NASA/SP–2013–604

NASA Software Engineering Benchmarking Study

Heather L. Rarick
Johnson Space Flight Center, Houston, TX

Sara H. Godfrey
Goddard Space Flight Center, Greenbelt, MD

John C. Kelly
NASA Headquarters, Washington, DC

Robert T. Crumbley
Marshal Space Flight Center, Huntsville, AL

Joel M. Wilf
Jet Propulsion Laboratory, Pasadena, CA

May 2013

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing help desk and personal search support, and enabling data exchange services. For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA STI Help Desk at 443-757-5803
- Phone the NASA STI Help Desk at 443-757-5802
- Write to:

  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/SP–2013–604

NASA Software Engineering Benchmarking Study

Heather L. Rarick
Johnson Space Flight Center, Houston, TX

Sara H. Godfrey
Goddard Space Flight Center, Greenbelt, MD

John C. Kelly
NASA Headquarters, Washington, DC

Robert T. Crumbley
Marshal Space Flight Center, Huntsville, AL

Joel M. Wilf
Jet Propulsion Laboratory, Pasadena, CA

National Aeronautics and
Space Administration

NASA Headquarters
Washington, DC 20546

May 2013

Level of Review: This material has been technically reviewed by technical management

Available from:
NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161 Price Code: A17

Table of Contents

List of Tables

List of Figures

Executive Summary

To identify best practices for the improvement of software engineering on projects, NASA's Offices of Chief Engineer (OCE) and Safety and Mission Assurance (OSMA) formed a team led by Heather Rarick and Sally Godfrey to conduct this benchmarking study. The primary goals of the study are to identify best practices that:

- Improve the management and technical development of software intensive systems.
- Have a track record of successful deployment by aerospace industries, universities [including research and development (R&D) laboratories], and defense services, as well as NASA's own component Centers.
- Identify candidate solutions for NASA's software issues.

Beginning in the late fall of 2010, focus topics were chosen and interview questions were developed, based on the NASA top software challenges. Between February 2011 and November 2011, the Benchmark Team interviewed a total of 18 organizations, consisting of five NASA Centers, five industry organizations, four defense services organizations, and four university or university R&D laboratory organizations. A software assurance representative also participated in each of the interviews to focus on assurance and software safety best practices.

Interviewees provided a wealth of information on each topic area that included: software policy, software acquisition, software assurance, testing, training, maintaining rigor in small projects, metrics, and use of the Capability Maturity Model Integration (CMMI) framework, as well as a number of special topics that came up in the discussions. NASA's software engineering practices compared favorably with the external organizations in most benchmark areas, but in every topic, there were ways in which NASA could improve its practices. Compared to defense services organizations and some of the industry organizations, one of NASA's notable weaknesses involved communication with contractors regarding its policies and requirements for acquired software. One of NASA's strengths was its software assurance practices, which seemed to rate well in comparison to the other organizational groups and also seemed to include a larger scope of activities.

An unexpected benefit of the software benchmarking study was the identification of many opportunities for collaboration in areas including metrics, training, sharing of CMMI experiences and resources such as instructors and CMMI Lead Appraisers, and even sharing of assets such as documented processes. A further unexpected benefit of the study was the feedback on NASA practices that was received from some of the organizations interviewed. From that feedback, other potential areas where NASA could

improve were highlighted, such as accuracy of software cost estimation and budgetary practices.

The following detailed report contains discussion of the practices noted in each of the topic areas, as well as a summary of observations and recommendations from each of the topic areas. The resulting 24 recommendations from the topic areas were then consolidated to eliminate duplication and culled into a set of 14 suggested actionable recommendations. This final set of actionable recommendations, listed below, are items that can be implemented to improve NASA's software engineering practices and to help address many of the items that were listed in the NASA top software engineering issues.

1. Develop and implement standard contract language for software procurements.
2. Advance accurate and trusted software cost estimates for both procured and in-house software and improve the capture of actual cost data to facilitate further improvements.
3. Establish a consistent set of objectives and expectations, specifically types of metrics at the Agency level, so key trends and models can be identified and used to continuously improve software processes and each software development effort.
4. Maintain the CMMI Maturity Level requirement for critical NASA projects and use CMMI to measure organizations developing software for NASA.
5. Consolidate, collect and, if needed, develop common processes principles and other assets across the Agency in order to provide more consistency in software development and acquisition practices and to reduce the overall cost of maintaining or increasing current NASA CMMI maturity levels.
6. Provide additional support for small projects that includes: (a) guidance for appropriate tailoring of requirements for small projects, (b) availability of suitable tools, including support tool set-up and training, and (c) training for small project personnel, assurance personnel and technical authorities on the acceptable options for tailoring requirements and performing assurance on small projects.
7. Develop software training classes for the more experienced software engineers using on-line training, videos, or small separate modules of training that can be accommodated as needed throughout a project.
8. Create guidelines to structure non-classroom training opportunities such as mentoring, peer reviews, lessons learned sessions, and on-the-job training.
9. Develop a set of predictive software defect data and a process for assessing software testing metric data against it.
10. Assess Agency-wide licenses for commonly used software tools.
11. Fill the knowledge gap in common software engineering practices for new hires and co-ops.

12. Work through the Science, Technology, Engineering and Mathematics (STEM) program with universities in strengthening education in the use of common software engineering practices and standards.
13. Follow up this benchmark study with a deeper look into what both internal and external organizations perceive as the scope of software assurance, the value they expect to obtain from it, and the shortcomings they experience in the current practice.
14. Continue interactions with external software engineering environment through collaborations, knowledge sharing, and benchmarking.

1. Introduction

In almost every NASA program and project, software is a critical product. Nearly every piece of hardware in use on a launch vehicle, spacecraft, science experiment, ground system or network requires software to monitor or control its operation. "Today, software touches everything in modern spacecraft development. Why does software fix hardware problems? Because it can. ...Bottom line: the game has changed in developing space systems. Software and avionics have become the system."[1] Success of software is critical to the success of NASA.

NASA leadership has worked diligently to improve software engineering and has decided that an examination of internal and external practices would greatly benefit the work being done within and for NASA. Thus a benchmarking effort has been undertaken to identify, review and employ best practices relevant to the software that is critical to NASA's missions.

1.1. Background

NASA policy directives (NPDs) and NASA Procedural Requirements (NPRs) govern the policies, process and requirements for each NASA program/project, including specific software engineering requirements. In addition to these documents, other mechanisms are used to effectively manage software engineering such as common training and feedback from programs/projects. The NASA Software Executive and the Software Working Group (SWG) are responsible for establishing the software engineering processes and training required to manage and produce software vital to the success of NASA's missions.

The SWG is led and chaired by the NASA Software Executive from the OCE, and functions as an advisory group from its charter to manage software engineering and the advancement of software engineering practices. The SWG is comprised of NASA Center software engineering and software assurance experts who meet regularly to plan and execute their tasks.

1.2.1 The NASA Headquarters Office of the Chief Engineer shall lead, maintain, and fund a NASA Software Engineering Initiative to advance software engineering practices. [SWE-002]

1.2.2 Each Center shall maintain, staff, and implement a plan to continually advance its in-house software engineering capability and monitor the software

[1] Is Software Broken? by Steve Jolley , NASA ASK Magazine, Issue 34, September 2009 http://askmagazine.nasa.gov/issues/34/34i_software_broken.html.

engineering capability of NASA's contractors, as per NASA's Software Engineering Initiative Improvement Plan. [SWE-003][2]

The NASA SWG Chair initiated a benchmarking effort for fiscal year 2011 (FY11) to discover new software engineering techniques and tools; review and learn from internal and external organizations; and, examine the current and near-term software engineering environment. Initially, this benchmarking effort was organized with the expectation of gathering best software engineering practices from among a sampling of organizations, and sharing these practices throughout NASA. Although this remains the cornerstone of the effort, the purpose was further developed to include best practices within NASA. By including internal organizations (NASA Centers), the effort also gained baseline data that can be compared with the external organizations. This effort and analysis of the information collected is expected to advance software engineering practices within NASA.

[2] *NPR 7150.2A, NASA Software Engineering Requirements*, NASA Office of the Chief Engineer, 2009.

2. Purpose

The purpose of this report is to share the results of the interviews and discussions that were arranged with select software engineering and software assurance organizations from NASA Centers and from other organizations that had or have done work with or for NASA. Identified are internal best practices alongside external best practices; new techniques and tools; and possible improvements that could be or may necessarily be achieved within the next few years. This report, summary briefings, and execution of forward actions are expected to advance the software engineering practices of NASA.

3. Scope

Since this benchmarking effort could encompass many aspects of software engineering and many organizations, the Benchmark Team narrowed the focus to strategic target areas and requested interviews with the specific categories of organizations. Software assurance, although often treated independently in NASA and elsewhere, was included and should be assumed whenever "software engineering" is referenced or used.

3.1. Scope of Topic Areas

It was necessary to focus the effort on target areas that were currently of high concern or interest to NASA Centers, and which offered a practical opportunity to obtain useful information. Therefore, the Benchmark Team utilized a list of NASA's top software issues compiled in 2010[3] (Appendix B), as a basis for determining the key target areas. For example, software cost estimation is on the top software issues list, but the topic was excluded since it is unlikely that competitive organizations would divulge their cost strategies. The key target areas and a synopsis of the questions asked during the interview are summarized below. A few specific comments from the top software issues list are also noted and incorporated into the suite of questions used at the interviews (Appendix D).

1. **Software policies:** Identify level of detail and use of industry standards, who/how are policies developed, changed, communicated and complied with. Is there a more effective way to determine, communicate and ensure compliance with policies?
2. **Software acquisitions:** Examine how to maintain organizational requirements in software acquisitions. Are policies and requirements waived, tailored or fully met, successful or not?

