2

Building the Next Generation of Aerospace Data Processing Systems by Reusing Existing Software Components

James J. Marshall[1], Robert R. Downs[2], and Shahin Samadi[1]
[1]*Innovim / NASA Goddard Space Flight Center,*
[2]*CIESIN, Columbia University*
USA

1. Introduction

Software is a key ingredient when developing any aerospace system. It is used in embedded electronics, in flight dynamics, in ground and space data processing, and in the current generation of data products. For example, the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center's Innovative Partnerships Program Office offers licensing opportunities for software and technologies from a variety of areas relevant to the hardware and software requirements of Earth and space science missions and projects: aerospace/aeronautics, computer software, data processing/analysis, electromechanical devices, electronics, manufacturing equipment, mechanical technologies, nanotechnology, optics and photonics, sensor and detector technologies, subassemblies and components, telecommunications and internet, and test and measurement (IPP Office, 2009). Reuse of existing experience and artifacts eliminates having to "reinvent the wheel" and is a key element to achieving progress in many areas of complex aerospace system development.

Originally, in the absence of vendor-provided solutions and commercial off-the-shelf software components, many data and information systems were designed and built as custom applications. However, as the practice of systems and applications development has matured, facilitating reuse of software and reusing previously developed software have been recognized as beneficial for meeting the challenges of developing and maintaining complex systems. Some of the challenges commonly faced by system developers can include dealing with very large quantities of data (e.g., terabytes per day), working with a distributed knowledge base, the expense and complexity of required technology infrastructure, and the need for domain-specific knowledge in software development (Samadi et al., 2007). In software development, reuse can assist today's development teams in various aspects of the system development life cycle, especially when they share common goals (Samadi et al., 2007).

The development of new systems can benefit from the efforts that contributed to the development of current and previous generations of systems. Considering the costs of building new systems, and the learning curve that contributes to such costs, leveraging the results of previous system development activities has the potential to reduce system

Source: Aerospace Technologies Advancements, Book edited by: Dr. Thawar T. Arif,
ISBN 978-953-7619-96-1, pp. 492, January 2010, INTECH, Croatia, downloaded from SCIYO.COM

development costs and improve capabilities for new development. Previously developed system components, such as plans, design requirements, system documentation, algorithms, and procedures, offer the results of previous design, testing activities, scientific algorithms, and learning experiences that were part of the initial systems development effort. Such legacy resources contain the evolved expertise of previous generations and can have continuing value if reused when building the next generation of systems or enhancing existing systems.

While the potential gains from software reuse appear quite promising, these gains also can be accompanied by costs and risks. Effectively preparing software components and other artifacts for potential reuse requires efforts to ensure that such artifacts can be reused in a manner that offers adopters efficiency gains that can be realized through reuse. Without such preparation efforts, adopters might not achieve the efficiency benefits anticipated for their reuse activities. However, the costs of preparing software for reuse could be small in comparison to the potential gains to be attained from reuse. For example, in cases where a planned or "forward-looking" approach to software reuse has been employed in the aerospace industry, improvements have been observed in terms of increasing the quality and reducing the costs of software reuse (Finnigan & Blanchette, 2008).

Similarly, risks can be incurred through software reuse. The adoption of previously-developed software can pose risks for those who contribute software as well as for those who adopt software. For example, assumptions about the origination, validation, and verification of software can increase risks to software projects, especially if the software affects mission-critical operations or safety-critical systems (Orrego & Mundy, 2007). While costs can be controlled and risks can be mitigated, they need to be recognized and assessed to determine their potential effects on software reuse. Recognizing the existence of the possible costs and risks also enables the involved parties to engage in software reuse with grounded expectations. Taking a deliberate, planned, and systematic approach to software reuse will help producers and adopters of reusable software to attain the benefits of reuse while limiting its costs and managing its risks.

