, the license brings legal bearing over the software's redistribution. The license terms might affect whether the software can be freely re-shared. The license also dictates the ownership of the intellectual property that comprises the software. The license might impinge on who retains ownership of enhancements and modifications made to the original software by other developers, and it could dictate the terms under which those enhancements and modifications can be redistributed. It might also control what can and must be said about the software's use in the NASA context. For example, the license might forbid mentioning the software within a NASA promotional lithograph, or it might require revelation of the software each time the program is started.

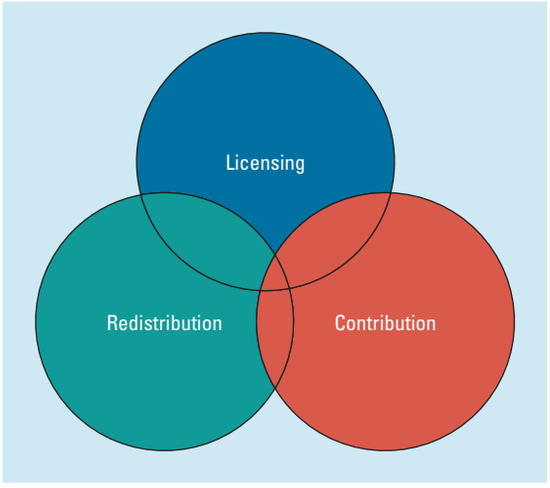

Figure 1. The three main components of the open source appraisal framework.

Let's consider a specific example. BSD is well-recognized open source license developed for the Berkeley Software Distribution of Unix. It lets NASA choose whether to redistribute the software. If NASA chooses to do so, it must include the license itself in the redistribution. However, NASA can't use the names of the authors in any endorsements or promotional materials without first getting permission. Finally, NASA's own contributions to the software have no intellectual property restrictions.

As a contrasting example, consider the GNU GPL, which is currently the most popular license for open source software, despite being far less permissive than the BSD license. For example, it allows redistribution, but only under the identical license terms. Enhancements and other modifications to the software must fall under the aegis of the GPL if they're ever redistributed outside their original NASA context.

For completely original software developed for a NASA-specific project, the choice of license becomes vitally important and intrinsically linked to the unit, division, site, or laboratory that developed it. A center's technology transfer office could, for example, limit the types of open source licenses that other developers can brand onto NASA open source software to avoid discouraging commercial development of such software. Software with the BSD, MIT, and Apache licenses, for example,

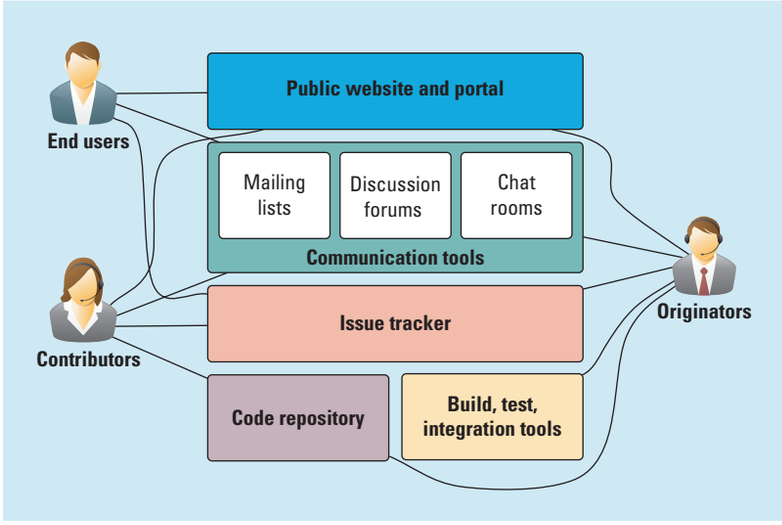

Figure 2. Redistribution of open source software.

are more commercial friendly, giving private companies quite a bit of leeway with the programs and what they can do with them, whereas the GPL requires company-developed modifications to remain as open source, "copyleft" software, thus destroying the value of privately held company-developed enhancements.

Other factors to consider in choosing the license include the ways outside-of-context and non-NASA contributors can modify the software and under what intellectual property restrictions and international limitations.

Redistributing Software as Open Source

Redistributing NASA software as open source has ramifications for many areas within the software development process (see Figure 2).