   Comments from 2010 Top Issues List: Software does not receive adequate attention in contracts where the procurement is a hardware product containing software. Required software documentation, measurements and reviews are often omitted from contract (due to cost)… Issues identified with incorporating commercial off-the-shelf (COTS) and open source products into mission critical software developments (e.g., flight software, ground software, and the software development environment) and maintaining rigorous processes.

[3] *Top NASA Software Issues of 2010*, Joint software Working Group / Mission software Steering Committee Meeting at NASA Plum Brook, August, 2010.

3. **Testing:** Seek insights on testing and opportunities to improve testing time, reduce errors found in testing, manage schedules and composition of test teams. What factors improve the efficiency and shorten the testing time?
4. **Software assurance:** Obtain understanding of software assurance tools, training, metrics and relationship with software quality, reliability and safety. How is software engineering integrated with software assurance?
5. **Training:** Inquire about training and whether or not in house training programs are more effective/efficient. What are some different types of training techniques that are used and what are the most useful topics that are provided to software engineers?

   Comments from 2010 Top Issues List: There are insufficient software experts available to assist software development teams that are composed of scientists and engineers without formal software engineering training. The education level of non-software discipline engineers, project managers and Center management knowledge regarding software [value, why, how it's designed, related NPR's, software classifications, value of software assurance/Independent Verification and Validation (IV&V)] could be substantially improved.
6. **CMMI – maturity level benefits and benefits of advancement:** Analyze organizations that have CMMI maturity levels of two or greater and solicit benefits and costs. Who and what drives the maturity levels, how are the benefits demonstrated, how are costs minimized and accepted?
7. **Small projects:** Investigate how rigor is maintained. Are policies and requirements tailored, are there alternate tools/techniques used for small software projects? NOTE: Small projects at NASA are defined as using five or fewer software engineers.

   Comment from 2010 Top Issues List: The number of requirements and documents are disproportional to the funding and risk levels of small missions, technology demonstration projects and smaller robotics spacecraft with accepted risk.
8. **Tools:** Find out what tools are prevalent and what success or efficiencies are gained from the use of the tools. What tools are most often used and are there greater dependencies on specific tools for small projects with limited resources?

Two other topics were also discussed via this benchmark. Both topics were considered areas of interest and treated as secondary and optional objectives. Questions regarding metrics were asked of most of the organizations but were more heavily pursued with

organizations that had higher level CMMI maturity levels. (For the purpose of this benchmark and report, NASA considered CMMI maturity levels 3 (ML3) and higher as "higher maturity levels.") Questions regarding programmable logic devices (PLD) were asked but discussions were not detailed.

- **Metrics:** Ascertain the success of metrics in improving software and impacts of life cycle. What metrics are most useful and how are they used?
- **Programmable logic devices:** Discuss how software organizations are handling and their concerns about PLDs. Are software organizations participating in PLDs from a process and requirements perspective, should they be if they are not?

  Comment from 2010 Top Issues List: There is an increasing tendency to put mission critical functionality in field programmable gate arrays (FPGAs) because of the perception that there are cost and schedule benefits to doing this. Perception that complex electronics are not held to the same level of rigor in requirements, development, testing, and verification as software. Need combination of hardware and software approaches to developing complex electronics.

Each organization was asked a set of questions on the key target areas. These questions were limited to keep the length of the interview to less than three hours. The questions were also targeted to organizations that develop and/or acquire software comparable to NASA mission critical software. The benchmarking team decided on this limitation because software for projects of a much less critical nature may not be applicable to software issues at NASA (based on cost versus benefit). Several smaller organizations and organizations with unique and possibly innovative competitive attributes, but which still met the critical software development criteria, were selected to minimally compensate for those excluded organizations.

3.2. Scope of Chosen Organizations

The scope of the organizations to be interviewed was focused on those that had similar software criticality. The Benchmark Team chose defense services organizations, aerospace companies, universities and university laboratories. These organizations had or have done work with or for NASA. Several NASA Centers were also interviewed as part of the benchmark.

A minimum of three organizations from each category was originally planned but ultimately expanded to include 4 Defense Services organizations, 5 Aerospace Industry organizations, 4 Universities (2 Universities and 2 University Research and

Development Laboratories) and 5 of the 10 NASA Centers. Although this study interviewed more organizations than originally anticipated, the collection is still a small sample of organizations that could have been interviewed as well as a very limited sample of organizations within each category.

The Benchmark Team extends its thanks and appreciation to all organizations and participants. The openness and willingness to communicate exhibited by the interviewed organizations was tremendous and is of great value to NASA and to the software engineering community. In exchange for the honest and accurate information provided by the interviewed organizations, NASA has agreed to not disclose the names of the organizations or of the personnel that participated.

4. Methods

4.1. Preparation

The benchmark effort was established with eight key target areas based on many of NASA's top software issues. These target areas were converted into interview questions. An extensive list of questions was generated and then reduced to fit into a three hour interview. Appendix D contains the list of questions shared with the participating organizations and a longer list of questions used by the Benchmark Team to assist in the discussions (not provided to the organizations). The questions, along with a short briefing (Appendix C), were sent to each participating organization. With this request a recommendation was made on which persons in the organization, based on roles and responsibilities would be appropriate to attend.

Contacts with personnel in the 18 organizations were used to initiate the process. Once the organizations responded with interest, details were provided and an interview was scheduled.

4.2. Interviews

Interviews consisted of three to five NASA personnel, typically one or both of the Benchmark Team leads, the SWG Chair and/or Deputy Chair, a software assurance person and an SWG member local to the interview location. Appendix G lists the NASA personnel who participated on the interview teams.

The NASA personnel were responsible for asking questions and taking notes. Although the questions were formalized and provided, the face-to-face meeting allowed for a more natural and comfortable dialogue among participants, leading to more open and robust discussions. For some interviews, travel by all of the NASA personnel was not possible, so the interviews were conducted using both on-site personnel and teleconference. The intention was to maintain some consistency within the benchmarking effort, using the Benchmark Leads and the SWG Chair/Deputy, but also to include some diversity, using local SWG members. Including a member from the SWG also enhanced the SWG's first-hand knowledge of internal and external software engineering organizations.

A typical interview began with a brief summary presentation of the purpose and scope of the benchmark effort (Appendix C). The summary also included an explanation of NASA's software engineering and assurance organizations, NASA and Center policies and requirements, software classifications, training, CMMI at NASA and the top software issues. The organization often provided a reciprocal briefing to explain their organization and their policies and requirements. Following these introductory and

background discussions, the NASA personnel asked questions and facilitated discussions on the key target areas. Some organizations had greater interest and involvement in some topics, so the interview was adjusted to delve into those areas and spend less time on other target areas. It is important to note that NASA chose to focus on understanding each organization and used the questions to gain that understanding and identify different ideas rather than pursuing answers to each specific question.

4.3. Data Analysis and Report Development

After the interview, notes were collected and used to generate the key target area summaries within Section 5 (General Observations) in this report. Each summary topic begins with a list of the topic's interview questions, and then focuses on organizing the data into subsections:

- General discussion with specific information regarding:
  - Aerospace industries
  - Universities and university labs
  - Defense Services Organizations
  - NASA Centers
- Major observations
- Recommendations

In Section 5, similarities and differences between the organizational groups can be recognized and common and unique best practices can be identified. Both the similarities/differences as well as the commonality/uniqueness may be helpful in determining the success or applicability to implementation within NASA. For each key topic area, a few major observations were assembled to capture some of the more relevant data in the Discussion section. Recommendations were specified to apply the relevant data into possible forward plans for implementation within NASA.

The remaining sections, Benefits (Section 6) and Comparison and Trends (Section 7) were written to explore where NASA is exceeding the current software engineering environment and where NASA should focus its efforts to improve on its software engineering. The data and information presented is based on the review of the Benchmark Team leads and the SWG Chair/Deputy which was discussed and analyzed from recollection and interview notes. The NASA interview participants shared in the review and completion of the key target area summaries and the review of the entire report.

The Benefits Section introduces unanticipated outcomes and an assessment on the benefits of this study. The Comparisons and Trends Section looks at how NASA fared

against the different organizational groups. This perspective provides corroboration in determining the areas where NASA should focus to achieve improvements and which improvements are most needed.

The recommendations listed in Section 8 include the full list of recommendations from each topic area (Section 8.1) and also includes a suggested set of recommendations for NASA review and potentially NASA concurrence for inclusion in near-term plans (Section 8.2). These recommendations are focused on topics that were included in this study because they were considered areas that the NASA software engineering community was interested in improving (Top Software Issues, Appendix B). The recommendations are constructed based on best practices gathered through the benchmarking effort and were analyzed to be areas that would bring NASA's practices closer to the common practices of the external software engineering environment.

At a minimum, this report is intended to be shared within NASA and with the organizations that participated. For NASA, it will serve as a guide on near-term decisions and focal points for further development. It should also serve as an opportunity to continue assessing software engineering on a routine (but smaller) basis since this effort has produced positive results.

5. General Observations

As mentioned in Section 4.3 Data Analysis and Report Development, this section contains much of the detailed information gathered on each of the focus topics during the interviews. Each topic section lists the topic area interview questions and then contains a section of general discussion where the focus is on the broad perspective of the topic across all of the organizations interviewed. Many of the observations that were common for the majority of the organizations are discussed in this initial discussion section.

The initial discussion section is followed by individual discussion sections covering the observations from the interviews, grouped by the types of organizations, i.e., Industry organizations, University and University Research and Development Laboratories, Defense Service organizations, and NASA Centers, so similarities and differences can be seen within the various groupings of organizations.

The next section in each topic area pulls out the “key” observations. Generally these key observations were differences or commonalities with NASA but also include notable comments, some of which are best practices or specific areas where an improvement opportunity for NASA was noted.