Reuse has been shown to be effective in developing large and complex data systems to support space-based missions. One example is the National Polar-orbiting Operational Environmental Satellite System Preparatory Project where high levels of reuse have enabled system developers in the Science Data Segment, which provides ground systems that support data product assessment and quality verification, to reduce development effort as described by Samadi et al. (2007). Software reuse also contributes to flight software, such as the instrument manager software for the Mars Science Laboratory rover (Murray et al., 2009). This chapter describes how reuse activities can fit into building the next generation of aerospace data processing systems by providing guidance on methods for improving reuse practices in order to realize the benefits of reuse.

2. Background on reuse

The reuse of existing software and related artifacts is commonly referred to as software reuse. More specifically, the reuse of a software artifact is its integration into another context (Leyton, 2008). The reuse of software has been practiced informally and formally by software developers, since the practice offers potential benefits for developers and development organizations to attain improvements in productivity and software quality (Ajila & Wu, 2007; Mohagheghi & Conradi, 2007; Nazareth & Rothenberger, 2004).

Decreases in software product development time and software development costs also have been reported (Frakes & Kang, 2005). In addition, systematic software reuse could provide additional benefits if the organization establishes incentives to motivate developers (Sherif et al., 2006).

The results of surveying members of the Earth science community on their reuse practices and experiences showed that the reuse of existing technologies is most often performed in order to save time, save money, and ensure the reliability of the product (Marshall et al., 2006). When measuring potential gains in productivity, quality, and time-to-market from software reuse, metrics must be analyzed from the entire software development life cycle as well as from software maintenance efforts (Mascena et al., 2005). If such gains can be realized, then efforts to reuse software code and other artifacts should be pursued by software developers and their employing organizations. Likewise, such efforts should be documented to identify and measure actual benefits attained. Furthermore, measuring the benefits of software reuse can assist in benchmarking the activities that led to the measured gains and enable the identification of additional gains that can be compared to such benchmarks.

In addition to the potential benefits previously described, reliability is another objective of software reuse (Frakes & Kang, 2005). NASA recognizes the value of reuse in providing "a lower likelihood that the item will fail tests and generate rework" (NASA, 2007). Anticipating potential uses of software artifacts should help to reduce failures of reusing such artifacts within new domains and operating environments (Suri & Garg, 2008). In addition, efforts to ensure the reliability of software for reuse within new environments should begin during the early stages of the software development process (Immonen & Niemela, 2008).

Various software artifacts, in addition to source code, can be reused to the benefit of software developers and their organizations. Some reusable software components are listed in Table 1 (derived from Samadi et al., 2007, with additions).

Reusable Software Components
Operational source code
Analysis and design specifications
Plans (project management)
Data (testing)
Synthetic data generators and analysis tools
Expertise/experience (life cycle model, quality assurance)
Information used to create software and documentation
Testing procedures
Documentation
System and user guides
Use cases and user scenarios

Table 1. Reuseable Software Components

In order to make use of reusable components, information about "how" they work must be included along with information about "what" they do (Neighbors, 1992). Additional work

products, such as comments and other artifacts, also can contribute to the reusability of source code and are often needed to migrate software components between platforms, contexts, and projects. In many cases, documentation, such as user guides, installation guides, and other specifications, also is essential to facilitate the reusability of software components.

Establishing reusability as a requirement for the development of software components can contribute to their potential for reuse. However, even in cases where reusability was not planned or originally intended during initial software development, an artifact may still be reused in domains and contexts similar to the original. In such cases, domain analysis may be required to facilitate reuse, even when the reuse is within an application or problem domain that is similar to the original. Domain analysis includes identifying, capturing, and organizing information that was used in developing the original system to facilitate its potential reuse within a new system (Preito-Diaz, 1990), enabling subsequent evolution of the original information infrastructure.