The days of distributing software on physical media have long passed on, especially for open source software. The venerable Emacs text editor has been available by FTP since the 1980s. If a NASA project so wishes, it too could set up a basic FTP or HTTP server for its software distribution needs. However, open source software typically requires more than just some way to get at the program. As shown in Figure 2, it also needs

- mailing lists or discussion forums;
- issue tracking for bug reports and feature requests;
- source code repositories and repository browsing tools;
- a public website and portal; and
- build, testing, and integration tools.

Many open source projects, both at NASA and elsewhere, elect to run such infrastructure themselves and exploit the many open source packages that implement mailing lists (such as Mailman), issue trackers (such as JIRA), source code (such as Subversion), website development and support (such as Plone), and build systems (such as Jenkins). When the infrastructure becomes unruly and gets in the way of open source development, external entities can provide many of the same features, including the ASF, Bitbucket, Github, Google Code, Launchpad, or Sourceforge.

These providers (and many others) have various advantages and disadvantages for NASA-originated open source projects. For example, all of the providers mentioned offer source code control facilities and file distribution features. Google Code blocks access by countries on the US Office of Foreign Assets Control sanction list. The ASF requires a vote on software releases and authenticated digital signatures on all software releases. Github encourages social aspects of code development and has straightforward ways of branching, modifying, and "forking" both the code and the communities behind them. Sourceforge hosts hundreds of thousands of projects, but many are no longer maintained.

In releasing original software as open source, NASA must decide on a model of public involvement to implement. We've identified two major collective models exhibited throughout the open source field.

Help-desk model. Under the help-desk model, the software originators release it as open source yet remain the final arbiters of what comprises the source code. The rest of the world forms a collective of "end users," who submit bug reports and feedback through mailing lists, forum reports, or issue trackers. Contributed code takes the form of patches against major open releases that the originators can, at their discretion, include in future releases. Originators can also invite major code contributors to join the originating organization. Sourceforge and Google Code implicitly support the help-desk model.

Community-building model. Under the community-building model, the software originators engage the collective in an in-depth way, forming a virtual organization. Patches are still accepted, but the community vets individuals with the ability to commit concrete changes to the software, and it decides what and when to release subsequent versions of the product through a formal process.

We can further divide this model into two submodels: one with a centralized, top-down management that provides the framework for overall direction of its product or product set, and another that uses decentralized management, requiring no overall steering force and allowing the community to amorphously choose its direction. Examples of the former include the Apache Software Foundation for the Apache family of open source products, the Plone Foundation for the Plone content management system, and the Eclipse Foundation for the Eclipse development environment. Examples of the latter include Bitbucket and Github.

Ultimately, the concerns that guide the model or submodel selection and internal versus external hosting environments will depend on the business process methodologies regulating the NASA software release into open source, the organization's desire for collective involvement and for retaining intellectual property rights, the support afforded by the infrastructure, the formal mechanisms by which future releases are made, and the adoption level needed for the open source program to survive.

Qualifying Contributions to Open Source

Open source software's most salient feature resides in its continuous operation as a functional concern that progresses, matures, and evolves over time. For this, open source relies on contributions.

Contributions take on a surprising morphology, as depicted by the different types shown in Figure 3. A true appraisal of open source software must comprehend this structure to qualify and, where possible, measure contributions and gauge the health of any open source project.

The contributions might include responses to questions about the software posted on the project's mailing lists or discussion forums. Archiving these contributions in a mailing list or forum and indexing for searching provides ad hoc system documentation.

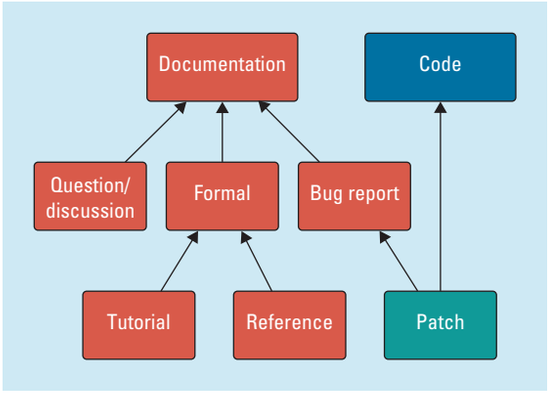

Figure 3. Different types of open source contributions at NASA, broken down into two families: documentation and code, with a patch blurring the line between the two.