And finally, each topic section contains recommendations for NASA improvements in the topic area. The topics in this section include:

- Policy
- Acquisition
- Testing
- Software assurance
- Training
- Metrics
- CMMI
- Small projects
- Tools
- Programmable logic devices (PLD)

5.1. Policy

5.1.1. Questions

- Please identify and summarize software policies, directives or requirements you have that are implemented organization-wide (governing documents)?

- Identify the source of software policies, directives or requirements which are applied to your organization's software activities? Who (organizationally) is responsible for these high-level documents and how/when are they updated?
- Describe how your organization ensures compliance with these governing documents?
- Please explain how these documents are communicated to the users?
- How are policies and requirements included in contracted work (e.g., acquisitions)?
- How are project reviews and milestones coordinated with software reviews and milestones?

5.1.2. Discussion

The first topic of the Benchmark, Software Policy, provides an opportunity to examine approaches to policies, requirements, standards, and guidance which have been successful for outside organizations, as well as those local to NASA Centers. Some of the unique elements of software policy were identified by organization in the table below.

Table 1: Common Software Policy Strategies

	Organizations																
	N1	N2	N3	N4	N5	I1	I2	I3	I4	I5	G1	G2	G3	G4	U1	U2	U3	U4
Policy Topic																		
Formal up front tailoring process	X	X	X	X	X	X	X	X	X		X		X	X		X		X
Use of software classification	X	X	X	X	X	X			X							X		
Corporate/enterprise-wide Processes						X		X	X									
Software engineering principles	X		X	X														
Months to flow down software NASA procedural requirement (NPR) to Center procedural requirement	19	19	35	19	36													
CMMI level - at interview	2	2	2	3	3	3	5	3	3	-	#	2	5	3	-	-	-	3
Previous CMMI level 5 organization						X			X			X						

Nx	NASA Centers	Ux	University, Univ. R&D Labs
Ix	Industry Organizations	Gx	Defense Services
	Not Relevant/Question was not pursued with this type of organization		

Note: X's in matrix indicate practices mentioned in interviews. Lack of X's in boxes DO NOT indicate that the practice is not performed. The chart should be viewed as an indication of the practices mentioned. The light gray X under I1 indicates the organization had used a classification scheme, but no longer uses it.

# indicates organization had a previous CMMI Maturity Level 2 rating, which was expired at time of interview.

5.1.2.1. Aerospace Industries

Representatives from five industry organizations were interviewed for this benchmark. Some interesting trends resulting from these interviews include:

- **Policy:** Four of the five industry organizations interviewed had corporate-wide software policies which are flowed down to projects.
- **Assets:** Of the four with corporate-wide software policies, all had a variety of lower level software direction that was also held corporate-wide. Examples of other corporate-wide software assets that were mentioned include: a standard software process manual; checklists; templates; and a tailoring guideline.
- **Corporate/enterprise-wide processes:** Three of the aerospace industries interviewed had transitioned to a consolidated set of software processes corporate-wide. The transition in two cases included allowances for local differences provided they yielded specific benefits. A third aerospace organization had some of the elements of an enterprise-wide software process approach in place. Some of the benefits included: mobility of software engineers across projects and divisions; strength in software training across the corporation; significant economic advantages in the development and maintenance of software process assets, associated tools, and engineering support kits; common software measurements; and efficiencies gained in CMMI appraisals.
    - **One of the experienced interviewees provided the following caveat:** "The challenge in this approach is to have buy-in to the common practices, as everyone thinks they are unique."
- **Compliance checks:** Industry in-house software quality assurance (SQA) was mentioned as playing a key role in the way each organization ensures compliance with institutional software requirements (involved SQA suggests compliance checks are done). Other supporting checks included: audits; monthly reviews with the software process owner (technical authority); use of checklists that have the software requirements embedded in them; use of compliance matrices; peer reviews (e.g., software inspections, walkthroughs); and keeping of tangible software metrics.
- **Contracting:**
    - Most of the industry organizations did not use subcontractors for software development. Of the two that did, software requirements were flowed down (including CMMI). One company that mentioned outsourcing software development said that they assigned a company flight software technical lead oversight responsibility for the contracted software.

  - There was a general agreement that software policies and requirements for a project were communicated through contracts for all acquirer/supplier relationships. Internal software policies and requirements were communicated through company training programs, mentoring, and general culture.
  - NASA's flow down of software requirements in contracts seems to be variable in this small sample. Organizations reported both excessive software requirements flowed down, as well as a case where no software requirements were put on the contract.
- **Software classification:** Two of the aerospace industry organizations developed internal software classification schemas for engineering systems. These industry schemas have either four or five categories. For comparison purposes, NASA has five software categories for engineering systems and the Federal Aviation Administration (FAA) has four. While these classification schemas vary in their details, there are similarities with mission critical software being at the top of the classification structure.
  - Another industry interviewee mentioned that software classification schemas weren't that useful in their environment as all of the code was considered safety critical.
  - One of aerospace industry's mentioned that their software classification schema was dropped when the company transitioned to a corporate-wide software process asset strategy.
- **Up front tailoring:** Four of the five aerospace organizations described formal upfront tailoring approaches with designated approval authorities. In each of these cases, the tailoring was formally documented and the approval authorities were independent of the project itself.

5.1.2.2. Universities and University Research and Development Labs

Representatives from four Academic/University organizations were interviewed for this benchmark. Two of these locations represented a typical university environment; while the other two were specialized off-site R&D laboratories run by universities. Some interesting trends resulting from these interviews include:

- **Policies and processes in universities:** The two typical university environment organizations had no internal software policies/requirements, nor software process infrastructure [e.g., process asset libraries (PALs), procedures, templates]. Additionally, voluntary consensus standards did not appear to be in common use for software development. The only software development direction

mentioned by these organizations were requirements specified by external customers.

- **Policies and processes in university R&D labs:**
  - The off-site R&D laboratories have institutionalized software processes and procedures based on the CMMI model which are used across multiple projects.
  - The off-site R&D laboratories have documented quality plans and processes to check compliance with both institutional as well as product-specific software requirements.
  - The off-site R&D laboratories have organized training for developers in standard procedures and development methods. This training also serves the purpose of communicating the organization's software policies and procedural requirements.
- **Software classification:** One of the R&D laboratories established a software classification schema which was based on four factors. Like industry and defense services classification schemas mentioned in the preceding and following section, criticality of the software played an important role.
- **Up front tailoring:** One of the R&D laboratories described a formal tailoring process which was based on size, complexity, and guidance from the sponsor.

5.1.2.3. Defense Services

Representatives from four defense services organizations were interviewed for this benchmark. Some interesting trends resulting from these interviews include:

- **Policy:** The source of software policy and requirements came directly from military regulations, standards, CMMI, and *AS9100 Quality Management Systems - Requirements for Aviation, Space and Defense Organizations*. These were augmented by a variety of local software policies and requirements mechanisms:
  - Capstone documents which contained high level 'business area' objectives at the top, down to templates in process asset libraries at the bottom.
  - Plans, local policy (roles and responsibilities), work environment and risk standards, non-conformance product guide (based on *AS9100*), etc.
  - Local organizational software policies and application area workbooks.
  - Process structure: software policy, then processes, then work aids (e.g., templates).

    - Defense services organizations have significant software policy and requirements infrastructures which exceed NASA's in breadth and cultural acceptance (see also Section 5.2, Acquisition of this report). This includes the use of standard contracting language to ensure the software policies/requirements are included in acquisitions which contain software.
- **Software classification:** Within the defense services organizations, interviewees mentioned that all aspects of their systems were critical in nature, and they didn't see the usefulness of a classification schema for the purpose of tailoring requirements and processes out of the development life cycle (i.e., all software was given equal attention and rigor).
- **Up front tailoring:** Three of the four defense services organizations described using formal upfront tailoring approaches (typically involving an approval authority, tailoring guidance, and a record of the tailoring). The other organization didn't mention a formal tailoring process, which could have been a reflection of the critical nature of their application and risks associated with tailoring out procedures.

5.1.2.4. NASA

A summary of NASA's policy and procedural requirements history for software engineering is included in Appendix E for reference. Representatives from five NASA Centers were interviewed for this benchmark. Some interesting trends that resulted from these interviews include:

- **Flow down of NASA's software NPR to Center procedural requirements (xPRs):** All five of the Centers had or were working on establishing local Center Procedural Requirements (xPR) for software based on the Agency-wide requirements. Three of the five Centers had the local xPRs approved and operational, while the other two were in the development or approval process.
- **Principles:** Three of the Centers established engineering principles or rules that include numerous software entries. These principles/rules are basically a way of capturing and promulgating practical lessons learned on projects. The principles/rules are taken very seriously and checked at the project's major milestone review. Two additional Centers are in the process of developing engineering principles/rules for software engineering.
- **Compliance checks:**
    - Compliance checks against applicable software requirements were routine across all of the Centers benchmarked. Software assurance personnel were typically the principle ones performing this verification, but instances

where the software process improvement organization, branch manager, or technical authority played a major role were also found.
    - Software compliance matrices were commonly used to document applicable requirements on projects. Software requirements tailoring was captured in the compliance matrix.
- **Software classification:** NASA has a five-level schema for classifying software based on 1) usage of the software within a NASA system, 2) criticality of the system to NASA's major programs and projects, 3) extent to which humans depend upon the system, 4) developmental and operational complexity, and 5) extent of the Agency's investment. The number of applicable procedural requirements (including a CMMI rating) for software development is determined by the NASA-wide classification schema. There is also an independent second classification on whether software is safety critical (which invokes additional requirements contained in the NASA Software Safety Standard). Interviewees commented on difficulties encountered in meeting NPR requirements on small projects with tight budgets and few FTEs, yet still needing mission critical software development.
    - The internally developed NASA software classification tool was mentioned as useful in helping to eliminate confusion. Another Center mentioned difficulties in using NASA's software classification schema as well as the software classification tool. This interviewee has hopes that a soon-to-be released xPR will help remedy divergent points of view in this area. A couple of other Centers interviewed didn't seem to have difficulty in using the software classification schema.
- **Up front tailoring:** All five NASA Centers described formal upfront tailoring approaches with involvement of designated approval authorities (called Software Technical Authorities). In each of these cases, the tailoring was formally documented, approved, and involved a software compliance matrix.
- **Items only found at single NASA Centers, but noteworthy include:**
    - Specific processes documented for the use on Agile development efforts.
    - Certification of mission critical software engineers via a Center training program.
- Some of the Centers commented that the measures put in place in response to the 2010 *Top NASA Software Issues of 2010* study provided useful progress in solving some of the policy related issues (*NASA-HDBK-2203 Software Engineering Handbook*, Training on *NPR 7150.2A*, etc.).