3. Adoption process for reusable components

Reusing previously developed components within a new system requires a directed effort. In some cases, a top-down and single-project life cycle model such as the "waterfall" model might not be appropriate for software reuse (Simos, 1988). Software reuse is not inherently top-down, and it needs a perspective which may be beyond the development activities of a single project or program. The reuse process needs to follow a structured approach such as the approach recommended by Basili et al. (1989):

1. Specifying the object to be created
2. Searching the project, domain, and general databases for reuse candidates
3. Evaluating the candidates to determine which (if any) should be used
4. Modifying, if necessary, to fit specific needs
5. Integrating the reusable components
6. Validating
7. Feeding back the knowledge regarding the payoff of reuse

Initially, potentially reusable components need to be identified through domain analysis, and the candidates for reuse need to be assessed in terms of how they might be reused. For example, a categorized list of sub-routines might inform current development activities even if the subroutines are not considered to be reusable. The extent of reuse could depend on common aspects between the original functionality of the reusable components and the planned purposes for their potential reuse. Also, developers need to determine whether the candidate reuse would be cost-effective when assessing how much preparation is required for potential reuse. The more mature the software is, in a reusability sense, the more cost-effective its reuse is likely to be. Less mature software is likely to take more time, cost, effort, etc. than building the software from scratch, and is less likely to provide cost-effective solutions.

3.1 Locating reusable components

In addition to identifying candidates for reuse from current and legacy systems, other systems, either within the same organization or available externally, also should be considered when seeking reusable components. Developers from various fields are

contributing to open source collections of software that are available for reuse by other adopters. SourceForge (2009) is one example of such a collection, and the site covers essentially all domains. Other collections may be focused on one or more particular domains, such as the Netlib for mathematics (2009) and the Space Telescope Science Institute's list of software and hardware products for astronomy (2009).

There are also collections for particular programming languages, tools/formats, and applications; some examples of these are the Comprehensive Perl Archive Network (2009), the software available at the HDF-EOS Tools and Information Center (2009), and the Comprehensive TeX Archive Network (2009). While adopters are most likely to find reusable components for their particular domain, programming language, tool/format, or application in these more specific collections, they should check the more general collections as well, since some components may be suitable candidates for reuse even if they were originally developed for a different domain.

Often, these open source resources are offered with non-restrictive licenses that allow and encourage others to reuse the components. Open source communities also encourage developers to share improvements to open source resources so that the resources and the adopters will benefit from enhancements that have been developed and offered by others.

3.2 Assessing reusable components

Various tools are available for systems developers when assessing the potential for reusing legacy resources in the development of a new system. One example is reuse readiness levels (Marshall & Downs, 2008), which can be used to assess the potential for existing software being considered for reuse. Reuse Readiness Levels (RRLs) also can be used to assess the value of software that is being developed and to determine areas where improvements could contribute to the reusability of the software, particularly through the use of the topic area levels from which the overall RRLs were developed. A brief introduction to this method of assessing the reusability maturity of software components follows.

As described in Marshall & Downs (2008), the RRLs were developed by members of the Earth science data systems software development community through an iterative process. Initially, the needs of both software contributors and software adopters were considered in order to identify topic areas that could be important for measuring the reuse maturity of software components. Nine such areas were identified; alphabetically, they are documentation, extensibility, intellectual property issues, modularity, packaging, portability, standards compliance, support, and verification/testing. Nine teams, each consisting of at least two people who have been involved with software development, worked together to write a set of levels for these nine topic areas; descriptions of the topic level areas can be found in Marshall et al. (2008). Next, members of these teams looked across all topic areas at each level to draft summaries of the overall RRLs, combining key information from all topics at the same level. Based on feedback received from other members of the community, the overall RRLs and topic area levels were revised and improved. This process of revising the levels and obtaining additional feedback to use in further revisions is repeated as the community continues to improve the RRLs. A summary of the RRLs is presented in Table 2.

Additional details on the RRLs and the topic area levels from which they were developed can be found in materials produced by the NASA Earth Science Data Systems Software Reuse Working Group, many of which are publicly available on the group's web site (2009).