Discussions introduced in virtual chat rooms, at conferences, and in person with project principals constitute another form of contribution. These discussions can spur the exploration of unusual corner cases, motivate new features, or even influence the overall direction in an open source's evolution.

Formal documentation can provide tutorials, reference material, API descriptions, FAQs, integration guides, and so on.

Open source software, like its commercial counterparts, undergoes formal releases. Releases are typically tagged with version numbers, and end users adopt and use specific versions for use in their own applications or as extension points for new projects. Engineering a release requires nontrivial effort, and such effort is required from both the community and from those who can actually modify the code for release.

The report or filing of a bug highlights a defect or missing feature within an open source package and thus serves as a form of documentation. When bug reports include patches, the submitted code constitutes a direct contribution to the code corpus.

That leads us, naturally, to the code itself. All of the contributions are important, but none displays a project's liveliness more than continual code contributions, which are the lifeblood of open source. Even successful programs and products with good utility tend to falter and fade when those contributions cease.

How an open source effort accepts code contributions depends largely on the model adopted.

For the help-desk model, code contributions might not occur at all, or only as a result of bug reports. Patches, if they're accepted, are often given intense scrutiny before they're committed to the code corpus. For the community-building model, patches are encouraged and can be reviewed well after they've been committed; direct commits by outsiders can occur as well, especially in the distributed management flavor of the community-building model.

These leads to two overarching models of code contributions.

Review-then-commit (RTC) model. In this model, code contributions arise from outside of the originating team's purview in the form of patches. The originating team reviews the patches and can, at its discretion, apply them directly to the code, adapt them in some way for the code, reject them with feedback to the contributors, or reject them outright with no feedback. The latest state of the source code is typically held sacrosanct and is often considered a complete, runnable, stable system.

How an open source effort accepts code contributions depends largely on the model adopted.

This model tends to rely heavily on the issue tracking system to manage the state of each patch; less emphasis is placed on automated testing. This model helps reduce the ego tied to individual contributors and, to a lesser extent, a team, because each review or patch cycle takes quite a bit of time. It also enables an easy metric: the number of accepted patches per contributor, which the original team can use to bring in new contributors as fully privileged committers.

Commit-then-review (CTR) model. In this model, code contributions can come from patches—that is, "pull" requests from external developers that have a complete set of changes to merge in—or from outside members who have permission to commit changes directly. The project's issue tracker plays less of a role because there are fewer, if any, patches to manage, but automated testing is far more vital to ensure a complete, stable, runnable system.

Although this model enables far more rapid turnaround time (and more rapid releases), it also poses a risk for creeping features that might not have been properly discussed or might even be wholly inappropriate to the overall system function.

The impetus to comprehend the dimensions and tradeoffs of open source software is emerging from its mainstream use within NASA projects. Our proposed appraisal framework, centered on licensing, redistribution, and contribution, is applicable to NASA as well as other government institutions considering the use of open source software.

Acknowledgments

This work was conducted at the Jet Propulsion Laboratory, California Institute of Technology under contract to the National Aeronautics and Space Administration. Support for Robert R. Downs was provided by NASA contract NNG08HZ11C.

References

1. T. White, *Hadoop: The Definitive Guide, 2nd ed.*, O'Reilly, 2011.
2. P.R. Orszag, "Memorandum for the Heads of Executive Departments and Agencies," M10-06, Dec. 2009; www.whitehouse.gov/open/documents/open-government-directive.
3. C. Mattmann et al., "A Software Architecture-Based Framework for Highly Distributed and Data Intensive Scientific Applications," *Proc. 28th Int'l Conf. Software Eng.* (ICSE06), Software Engineering Achievements Track, IEEE CS Press, 2006, pp. 721–730.
4. R. Comerford, "The Path to Open-Source Systems," *IEEE Spectrum*, vol. 36, no. 5, 1999, pp. 25–31; DOI:10.1109/6.763200.
5. M. Davis et al., "Linux and Open Source in the Academic Enterprise," *Proc. 28th Ann. ACM SIGUCCS Conf. User Services: Building the Future* (SIGUCCS 00), ACM Press, 2000, pp. 65–69.
6. T. Bollinger, "Linux in Practice: An Overview of Applications," *IEEE Software*, vol. 16, no. 1, 1999, pp. 72–79; DOI:10.1109/52.744572.
7. D.S. Katz and R.R. Some, "NASA Advances Robotic Space Exploration," *Computer*, vol. 36, no. 1, 2003, pp. 52–61; DOI:10.1109/MC.2003.1160056.
8. J.S. Norris and P. Kamp, "Mission-Critical Development with Open Source Software: Lessons Learned,"