5.1.3. Observations

- **Policies and processes in universities:** Among the organizations interviewed, only the universities (excluding the off-site university R&D laboratories) lacked internally established software policies, requirements, standards [including Voluntary Consensus Standards (VCS)], procedures, and processes. There appears to be a gap in the awareness and use of standard software engineering practices within the university environment.
- **Flow down of NASA's software NPR to Center xPRs:** The trickle down of top-level NASA software policy and requirements to Center-level direction is slow. The average time lag from Agency-wide software requirements release to Center-level requirements adoption appears to be over two years.
- **Contracts:** NASA should be more consistent in the flow down of software requirements on contracts (see also Section 5.2, Acquisition of this report).
- **Compliance checks:** In-house quality assurance (QA) plays a key role in compliance checks for adherence to software policies and requirements.
- **Principles:** Several NASA Centers are finding value in the documentation and use of engineering principles with respect to software.
- **Classification:** Software classification schemas were relevant for organizations responsible for a wide variety of systems, but not necessarily for military systems where all aspects of the system are critical. The software classifications schemas encountered in industry and R&D Labs didn't appear to be fundamentally different from NASA's approach.

5.1.4. Recommendations

PO1 Improve policies and processes with regard to universities. Although this is a small sample, there is an important distinction between University versus University R&D Laboratory software providers. It can't be assumed that university graduates have a firm foundation in the awareness and use of common software engineering practices. This has implication for co-op assignments and in-house training of new hires by NASA. Recommendations include:

- NASA should be proactive in filling the knowledge gap in common software engineering practices for new hires and co-ops.
- Be aware of the risk of ad hoc software development practices when evaluating proposals from universities.
- Work through the STEM program with universities in strengthening

education in the use of common software engineering practices and standards.

PO2 Improve the flow down of NASA's software NPR to Center xPRs. Establish an Agency-wide 1 year time limit/grace period on flowing down approved NASA Procedural Requirements (NPR) updates into approved Center level direction via xPRs (i.e., NASA Center 'x' Procedural Requirements).

PO3 Improve contracts. Recommendations include:

- NASA should examine the details of how defense services organizations maintain consistency in the flow down of software requirements through contract vehicles, then create and implement standard language with respect to the Agency's software requirements.
- Since in-house SQA plays a key role in compliance with policies and standards, NASA Request for Proposals (RFPs) should request information on supplier's quality assurance (QA) capabilities and use it as one of the evaluation factors in contract awards.

PO4 Establish corporate/enterprise-wide processes. The corporate-wide software engineering strategy communicated by two aerospace industries should provide a model for future NASA software engineering improvements (with appropriate tailoring to ensure it provides benefits within NASA's environment).[4]

PO5 Collect and publish a set of well documented software engineering principles/rules from the NASA Centers, to promulgate software lessons learned in a natural periodic manner available to all NASA Centers and projects.[4]

[4] The NASA Headquarters Office of Chief Engineer began an initiative called Agency Processes and Principles for Software (APPS) under the leadership of Sara H. Godfrey and Steve Larson to develop a NASA capability in this area.

5.2. Acquisition

5.2.1. Questions

- Do any of your projects include software acquisitions as a deliverable or as a piece of the complete software project? If so, could you describe your acquisition process, specifically how it integrates into your software organization?

5.2.2. Discussion

In 2010 NASA identified missing flow down of software engineering requirements[5] as a potential Agency systemic issue. This issue involved both the internal-NASA flow down for work performed at NASA Centers as well as work contracted out to suppliers. In a separate NASA study, a representative identified “Insufficient attention to software on contracts,” in the top ten software issues for the Agency[6]. In an effort to remedy these problems and learn from the experience of other organizations, questions related to acquisition and flow down of software engineering requirements were included in this Benchmark.

Five NASA Centers were interviewed in this Benchmark. Based on these inputs and related data from the above mentioned NASA software issues (Appendix B) an Ishikawa Diagram was developed to sort feedback into seven major categories for analysis of potential root causes (see Figure 1). Sufficient inputs related to each of the seven categories indicate that acceptable solutions for this issue will need to be multifaceted within the Agency to be successful.

[5] "Systemic Software Issues found during OCE Surveys and OSMA QAARS", Martha Wetherholt & John Kelly, NASA Safety Center’s Audits and Assessments Office (AAO) Workshop, Houston, TX, Nov. 2011.

Note: QAARS - Quality Audit, Assessment, and Review

[6] "Software Engineering Improvement at NASA: Past, Present, & Future", John Kelly & Tim Crumbley, Systems and Software Technical Conference, Salt Lake City, UT, April 2012.

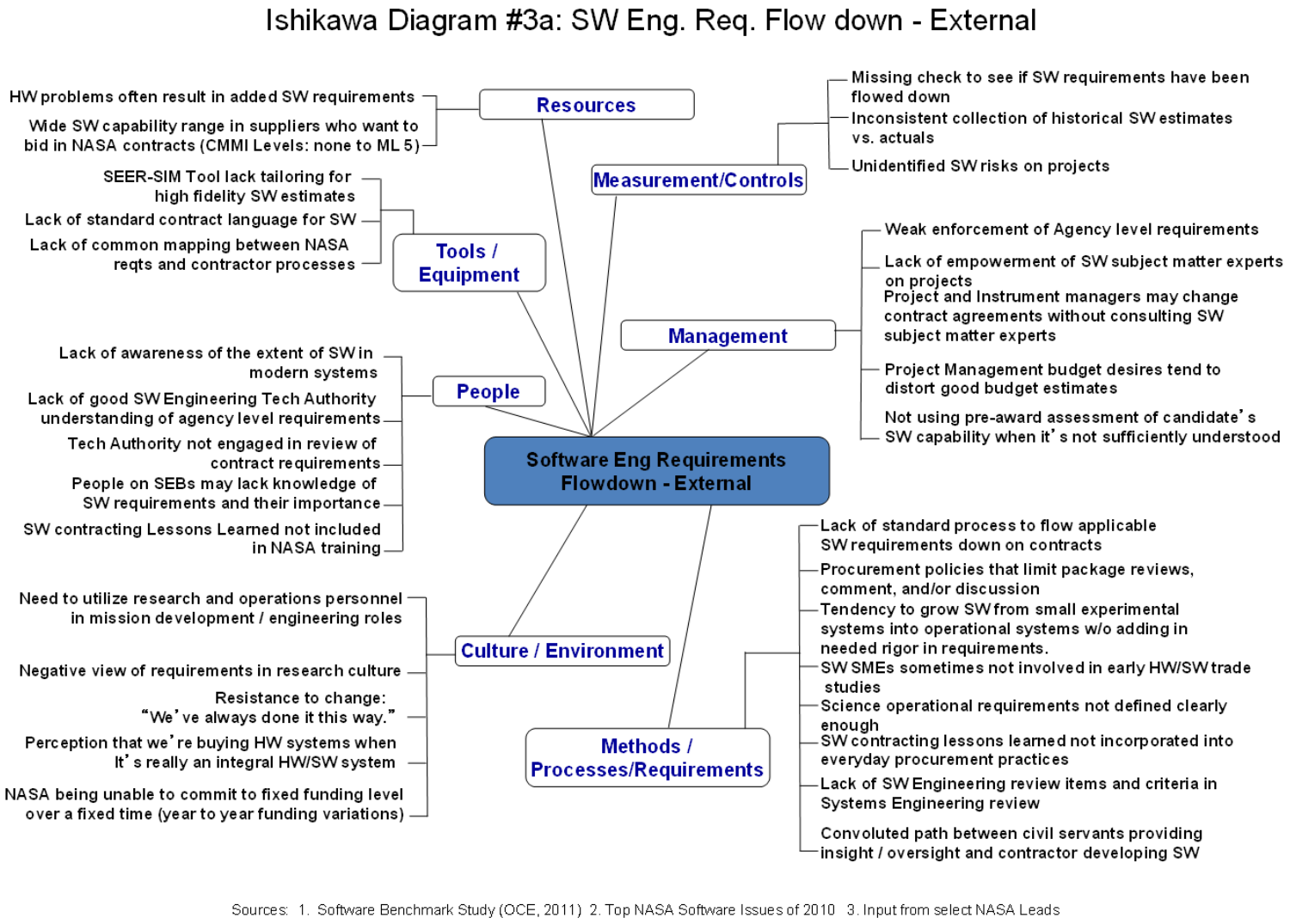


Figure 1: Ishikawa Diagram Mapping Potential Root Causes When NASA Software Engineering Requirements Fail to be Adequately Included in Contracts.

5.2.2.1. Aerospace Industries

The four industry organizations provided information from mostly a supplier perspective. One of their issues was the difficulty on reaching agreement with NASA on software cost estimates. Major industry partners have become sophisticated in estimating cost and schedule for the software portion of their product with good track records of planned versus actuals. Some of the comments received in this area include:

- The company has measurable process estimation – within 10% variance. Every NASA project challenges the amount of code, time or schedule.
- NASA cuts good estimates and introduces risk.
- NASA forces the project (contractor) into a “smaller” box, then forgets who made the decision.
- NASA starts with the budget to create a project instead of finding out how much budget it will take to do the job.