RRL	Summary
1	Limited reusability; the software is not recommended for reuse.
2	Initial reusability; software reuse is not practical.
3	Basic reusability; the software might be reusable by skilled users at substantial effort, cost, and risk.
4	Reuse is possible; the software might be reused by most users with some effort, cost, and risk.
5	Reuse is practical; the software could be reused by most users with reasonable cost and risk.
6	Software is reusable; the software can be reused by most users although there may be some cost and risk.
7	Software is highly reusable; the software can be reused by most users with minimum cost and risk.
8	Demonstrated local reusability; the software has been reused by multiple users.
9	Demonstrated extensive reusability; the software is being reused by many classes of users over a wide range of systems.

Table 2. Reuse Readiness Levels

The Earth science data systems development community also has identified a number of potential uses for the RRLs. As metadata for reusable software components stored in catalogs and repositories, RRLs provide a guide to software adopters. They can help adopters quickly asses the maturity of candidate components for their reuse efforts, narrowing down the number of possible solutions they must consider in detail. The RRLs, and the RRL topic area levels in particular, can serve as an indicator of areas to focus on when creating reusable components, as a guide to contributors. Each topic area was selected because it was identified as an important factor that contributes to the reusability of software. By assessing their software components in each of the topic areas, contributors can identify the strengths and weaknesses of their components and work to improve the reusability of the components by using the levels as a guide. The RRLs are being evaluated and work is in progress to develop specific use cases for the RRLs, for both software contributors and software adopters.

It also has been recognized that RRLs could eventually become part of requests for proposals or contracts, which require a reuse approach or explanation of how components are being made reusable. Projects could undertake reusability improvement efforts, indicating that software that begins at one RRL will be developed to and released at a higher RRL. By maturing the reusability of the software, the chances of it being reused would be higher, enabling more projects to benefit from previous efforts. Projects that involve new software development could propose to make their resulting work available for reuse and indicate the planned RRL that would be targeted for release of the software. Projects that will be reusing software could indicate the RRLs of the component(s) being considered for reuse and demonstrate how this reuse provides benefits to the proposed work (e.g., by reducing development time and costs).

Tools such as the RRLs can be very helpful in both creating and adopting reusable software components. Developers are advised to take advantage of such tools when possible.

4. Building components for reuse

Creating reusable software and other components also offers benefits for system developers and their employing organizations. In addition to the benefits of reusing software, system developers also need to recognize the potential value that their current system development activities could have for future system development efforts. Besides contributing to the new system that is being developed, the results of current system development projects also can contribute to the development of future systems that are created by the same organization or by other organizations. Considering the potential value of reusable components for future systems, building software components so that they can be reused could be economically beneficial for organizations involved in software development. There is evidence that reuse offers cost reduction, cost avoidance, and increased profit even though there can be additional costs involved in developing software for reuse (Lim, 1998). Furthermore, building software components to be reusable should be considered good business practice (Stephens, 2006), enabling organizations to reduce redundancy, avoid increasing maintenance costs, and meet evolving requirements.

4.1 Motivations for building reusable components

Other factors also motivate individual developers to create reusable software components, either for reuse within the same organization that originally created the software or for reuse by others. Open source software developers can be motivated both intrinsically, for altruistic reasons, and extrinsically, for potential gains (Hars & Ou, 2002; Wu et al., 2007). Such motivations can be complementary for individual software developers (Roberts et al., 2006) and also could evolve (Shah, 2006). However, extrinsic factors, such as improving reputation and self-development, could be more motivating than intrinsic factors, such as altruism (Oreg & Nov, 2008). Factors motivating an individual developer's involvement in open source Internet communications software development projects include personal software needs, expectations of skill development, reputation enhancement, and enjoyment (Xu et al., 2009). Community factors such as leadership effectiveness and interpersonal relationships also were found to be motivators for involvement in these software development projects (Xu et al., 2009).