IEEE Software, vol. 21, no. 1, 2004, pp. 42–49; DOI:10.1109/MS.2004.1259211.
9. D. Bell et al., "NASA World Wind: Opensource GIS for Mission Operations," *Proc. 2007 IEEE Aerospace Conf.*, IEEE Press, 2007, pp. 1–9.
10. "NASA Nebula Cloud Computing Platform," NASA Open Government Plan, NASA, 2 Oct. 2011; www.nasa.gov/pdf/440932main_Nebula.pdf.
11. J. Williams, "NASA Nebula in Action: Cloud Computing Case Examples," white paper, NASA, Sept. 2011; http://nebula.nasa.gov/media/uploads/nasa-nebula-in-action.pdf.
12. N. Skytland, "NASA Open Source Summit," slide presentation, 2011; www.slideshare.net/skytland/nasa-open-source-proceedings.

Chris A. Mattmann *is a senior computer scientist at the NASA Jet Propulsion Laboratory, working on instrument and science data systems on Earth science missions and informatics tasks. His research interests are primarily software architecture and large-scale data-intensive systems. Mattmann received his PhD in computer science from the University of Southern California. He's a senior member of the IEEE. Contact him at mattmann@jpl.nasa.gov.*

Daniel J. Crichton *is a principal computer scientist, program manager, and principal investigator at the NASA Jet Propulsion Laboratory. He oversees programs in Earth and planetary data systems and serves as JPL's thrust lead in climate research data systems. Crichton received his MS in computer science from the University of Southern California. Contact him at crichton@jpl.nasa.gov.*

Andrew F. Hart *is a staff software engineer at the NASA Jet Propulsion Laboratory. His research interests include user interfaces for Web applications, frameworks for rapid Web application development, and data modeling. Hart received his MS in computer science from the University of Southern California. Contact him at andrew.f.hart@jpl.nasa.gov.*

Sean C. Kelly *is a technologist at the NASA Jet Propulsion Laboratory through Northrop Grumman Corp. His research interests are in dynamic languages, rapid Web-application development, user interfaces, and autonomous systems. Kelly holds a BS in computer science and a BS in technical communication from the New Mexico Institute of Mining and Technology. Contact him at sean.kelly@jpl.nasa.gov.*

Cameron E. Goodale *is a software engineer at the NASA Jet Propulsion Laboratory. His research interests include user experience, using Web standards for geographic information systems (GISs), and open source data management. Goodale received his MBA with an emphasis in GISs from the University of Redlands. Contact him at cameron.e.goodale@jpl.nasa.gov.*

Paul M. Ramirez *is a senior software engineer at NASA's Jet Propulsion Laboratory, working on the Planetary Data System (PDS) and other science data systems involving both earth and planetary missions. Ramirez received his MS in computer science from the University of Southern California. He is an Apache Software Foundation member and committer on the Apache Object-Oriented Data Technology project. Contact him at pramirez@jpl.nasa.gov.*

J. Steven Hughes *is a principal computer scientist at the NASA Jet Propulsion Laboratory. His research interests focus on architecting and implementing system architectures in complex, distributed, heterogeneous environments. Hughes received his MS in computer science from the New Mexico Institute of Mining and Technology. Contact him at jshughes@jpl.nasa.gov.*

Robert R. Downs *serves as a senior staff associate officer of research and as the senior digital archivist in the Center for International Earth Science Information Network (CIESIN) of Columbia University. His research interests include the design, development, use, and evaluation of digital libraries. Downs received his PhD in information management from the Stevens Institute of Technology. Contact him at rdowns@ciesin.columbia.edu.*

Francis Lindsay *serves at the manager of the Earth Science Data Systems working group, a component of the Earth Science Data Information System (ESDIS) project at the NASA Goddard Space Flight Center. His research interests focus on NASA Earth sciences. Lindsay received his PhD in geography from the University of Maryland, College Park. Contact him at francis.lindsay-1@nasa.gov.*