5.2.2.2. Universities and University Research and Development Labs

The four university/university labs varied widely in the size and scope of the software supplied to NASA. In many cases, software was related to the flight or ground aspects of science instruments. One of the frustrations expressed was the unpredictable nature of NASA’s year-to-year funding:

- Each instrument was cost-capped. Instruments were doable within cost, but schedule was stretched out a year due to sponsor’s funding short fall. This significantly increased the cost of the project well beyond the original proposal.
- Being able to commit and fulfill a fixed funding level within an agreed to fixed time would make projects less expensive, as well as less annoying for the supplier. The European Space Agency (ESA) was mentioned as a sponsoring agency similar to NASA who took this approach.
- An interviewee mentioned that they would like to see NASA scale down documentation requirements for smaller projects that are typically contracted to universities. They mentioned the need for flexibility in acceptance of combined project documentation, wiki based documentation, and alternatives to extensive slide based reviews. Similar to industry, universities mentioned:
    - NASA forces the project into a “smaller” box, and then forgets who made the decision.
    - NASA starts with the budget to create a project instead of finding out how much budget it will take to do the job.

5.2.2.3. Defense Services

The four defense services organizations interviewed didn't appear to have the same problems with the flow down of software requirements on contracts as indicated by some NASA Centers with regards to Agency requirements. The Department of Defense (DoD) has a number of acquisition courses, guides, and support materials available through its Defense Acquisition University (www.dau.mil). In many cases individual DoD software organizations provide further policies and requirements applicable to in-house and out-of-house development within their specific application area. The Defense Contracting Management Agency (DCMA) is regularly utilized to oversee contracted software development and maintenance efforts. Defense services organizations appear to make wide use of standard contracting language for acquisition containing software.

5.2.2.4. NASA

Five NASA Centers provided inputs on experiences in acquisitions containing software deliverables. Some of the noteworthy comments from NASA Center interviewees during this Benchmark include:

- One interviewee recommending against invoking *NPR 7150.2* in a contract with a single sentence. Instead they recommended including it in SOWs by stating the applicable requirements from 7150.2 as well as other NPRs. One of their contractors didn't realize what they were signed up for based on a one liner in the contract with respect to *NPR 7150.2.*
- At one Center after a significant omission of software requirements on a contract, the Center's Contracting Office adopted the default position that "a project has software, unless proven otherwise."
- A couple of Centers recommended including a standard set of compliant data requirements documents for acquisition projects that are put into the Software Management Plan which is subsequently put on the contract.
- When applicable, Centers mentioned putting a requirement for CMMI on RFPs and contracts. This improves communication and expectations between acquirer and supplier.
- One of the interviewees communicated the lesson learned that NASA should ensure the contract management team includes domain experts, including software, during the acquisition process.
- A separate NASA study, *How Does NASA Estimate Software Cost? Summary Findings and Recommendations* by J. Hihn, et.al 2012, indicates the need for the

Agency to significantly improve its internal capabilities in software cost estimation for project planning, monitoring, and control.

5.2.3. Observations

Two main findings arose from the benchmark visits on the topic of software acquisition:

- NASA appears to be behind defense services organizations in using standard contracting language for acquisitions that include software.
- Recognition and use of good software estimates during project planning is a NASA management weakness.

5.2.4. Recommendations

Recommendations for software acquisition improvement include:

AQ1 Standardize contract language for software. Develop standard NASA contracting language to ensure software requirements are consistently flowed down on contracts that include software development or maintenance. Defense services and the ESA have contract language for software that should be examined in the development of standard NASA language. Contracting language should reflect any pre-tailored requirements to enable allowances for alternative documentation and review approaches [e.g., The approach of using a standard set of compliant data requirements documents was mentioned by a couple of NASA Centers, as were product data and life cycle management (PDLM) tools approaches.]. The contracting language should address subcontracting situations to ensure software requirements are flowed down to subcontractors from primes. This effort should leverage and enhance the software acquisition guidance provided in *NASA-HDBK-2203 Software Engineering Handbook* (Topic 7.3). While the Handbook provides needed guidance among the software community, the adopted standard contract language needs to be institutionalized within the NASA acquisition community set of common practices.

AQ2 Provide accurate and trusted software cost estimates. Improve the fidelity of NASA's cost estimates for software and utilize it to reach agreement with industry partners. NASA needs to enhance its capability in trusted software cost estimates to accurately evaluate contractor software estimates and make smart project trades. Adequate planning funds also need to be put in place by project management to ensure there are resources to perform better software cost

estimates.

AQ3 Improve the NASA contracting process to adequately address software:

- Introduce a check in the contracting process to ensure adequate software requirements have been included.
- When contracting for systems, knowledgeable software personnel need to be involved to ensure adequate agreements are put in place.
- Ensure representation or advice from software experts in the acquisition of systems that depend on software.
- When applicable, use CMMI to better communicate NASA's software needs and expectations on contracts.
- Secure and maintain adequate budgets to fund trusted software estimates.
- Clarify what is acceptable to NASA in terms of "equivalence" and "meets or exceeds" in the area of software requirements. Leverage the work performed in 2012 under the special NASA task which mapped the Agency's software requirements to voluntary consensus standards.

AQ4 With regard to training and guidance, recommendations include:

- Improve NASA acquisition training as it relates to supplied software (train personnel in standard software contracting language, applicable requirements, and cost estimation).
- Utilize the *NASA Software Engineering Handbook* and its guidance on acquisition and contracts. Ensure the Handbook is available to contractors to facilitate better communication and understanding of NASA's expectation in meeting software requirements.
- When applicable, use CMMI to better communicate NASA's software needs and expectations on contracts. Including the expectation of increased accuracy in software cost estimates from organizations at higher maturity levels (3 and above).
- Better awareness and use of the Federal Acquisition Regulations (FAR) supplemental clauses concerning software and data rights (at the Agency and Center levels) available to place on contracts.

AQ5 Clarify NASA's Software Requirements. In the 2014 update of *NPR 7150.2, NASA Software Requirements*, it is recommended that inherently governmental requirements be clearly labeled to eliminate confusion when *NPR 7150.2* requirements are put on contracts.

5.3. Testing

5.3.1. Questions

- Please describe software testing: include strategy and scope, test plans, testing types, success/completion criteria.
- What is the organization and composition of your typical software test team?
- Please identify any tools used or autonomous testing performed?

5.3.2. Discussion

Developing software is a complex task and there are risks involved in the development process. The risk is directly influenced by the software testing effectiveness and efficiency. Various problems affect the software testing process. In order to build robust software and to have efficient software testing processes and methodologies, it is essential to understand the potential software testing problems. Often the cost and schedule for software testing is under-estimated and under-qualified personnel are assigned to perform software testing. Software testing should be done by skilled testers who have both the technical as well as domain knowledge. Schedule pressure can directly influence the testing quality. Deadlines negotiated by the management from the stakeholders will directly affect the nature of testing. If the tester is under pressure, there is a higher risk of failing to find defects and/or schedule slippage which can adversely affect a project. Maintaining a healthy balance between both schedule and cost can be a major problem. Software testing should include metrics for project monitoring and control as well as improving an organization's ability to produce quality software. The intent of these benchmark questions was to assess how software testing is being addressed and handled by the organizations being interviewed. General note-worthy software testing lessons include:

- **Software test approaches:** "Test-as-you-fly" is a common approach across organizations. The principle of "test-as-you-fly" means that tests and simulations accurately reflect the planned use of the software. It also includes testing the planned mission profile along with off-nominal scenarios. Testing of all critical mission-operation elements as they will be flown greatly reduces the risk of encountering negative impacts on mission success, from partial to full loss of mission capability.
- **Software testing challenges:** Many organizations identified the availability of hardware test time prior to software delivery as one of the biggest software testing challenges.

- **Testing of commercial-off-the-shelf (COTS) and government-off-the-shelf (GOTS):** Some of the organizations interviewed tested COTS software (tool kits, library software, and open source) at the same level as developed software. In most organizations, COTS and GOTS go through the same QA process as developed software. Other organizations did not do any separate testing for purchased operating systems or COTS. Part of the decision process in deciding to purchase COTS is to consider the supplier's testing. One organization used a strategy for real-time operating systems (RTOS) specification of obtaining a DO-178B certifiable operating system. For GOTS testing, one organization's test team doesn't repeat previous tests done by the original developer, but instead runs a comprehensive set of tests in the context of the full build of the project's application of the GOTS software.
- **Software test time allocation:** As indicated by Table 2 below, most organizations estimated the percentage of time on software testing to be between 30 to 50 percent of the development life cycle. A rule of thumb used by one organization was to plan to test twice as long as you code.

Table 2: Software Test Time Allocation

Organizations	% of time on software testing
NASA Centers	21% to 40%
Defense Services	25% to 50%
Industry	35% to 50%
University/University Labs	Percentage ranges not available. One organization offered their rule of thumb for planning the amount of test time was to "test twice as long as you code".

- **Data on software testing approaches and a summary of software testing metrics collected:** Table 3 shows the typical test metrics that were collected by the organizations interviewed.