While systems development technology and available resources and tools can be expected to change, the purposes for developing future systems could have many aspects in common with the systems that are being developed today. In addition, as techniques and capabilities for developing reusable components improve, future systems could gain even more from current software development efforts. Software development organizations also could contribute to the motivations of their employees to engage in software reuse activities by providing the resources to create an infrastructure that facilitates software reuse (Sherif et al., 2006). Commitment to reuse by the top management of the organization also could be a contributing factor (Morisio et al., 2002).

4.2 Planning and developing reusable components

Planning is necessary when developing systems components for potential reuse. In some cases, initial efforts to ensure the reusability of system components can offer future benefits without adding to development costs. If additional development costs will be required to produce reusable components, these must be weighed against the potential benefits and cost

savings of reusing the components in other projects in the future. If a small additional expenditure on the current project can save a larger amount of money in future projects, it would be justified. However, if the organization developing the reusable components is not expected to be the one to reap the benefits of reusing the components in the future, it may be more difficult to justify the additional cost of preparing components for future reuse. Engaging in a planned, systematic approach to software reuse, including the identification of its benefits, can reduce the potential risks of software reuse (Frakes & Isoda, 1994).

Certainly, many of the activities that foster reusability, such as effectively documenting the work, should be conducted routinely during system development even if reuse is not planned to ensure that enhancements can be completed efficiently. While seemingly trivial and sometimes overlooked by software developers, adding comments to source code can assist in its subsequent analysis and reuse. Furthermore, the inclusion of comments was found to be a success factor for module design that contributed to reuse in large-scale systems at NASA (Selby, 2005).

Developing to standards also can facilitate reuse by others. By developing to established standards for software (see, for example, Baldo et al., 1997 and IEEE, 2004) and data (Baxter et al., 2006), reuse of both software and data can be fostered.

Furthermore, establishing standards for system components enables others to develop to such standards and potentially foster reuse of components that meet the standards. For example, NASA has proposed an open architecture standard for software-defined radio systems that could facilitate the reuse of design expertise as well as software code, offering potential gains in efficiency for future development and modification efforts (Reinhart et al., 2008).

Independent of whether components have been developed to meet established standards, it is necessary to provide sufficient information about resources available for reuse so that they can be identified and assessed for adoption by potential adopters. In addition, enabling reuse also requires developers to grant appropriate intellectual property rights to eliminate legal barriers to reuse, as described in the following section on contributing components for reuse.

5. Contributing components for reuse

Contributing reusable system components to open source software collections and catalogs can enable others to reuse such components if they are licensed as shareable resources that foster reuse by others. Enabling others to reuse system components can contribute to the pool of resources available for all members of the open source community and result in further improvements to these resources by others. If there are barriers to adoption, it will be more difficult for potential adopters to reuse software and other available components. Such barriers can appear at the individual level or the organizational level, and the interaction of barriers at the organizational level can create barriers at the individual level (Sherif & Vinze, 2003). It is important for software developers to consider the potential barriers that exist at all levels and to take the steps that are necessary to address such barriers in order for their software components and related artifacts to be reused more easily by potential adopters.

5.1 Authorizing reuse

Potential adopters of reusable components need to be authorized to reuse components that they identify for potential adoption. Without the legal rights to adopt available components,

it would be imprudent for potential adopters to incur potential risks by adopting components that they have not been authorized to use. Furthermore, considering the effort involved in adopting reusable components and integrating such components into systems, such investments would not be recommended without first ensuring that the rights for reuse have been obtained. Given these considerations, it is necessary for contributors of components to grant the intellectual property rights necessary for others to adopt components that have been developed for reuse. In addition, it also is necessary to adequately communicate information about the rights for reuse so that potential adopters can easily determine whether they have the right to adopt a particular component and determine any conditions for reuse that might be associated with the adoption.

Several licensing options are available for system and software developers to authorize others to utilize components that have been developed for reuse. One such option is open source, for which there are a number of available licenses; see, for example, Open Source Initiative (2009) for a list. However, prior to considering the available choices for granting rights for reuse, developers should identify any policies or rules within their own organization that might govern their choices. Some organizations have policies that prevent granting rights to others. In such cases, a waiver might have to be obtained from the appropriate authority within the organization so that authorization for reuse can be granted. Organizations also might be concerned about liability and could require review to ensure that granting rights for reuse does not expose the organization to such liability. In some cases, organizations that routinely create reusable components might have stock licensing statements available for developers to offer components for reuse. Such licensing and liability statements should be reviewed by an attorney.

5.2 Propagating reuse

Adopters of reusable software should consider sharing any enhancements that they have made to reusable software so that others also may benefit from their improvements. The changes that one organization finds useful might also be useful for other organizations. In addition, other software developers could evaluate the contributed changes and improve them even further. Likewise, such improvements might be considered valuable by the original adopters who had shared their enhancement. In such cases, the original adopters might decide to adopt the new version that includes the improvements to their original enhancements. These kinds of scenarios could be compounded as the community that adopts and contributes enhancements, for a particular reusable resource, continues to grow. Such cycles of individual contributions can improve libraries of software components, offering benefits to the software development community that participates in the software reuse process.

6. Summary

The reuse of software components and other artifacts offers potential benefits for software developers of aerospace systems and other systems. However, a planned and systematic approach to software reuse is recommended for software developers and software development organizations to realize the benefits of software reuse while minimizing potential costs and risks of reuse. The tools and techniques for software reuse that have been described offer options for developing the capabilities to engage in software reuse as

contributors and adopters of reusable software and related artifacts. In addition, suggestions for improving software reuse practices have been offered for software contributors and for software adopters.

7. Acknowledgements

The authors appreciate the contributions of the current and previous members of the National Aeronautics and Space Administration (NASA) Earth Science Data Systems Software Reuse Working Group to some of the work presented here, and very much appreciate the support received from the NASA for the work reported in this paper, including the support for Robert Downs under Contract NAS5-03117.

8. References

Ajila, S.A. & Wu, D. (2007) Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software,* 80, 9, (September 2007) 1517–1529, ISSN 0164-1212

Baldo, J.; Moore, J. & Rine, D. (1997). Software reuse standards. *StandardView,* 5, 2, (June 1997) 50–57, ISSN 1067-9936

Basili, V.R.; Rombach, H.D.; Bailey, J.; Delis, A. & Farhat, F. (1989). Ada Reuse Metrics, In: *Guidelines Document for Ada Reuse and Metrics*, P. A. Lesslie, R. O. Chester, and M. F. Theofaanos (Eds.), 11–29, Martin Marietta Energy Systems, Inc., Oak Ridge, Tennessee, under contract to U.S. Army, AIRMICS

Baxter, S.M.; Day, S.W.; Fetrow, J.S. & Reisinger, S.J. (2006). Scientific Software Development Is Not an Oxymoron. *PLoS Computational Biology,* 2, 9, e87, ISSN 1553-734X (Print) 1553-7358 (Online)

Comprehensive Perl Archive Network (2009). http://www.cpan.org/

Comprehensive TeX Archive Network (2009). TeX Users Group, http://www.ctan.org/

Earth Science Data Systems Software Reuse Working Group (2009). http://www.esdswg.com/ softwarereuse/

Finnigan, J.V. & Blanchette, J. (2008). A Forward-Looking Software Reuse Strategy. *Proceedings of the 2008 IEEE Aerospace Conference,* pp. 1–9, ISBN 978-1-4244-1487-1, Big Sky, Montana, March 2008, IEEE Press, New York

Frakes, W.B. & Isoda, S. (1994). Success factors of systematic reuse. *IEEE Software*, 11, 5, (September 1994) 14–19, ISSN 0740-7459

Frakes, W.B. & Kang, K. (2005). Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*, 31, 7, (July 2005) 529–536, ISSN 0098-5589