Table 3: Test Metrics for the Organizations Interviewed

	Organizations																	
	N1	N2	N3	N4	N5	I1	I2	I3	I4	I5	G1	G2	G3	G4	U1	U2	U3	U4
Test Information																		
Estimated % of time used for testing	30%		40%	21%	30%		40-50%	40%		35%		25%		40-50%				
Static analysis tools used in software testing	X		X	X	X	X	X	X	X	X	X		X	X	X	X		X
Uses an independent test team	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X		X
Test Metrics Collected																		
KSLOC	X	X	X	X	X	X	X	X	X		X	X	X	X		X	X	X
Defects	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
Defect Density				X	X	X	X	X	X				X	X				X
Phase containment					X	X	X	X	X				X					
Effort /Phase				X		X		X	X			X	X					
Effort/Activity or process					X	X		X		X		X	X	X				
Tests performed/passed	X	X	X	X	X	X	X	X	X		X	X	X	X		X		X
Requirements verified	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Schedule variance	X	X	X	X	X	X	X	X	X	X		X	X	X				
Cycle time						X			X	X		X						
# change reports		X	X		X	X	X	X		X								
QA audit findings	X	X	X	X	X	X		X	X		X		X	X				X
Development progress	X	X	X	X	X	X	X	X	X		X	X	X	X		X		X
Non-compliance Delinquency				X		X							X	X				
Test coverage (% code tested)										X								
CMMI Level - at interview	2	2	2	3	3	3	5	3	3	-	#	2	5	3	-	-	-	3
Previous CMMI Level 5 Org.						X			X			X						

Nx	NASA Centers	Ux	University, Univ. R&D Labs
Ix	Industry Organizations	Gx	Defense Services
	- Not Applicable		

Note: X's in matrix indicate practices or metrics mentioned in interviews. Lack of X's in boxes DO NOT indicate metric is not collected or practice is not performed. The chart should be viewed merely as an indication of the practices mentioned often and the metrics that seemed to be commonly collected. For U1, no information was received. # indicates organization had a previous CMMI Maturity Level 2 rating, which was expired at time of interview.

- **Software assurance role during software testing:** The role of software assurance was not consistently defined across the organizations interviewed. Some organizations had a software assurance role that used a sampling strategy during software testing to determine the level of assurance involvement that ranged from witnessing a subset of tests to reviewing a subset of test reports; some organizations did not require software assurance sign-off on each test case; some software assurance groups sampled software test reports; and a few organizations had a representative from software assurance witness all formal testing. A number of organizations did software assurance as a part of the engineering activities and did not have separate organizations for SQA.
- **Test team characteristics:** Most organizations use an independent test team which has an independent reporting chain from the developers. Independent testing was also done for small projects while large projects had independent and also specialized teams. Some organizations were able to use operations personnel in software testing, and considered the use of operations personnel as a best practice.

- **Peer reviews:** Most organizations used peer reviews extensively to catch defects early (including technical peer reviews at the unit code level). Peer reviews took a fair amount of time, but they were highly valued by interviewees.
- **Software test philosophies:** Several unique testing philosophies were identified from the interviews, including:
    - Purchase all boards at once, so that they all have the same configuration. This helps ensure that the boards all have the same firmware versions.
    - Foundation documents for testing are the Software Development Plan and the Software Test Plan.
    - The software integration and test plan and strategy should be done by Preliminary Design Review (PDR).
    - Test procedures should be ready by Critical Design Review (CDR).
    - Test results should be available as soon as the test is run.
    - Projects that performed design testing early in the project caught a number of defects early in the life cycle.
    - Test by breaking down the functions and testing them individually.
    - Unit test as the code is developed; perform informal testing during the code development cycle.
    - Defining test resources is essential to successful software testing.
- **CMMI and tools:** CMMI observations regarding software testing are discussed in the CMMI section (Section 5.7) and information on software test tools is covered in the Software Tools section (Section 5.9) of this report.

5.3.2.1. Aerospace Industries

The five industry organizations provided a number of useful lessons learned and good practices in software testing.

A few industry organizations used common metrics across all development sites, held monthly reviews, and took corrective action as needed. Higher rated CMMI organizations (Maturity Levels 3 through 5) seemed able to test more efficiently, possibly due to use of metrics during testing and the use of more mature testing philosophies. Metrics were also able to improve the overall software processes when common metrics were applied consistently across development organizations. One of the key metrics found in use was defects per thousand of source lines of code (KSLOCS). Metrics were also noted for testing from the other organizational groups as well. Table 3 includes testing metrics identified during the benchmark interviews.

Industry organizations typically had unit testing in place as early as possible. Testing was considered more difficult if unit testing was not done initially and as the software was developed. The following items were considered part of unit testing in one industry organization: path testing, boundary testing, and requirements flow down. For this organization, a complete trace of requirements through testing was done and a requirements traceability tool, like the IBM Rational Dynamic Object-oriented Requirements System (DOORS), facilitated requirements traceability. Off-nominal testing was included by some organizations because it was considered more productive in finding problems than nominal testing. One organization completed a test readiness review and a dry run of the testing before formal testing started. Inspections were also discussed as having a significant and possibly greater impact on defect removal.

Some interviewees recommended having adequate test resources to facilitate more test cases, faster turnaround and ability overall to complete more testing. It is critically important to have software simulations and hardware in the loop for testing, and simulation skills were considered vital to enable testing. Also noted was the importance of verifying test software. In at least one instance, test software was verified by doing peer reviews and using the software.

With respect to COTS, many industry organizations tested COTS. Requirements satisfied by the COTS were documented and then the software was tested against those requirements. Some regression testing was also found and one group validated COTS in the context of the system only.

5.3.2.2. Universities and University Research and Development Labs

The universities and university R&D Labs interviewed provided a number of useful lessons learned and good practices in software testing.

Generally most university and university Labs did not use path or test case coverage tools nor did they use automated testing techniques. Inspections were relied upon heavily and it was noted that testers should know static and dynamic testing and analysis tools.

One of the university/university lab organizations explained that test plans are done by PDR, including development test plan and ground test plan. Another organization created an “accredited” software development environment. And one of the biggest challenges noted was getting enough instrument test time before delivery. Similar to the defense services organizations, operators/users were often included on the test teams.

5.3.2.3. Defense Services

The four defense services organizations provided useful lessons learned and good practices in software testing. There were slight variations in the test team compositions; however, the test team comments were similar to those from industry and the NASA Centers. Almost all defense services organizations had independent test teams with many of them including users. One organization noted that testers for small projects could include a developer but not for the portions that developer had developed.

At least one of the defense services organizations had standard procedures available across the organization on how to write test plans, procedures and provide status reports. Test plans were started as early as requirements were established. A few of the best practices found include:

- Unit test data and results were delivered with the code.
- Progressive testing as modules were being built.
- Complete full regression testing in areas with changes and, when some aspect of the code cannot be tested, notify the next test level to ensure it gets tested there.

Extensive use of peer reviews, including technical peer reviews at the unit code level, and regression testing was common among the defense services organizations. Predictive test tools were noted but not extensively or commonly used across the defense services organizations. Some defense services organizations used some predictive test tools, but most relied more on regression tests. One organization was able to estimate defects from historical defect data and continued to compare estimated with actuals. In one interesting discussion regarding COTS and GOTS, it was learned that COTS and GOTS all go through the same QA process as in-house developed code. In addition, sometimes test results on the COTS and GOTS are received from the vendor.

5.3.2.4. NASA

The five NASA Centers provided useful lessons learned and good practices in software testing, and most NASA software testing appears to be done well. Test plans are started very early and follow *NPR 7150.2A* (SWE-104) requirements. Developers typically write their own unit tests. Criteria for test completion are as follows: peer review and inspection for build testing, coverage testing of requirements (100% or waiver required), trending curve of test results for each module, and coverage metrics provided by the Simulink tool. “Test-as-you-fly” or “Fly-as-you-test” is essential to the testing approach (as well as end-to-end testing). Metrics are being used but have a limited effect on organizational testing resource decisions and test planning decisions.

The COTS and GOTS testing approach is that the software test teams do not repeat previous tests done by the COTS or GOTS developer, but instead run a comprehensive set of tests in the context of the full build of the project's application of the COTS and GOTS Software. Predicted software defect data is not generally used by NASA projects, and automated test scripts (not including tools) are used at several Centers. Having simulators is important to software testing success.

5.3.3. Observations

The main observations that arose from the benchmark visits on the topic of software testing are:

- Some organizations considered inspections/peer reviews to have the largest impact on defect removal, followed by testing.
- Better use of software metrics could highlight areas where software testing productivity and software quality could be improved.
- Having software test personnel involved early can be effective from both a technical and cost savings standpoint.
- Limited use of predicted software defect data is occurring across the community.
- The role of software assurance in software testing is not consistently defined.
- Hardware availability and simulator availability is necessary in software testing.
- Software COTS testing requirements, approach and guidelines are not clearly defined. They varied from complete requirements testing, to testing as part of a system, to no testing, and just requesting test data from the vendor.
- Software test activities should include users and operators.
- Software simulation skills are a critical skill for software testing.
- There is reduced risk and reduced cost in software testing when using higher CMMI rated organizations (level three or higher) on software projects. Higher CMMI rated organizations seemed to be able to test more efficiently and had reduced defect rates.
- Static analysis tools have become commonly used across the software development community.

5.3.4. Recommendations

Recommendations for improving existing NASA software testing techniques include:

TE1 Develop a set of predictive software defect data and a process for assessing software testing metric data against it. Use the set to status progress during

NASA software testing phases and in software test reviews.

TE2 Identify a recommended set of test metrics for NASA software development. Assess whether code coverage is a viable metric for use in software testing reviews. Better use of software metrics could highlight areas where software testing productivity and software quality could be improved.

TE3 Review options for improving the NASA software COTS testing requirements and guidelines when the NPRs for software engineering and the associated NASA handbook are updated in FY14.

TE4 Assess the option of buying Agency-wide licenses for commonly used software test and static analysis tools.

TE5 Clarify the role of software assurance in NASA's software testing activities.

TE6 Perform a workforce assessment of the software simulation skills of NASA personnel and provide recommendations based on the findings from the assessment.

5.4. Assurance

5.4.1. Questions

While this benchmarking activity emphasized software development, the following software assurance topics were touched as part of the discussion:

- How is software engineering integrated with software assurance?
- What software quality, reliability, and safety activities take place – and who performs them?
- Is there a software assurance role for supplier software?
- How are the software assurance personnel trained?
- What software assurance tools are used?
- What software assurance metrics are gathered?