Hars, A. & Ou, S. (2002). Working for Free? Motivations for Participating in Open-Source Projects, *International Journal of Electronic Commerce,* 6, 3, (Spring 2002), 25–39, ISSN 1086-4415

HDF-EOF Tools and Information Center (2009). The HDF Group, http://hdfeos.org/ software.php

Innovative Partnerships Program (IPP) Office (2009). Featured Technologies, http://ipp.gsfc.nasa.gov/ technologies.html

Institute of Electrical and Electronics Engineers, Inc. (2004). *IEEE Standard for Information Technology – Software Life Cycle Processes – Reuse Processes*, IEEE Press, ISBN 0-7381-1735-8, New York

Immonen, A. & Niemela, E. (2008). Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7, 1 (February 2008) 49–65, ISSN 1619-1366 (Print) 1619-1374 (Online)

Leyton, M. (2008). Defining Reuse, presentation available online at: http://www.esdswg.com/softwarereuse/Resources/library/7th-esds-wg-meeting/ESDSWG7DefinitionOfReuse.pdf

Lim, W.C. (1998). *Managing Software Reuse: A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components,* Prentice Hall PTR, ISBN 0-13-552373-7, Upper Saddle River, NJ

Marshall, J.J. & Downs, R.R (2008). Reuse Readiness Levels as a Measure of Software Reusability, *Proceedings of 2008 Geoscience and Remote Sensing Symposium, Volume 3,* pp. III-1414–III-1417, ISBN 978-1-4244-2807-6, Boston, Massachusetts, July 2008, IEEE Press, New York

Marshall, J.J.; Downs, R.R.; Samadi, S.; Gerard, N.S.; Wolfe, R.E. (2008). Software Reuse to Support Earth Science. *Journal of Frontiers of Computer Science and Technology,* 2, 3 (June 2008), 296–310, ISSN 1673-9418

Marshall, J.J.; Olding, S.W.; Wolfe, R.E. & Delnore, V.E. (2006). Software Reuse Within the Earth Science Community, *Proceedings of 2006 IEEE International Conference on Geoscience and Remote Sensing Symposium,* pp. 2880–2883, ISBN 0-7803-9510-7, Denver, Colorado, July–August 2006, IEEE Press, New York

Mascena, J.C.C.P.; de Almeida, E.S. & de Lemos Meira, S.R. (2005). A Comparative Study on Software Reuse Metrics and Economic Models from a Traceability Perspective, *Proceedings of IEEE International Conference on Information Reuse and Integration*, pp. 72–77, ISBN 0-7803-9093-8, Las Vegas, August 2005, IEEE Press, New York

Mohagheghi, P. & Conradi, R. (2007). Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12, 471–516, ISSN 1382-3256 (Print) 1573-7616 (Online)

Morisio, M.; Ezran, M. & Tully, C. (2002). Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering*, 28, 4, (April 2002) 340–357, ISSN 0098-5589

Murray, A.; Schoppers, M. & Scandore, S. (2009). A Reusable Architectural Pattern for Auto-Generated Payload Management Flight Software, *Proceedings of the 2009 IEEE Aerospace Conference,* pp. 1–11, ISBN 978-1-4244-2621-8, Big Sky, Montana, March 2009, IEEE Press, New York

National Aeronautics and Space Administration (2007). *Systems Engineering Handbook,* SP-2007-6105 Rev1, Washington, D.C.

Nazareth, D.L. & Rothenberger, M.A. (2004). Assessing the cost-effectiveness of software reuse: a model for planned reuse. *Journal of Systems and Software,* 73, 2, (October 2004) 245–255, ISSN 0164-1212

Neighbors, J.M. (1992). The Evolution from Software Components to Domain Analysis. *International Journal of Software Engineering and Knowledge Engineering*, 2, 3, (September 1992) 325–354, ISSN 0218-1940

Netlib (2009). Netlib Repository at UTK and ORNL. http://www.netlib.org/

Open Source Initiative (2009). Open Source Licenses, http://www.opensource.org/licenses

Oreg, S. & Nov, O. (2007). Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values, *Computers in Human Behavior,* 24, 5, (September 2008) 2055–2073, ISSN 0747-5632

Orrego, A.S. & Mundy, G.E. (2007). A Study of Software Reuse in NASA Legacy Systems. *Innovations in Systems and Software Engineering,* 3, 3, (September 2007) 167–180, ISSN 1614-5046 (Print) 1614-5054 (Online)

Prieto-Diaz, R. (1990). Domain Analysis: An Introduction. *ACM Software Engineering Notes,* 15, 2, (April 1990) 6–16, ISSN 0163-5948

Reinhart, R.C.; Johnson, S.K.; Kacpura, T.J.; Hall, C.S.; Smith, C.R.; & Liebetreu, J. (2008). Open Architecture Standard for NASA's Software-Defined Space Telecommunications Radio Systems, NASA/TP—2008-214941, pp. 1–8, May 2008, National Aeronautics and Space Administration, Glenn Research Center, Cleveland, Ohio. http://gltrs.grc.nasa.gov/reports/2008/TP-2008-214941.pdf

Roberts, J.A.; Hann, I. & Slaughter, S.A. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects, *Management Science,* 52, 7, (July 2006), 984–999, ISSN 0025-1909 (Print) 1526-5501 (Online)

Samadi, S.; Gerard, R.; Hunter, M.; Marshall, J.J.; Schweiss, R.J.; Wolfe, R.E. & Masuoka, E.J. (2007). Reusing Software to Build Data Processing Systems, *Proceedings of 2007 IEEE Aerospace Conference*, pp. 1–12, ISBN 1-4244-0525-4, Big Sky, Montana, March 2007, IEEE Press, New York

Selby, R.W. (2005). Enabling reuse-based software development of large-scale systems, *IEEE Transactions on Software Engineering,* 31, 6, (June 2005) 495–510, ISSN 0098-5589

Shah, S. K. (2006). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development, *Management Science,* 52, 7, (July 2006), 1000–1014, ISSN 0025-1909 (Print) 1526-5501 (Online)

Sherif, K.; Appan, R. & Lin, Z. (2006). Resources and Incentives for the Adoption of Systematic Software Reuse. *International Journal of Information Management*, 26, 1, (February 2006) 70–80, ISSN 0268-4012

Sherif, K. & Vinze, A. (2003). Barriers to adoption of software reuse: A qualitative study, *Information & Management,* 41, 2, (December 2003) 159–175, ISSN 0378-7206

Simos, M.A. (1988). The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability, In *Software Reuse: Emerging Technology,* Will Tracz (Ed.), 354–363, IEEE Computer Society Press, ISBN 0-8186-0846-3, Los Alamitos, California

SourceForge, Inc. (2009). SourceForge.net: Find and Develop Open-Source Software. http://sourceforge.net/

Space Telescope Science Institute (2009). Software and Hardware Products from STScI. Baltimore, Maryland, http://www.stsci.edu/resources/software_hardware

Stephens, K. (2006) Web Services and Asset Reuse: Implications for Developers. International Business Machines Corporation, Armonk, New York, http://www.ibm.com/developerworks/webservices/library/ws-assetreuse/

Suri, P. K. & Garg, N. (2008). Simulator for evaluating Reliability of Reusable Components in a Domain Interconnection Network. *International Journal of Computer Science and Network Security*, 8, 3, (March 2008) 251–259, ISSN 1738-7906

Wu, C.; Gerlach, J.H. & Young, C.E. (2007). An empirical analysis of open source software developers' motivations and continuance intentions, *Information & Management,* 44, 3, (April 2007) 253–262, ISSN 0378-7206

Xu, B.; Jones, D.R. & Shao, B. (2009). Volunteers' involvement in online community based software development, *Information & Management*, 46, 3, (April 2009) 151–158, ISSN 0378-7206