5.4.2. Discussion

Software assurance at NASA appears to be a well-defined set of disciplines. These are given as software quality, reliability, safety, verification and validation (V&V), and IV&V

in *NASA-STD-8739 NASA Software Assurance Standard.* Assurance requirements and practices are further elaborated over additional key assurance documents:

- *NASA-STD-8719.13B NASA Software Safety Standard*.
- The Software Assurance, Software Safety and Complex Electronics Guidebooks.

Software assurance at NASA also has strong institutional backing, including support from OSMA at NASA headquarters, NASA IV&V and the NASA Safety Center (NSC). It also has the Safety and Mission Assurance Technical Excellence Program (STEP) for assurance education and the Software Assurance Research Program (SARP) to advance the practice. In addition, it has a strong relationship with the OCE and the SWG, and has its own working group, the Software Assurance Working Group (SAWG), to address issues through the practitioner community.

Yet despite these resources, there are questions about how software assurance is actually practiced at the NASA Centers, their suppliers, and in the industry at large. Issues are often raised at SAWG meetings about what should or should not be included in the practice, about the adequacy of funding, about relationships with the projects and the software developers, and even about "how to sell software assurance." All of these raise broader questions about what is and is-not software assurance and its value and place in developing NASA's software-intensive systems. Results from this benchmarking activity provide some insight into these questions.

The main assurance activities reportedly performed by the benchmarked organizations can be seen in Figure 2, below. The numbers show how many of the 18 benchmarked organizations reported having software assurance involved with each activity listed. Note that two universities had no formal assurance role at all. Also note that the activities are not counted if performed by other roles. For example, engineers rather than assurance personnel are often assigned to software safety and reliability.

Although some activities may not be reflected in the numbers (topics missed in the discussion), they provide a starting point for examining software assurance as documented versus the actual practice of it.

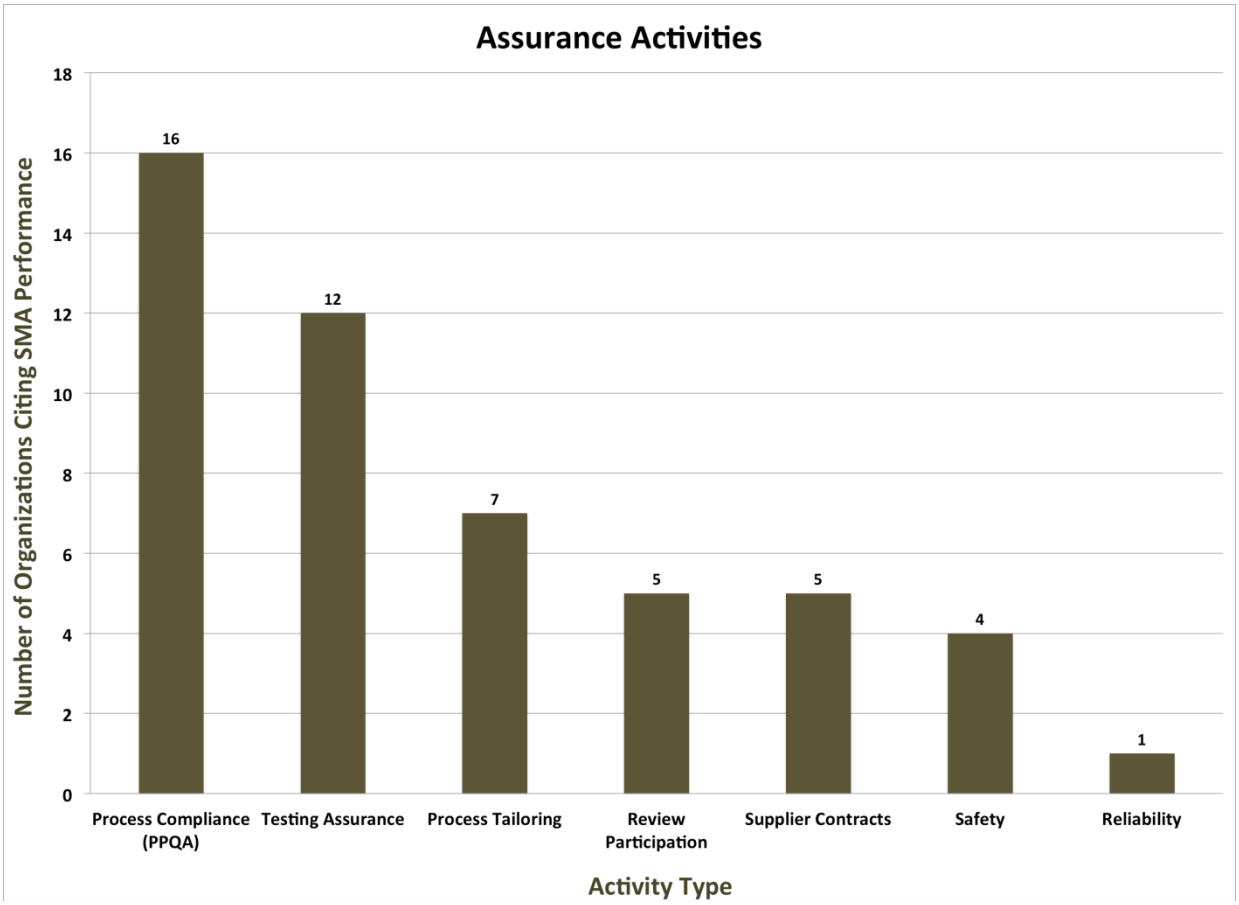


Figure 2: Activities Performed by Software Assurance

5.4.2.1. Aerospace Industries

The Five Industry organizations interviews can be summarized as follows:

- All industry organizations reviewed saw the main function of software assurance as performing process and product quality assurance (PPQA) and maintaining software compliance with institutional standards. Safety and reliability were seen as engineering roles.
- These organizations also tended to have a low ratio of software assurance engineers to the number of developers. For example, one organization had five to six software assurance engineers for about 200 developers. At the far end, an organization had only one “SQA person” for a 100-person software engineering project.
- The high CMMI process maturity of most of these organizations might be a factor in their perceived need for assurance. All but one had been appraised at CMMI

ML3 or higher and, as one noted, greater process maturity means more repeatable, institutionalized processes and fewer audit findings. The one organization that hadn't used the CMMI also used one assurance person for a team of 15 developers and 4 testers – the highest ratio in the group.

- The industry organizations tended to use tools and metrics on the engineering side. Two of the organizations mentioned wide use of six-sigma, which also correlates with high CMMI maturity.
- Three out of four organizations mentioned receiving support from NASA's IV&V center.

5.4.2.2. Universities and University Research and Development Labs

Two of the university organizations interviewed were in a typical university environment, while the other two were specialized off-site R&D laboratories run by universities. Notable information from these organizations include the following:

- This group tended to have less awareness of software QA. Two of the organizations had no independent software assurance. The first stated that they "had no formal QA," but try to have developers define and check their own processes.
- The second organization said they were "not sure what software quality means," but did run V&V tests and had external peer reviews.
- The other two organizations work with NASA Centers and have development processes and independent assurance organizations. Both of the latter two organizations have process/PPQA audits and witness or otherwise assure testing.
- The largest of the university organizations has a CMMI ML 3 rating. It also had a post-launch Software Systems Assurance Manager, which was unique among all the respondents.

5.4.2.3. Defense Services

Defense services organizations were interviewed, and their inputs on this topic can be summarized as follows:

- All the defense organizations had been appraised to some CMMI level; two had achieved CMMI ML 5 at some point, with one maintaining certification.

- Following a similar pattern to the industry organizations, all four organizations used software assurance primarily in a PPQA role and did not discuss their role in reliability or safety.
- Three out of the four organizations also used software assurance to witness or otherwise assure software testing.
- The one CMMI ML5 organization maintained a process assurance group, dedicated to process compliance, and QA and IV&V groups for checking products.
- Of the three organizations that discussed FPGAs, none mentioned software assurance.

5.4.2.4. NASA

Five NASA Centers were interviewed in this Benchmark. The following trends were identified, based on these interviews:

- NASA Centers tended to involve software assurance in a greater range of development activities. Four of the five assurance organizations were involved with process tailoring, in addition to the PPQA audits.
- Four out of the five (not the same four) were witnessing or otherwise assuring that tests were performed properly.
- Four out of five NASA Centers also performed some assurance activity related to software safety, even if only to run the litmus test.
- Three out of five of the NASA Centers used assurance personnel to monitor suppliers or software contracts in some way.
- Tool use and metrics collection seemed to apply mainly to the engineering side of the house rather than assurance. This was common across all the benchmarked organizations, regardless of type.
- All the NASA Centers have either started using or said they intend to use the NASA STEP program for training (see strengths, below).

5.4.3. Observations

- It is commonly observed that software assurance is resource limited. However, the scope of assurance activities varied widely across the benchmarked organizations.
- Software assurance organizations are involved to varying degrees in the software assurance disciplines as defined by NASA. The software quality

discipline is the most established as part of assurance; reliability and safety are the least, often assigned to engineering; V&V is in the middle, if it is taken to mean assuring/witnessing the tests rather than performing them; and IV&V seems to have found a place for large, mission/safety-critical systems.

- Software assurance requirements, like engineering ones, need to be consistently flowed down to contractors. More assurance involvement with supplier contracts would help.
- There are questions about engineering versus assurance use of tools and metrics. The practices seem more developed on the engineering side, but there are questions about who is chartered to do what. For example, are static code analyzers assurance or engineering tools, and who should collect the defect metrics?
- As mentioned in the Policy section (Section 5.1), universities were weak in internal software policies, requirements, procedures, and processes. This extends to software assurance, with two organizations having no real assurance at all.
- There is still confusion about whether or how software development and assurance requirements apply to PLDs.
- There appears to be a lack of awareness of the relationship between software and safety.
- Strengths:
    - NASA Centers are adopting the NASA STEP program for their assurance training.
    - NASA IV&V is being utilized for large, mission/safety-critical systems.

5.4.4. Recommendations

The following are recommendations for NASA Software Assurance:

AS1 Follow up this benchmark study with a deeper look into what organizations perceive as the scope of software assurance, the value they expect to obtain from it, and the shortcomings they experience in the current practice.

AS2 Improve QA awareness at universities as well as among project managers and engineers at organizations interfacing with NASA:

- Work needs to be done to better communicate the definition and scope of software assurance, with an emphasis on its value to projects, so that adequate resources can be allocated to it.

- Consider using the NASA STEP program and other means to reach out to workplace training and university software engineering/computer science programs. This is an opportunity for NASA to infuse its view of software assurance, educate its suppliers, and cultivate the next generation of practitioners and users.

AS3 Identify the tools and metrics needed to do a better job of assurance, and work out with engineering the best way to share them.

AS4 NASA RFPs should request information about the supplier's QA capabilities and use that information in deriving risk exposure and assessing the degree of supplier surveillance needed.

5.5. Training

5.5.1. Questions

- How does your organization train and develop software engineers and software quality engineers? Would you describe the 3 most beneficial classes for software engineers?
- Organizational responsibility for development or acquisition of training curriculum.
- Type of training given: in-house versus external training programs (or combined)? What has led to using in-house versus external training?
- Who has responsibility for identifying the training needs, how and when training is given?
- Is training given at individual, group or project level, career levels? If so, what kind and when?
- Is training mandatory or optional? How much time is allowed or expected for training per person?
- How is mentoring and on-the-job training (OJT) included in developing individuals (informal, structured, required)?
- What are your preferred methods and media for training?
- Describe how your training program addresses proficiency training, system engineering, metrics, risk management and project management.
- Describe any training provided to management (line management and/or project managers)?
- How does your organization manage training that might be needed for a specific project (just in time training)?

5.5.2. Discussion

The chart below shows some of the training characteristics discussed by the organizations interviewed. The chart shows that most organizations used several methods of training. A few organizations relied more heavily on hiring in the appropriate skills and mentoring the new hires.

Table 4: Organization Training Characteristics

	Organizations																	
	N1	N2	N3	N4	N5	I1	I2	I3	I4	I5	G1	G2	G3	G4	U1	U2	U3	U4
Mentoring/OJT	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X		X
On-line training			X		X	X	X		X		X	X	X			X		
Classes with exercises	X		X	X		X		X				X	X	X	X			
Tiered training program									X				X					X
Required periodic training					X	X	X		X		X	X	X	X				X
Lunch time seminars		X	X	X			X								X			
Continuous training program					X	X	X		X		X							
Relies on hiring knowledge										X		X	X	X			X	X
Role-based training			X	X	X	X	X	X	X		X		X					
Certifications for software				X			X									X		

Nx	NASA Centers	Ux	University, Univ. R&D Labs
Ix	Industry Organizations	Gx	Defense Services

5.5.2.1. Aerospace Industries

Of the aerospace industry organizations benchmarked, four of the five had many similar practices and all four of them used a variety of different training delivery methods and types of training. All four mentioned having common training across their enterprises (in one case covering 12 different sites). The common training ranged from providing a few basic classes like IT security or ethics to a more complete set covering organizational processes. Several of the industry organizations had role-based training developed by their software engineering process groups (SEPGs). Several of the organizations had a skills assessment process to determine whether training in a particular skill was needed. One organization mentioned a training tool that sends out reminders on required training.

Training delivery methods among the group of four aerospace industry organizations included classroom training, on-line training, and lunch and learn seminars. One of the organizations commented that the method of training depended on the situation and the

class. A new hire was more likely to be given classroom training while someone more experienced would take the class online when he needed it.

Mentoring, cross-rotational assignments, and the peer review process also played a significant part in the four aerospace industry organizations' training programs. Three of the four had formal mentoring programs, pairing a more experienced developer with someone less experienced. The other aerospace industry organization used a much less formal mentoring approach and identified subject matter experts who could be consulted for guidance. On-the-job training, such as participation in inspections or use of a new tool also played an important part in completing the training picture.

The most impressive aerospace industry organization in terms of training had training built into their culture, so that it was available when the software personnel needed it. They used both in-house developed classes and externally procured classes and conducted a mix of classroom courses, on-line classes and seminars. Their training was broken up into small "chunks" and provided as part of the daily work, so a training module might be part of a regular meeting. They also stressed one-on-one mentoring. One of the key elements in their training program was training on the organization's expectations, so that the software community learned that the organizational process was "just the way things were done," and they stressed a desire for perfection in their software. Their training program also included a flight software certification program and training for project managers.

The fifth aerospace industry organization used an entirely different philosophy on training for software personnel. Instead of having a highly structured training program, they focused on a very rigorous hiring program where potential employees were given a series of programming tests before they were hired. Once these skilled people were hired, they were mentored and carefully monitored by more experienced team members. The organization's peer review process was one of the OJT activities used by this organization as a training mechanism for software developers. The organization also used a wiki to promote communication and sharing of lessons learned and best practices among developers.

5.5.2.2. Universities and University Research and Development Labs

In general, the university organizations had much less structured training programs targeted for their software personnel. The exception was one of the university laboratory organizations. Like many of the other organizations, they tried to hire people with the right skills, but they followed that up with a program consisting of three types of training and formal mentoring:

- **Global:** The first type was similar to the organizational training seen in many of the industry organizations. Examples were quality, record-keeping, configuration control.
- **Functional:** The second type was by function where each software group decided on the needed training. This type of training included process training and was required every three years.
- **Certifications:** The final type of training was for technical certifications, like electro-static discharge (ESD), crane operators, etc.
- **Formal Mentoring:** Their formal mentoring program was somewhat unique in that it included members of every area of the organization including employees with 20 years or more of service. The program covered a broad range of skills and provided opportunities for experiences like giving a software engineer enough face time with a seasoned mission project manager to help him/her understand some of the software development problems.

The second university laboratory organization did awareness training on their processes, primarily using self-training modules. They kept training logs and had an informal mentoring program to supplement their training.

Both of the university laboratory organizations commented that they would like to be able to take advantage of some of the NASA training. Software safety was one class of interest.

One of the university organizations interviewed didn't offer any sort of formal software training program. They did have an internal wiki site that provided documentation to help the software developers come up to the desired knowledge level. In this particular case, the employees were not even hired with software engineering experience, but for their advanced knowledge in other scientific areas, such as physics. This type of expertise was sought to provide a good knowledge of the domain, a better understanding of the operational needs of the hardware, and a good perspective on the type of information useful to the researchers.

The fourth university organization operated their software development projects in the context of a four-semester course where the students learned the best practices as they went through the life cycle. Initially, they were given classroom instruction on project management, different software methodologies and best practices. They were not required to use any particular processes, unless the customer specified them. The students developed their own processes. A mentor was assigned to each project group to point out potential risks or consequences associated with their choices. The university asserted that the learning occurred by actually doing the software project and dealing with any of the consequences of their project management choices.

5.5.2.3. Defense Services

The defense services organizations also used a variety of different training methods and had a number of organizational classes available to them. For this set of organizations, many of the organizational classes were required. One organization mentioned classes in engineering, CMMI, tool training, contract management, and process awareness as examples of organizational software training. They also mentioned that they try to hire in people who already have the right skills for the job.

A second defense services organization described a set of videos that were shown to new employees before they were allowed to work on any projects. Then they progressed into classroom training with exercises and then computer-based training (CBT). For this organization, mentoring was the preferred method of training, even though they used a variety of other methods. Their mentoring included structured OJT and the use of a mentoring checklist.

The two other defense services organizations used a training point of contact called an advocate or a coordinator to develop the tactical and strategic plans. In one case, the training advocate collected training needs from the projects and if the need was common across the organization, the organization would provide the training. Otherwise the project would provide it. They used automated forms to request and record training. Part of the training included in this organization focused on the good process in the PAL, the monthly lessons learned exchanges, as well as interactive training, on-line training and mentoring. In the other case, there were three types of training, including continuing education, organizational-mandated training, and specialty training, like cost estimation. Most of the training was role-based and on-line training. Eighty hours of continuing education a year was required for software personnel.

5.5.2.4. NASA

The overall training programs at the five NASA Centers interviewed varied considerably. All the Centers have some NASA-required training like IT security and ethics in place. In addition, all the Centers interviewed take advantage of the software classes sponsored through the software area of the OCE. A yearly “wish list” is submitted by the Centers and training dollars are divided among the Centers to try to provide as much training as possible with the limited funding. Centers are expected to augment these training classes with Center-funded training. One Center reported that their Center does not provide technical classes with their training budget, but there is some training funding available at the directorate and project levels where some classes can be procured. There is no overall training plan for coordinating this training and there is no mentoring program at the Center level. Some mentoring occurs in the divisions. At this Center

some software personnel have taken advantage of the software assurance STEP program classes.

A second NASA Center also took advantage of the STEP classes, and the OCE software funded classes, as well as bringing in some of their needed classes through their Engineering Directorate funding. They had no mandatory training at the branch level, but they did have a good mentoring program for the newer employees. They also developed a few of their own classes, like very high-level design language (VHDL) for FPGAs. No software training is available for their mission project managers. This Center stated a preference for classroom classes with exercises.

The remaining three NASA Centers seem to have a more coordinated training program at their software organizational level. One of those Centers reported that they brought in classes on a regular basis, as funding allowed. Most of these classes were role-based classroom instruction. This Center developed some of their own training, including their process training and tool training. Even executives received tool training. Most of the process training was done online. Some training was accomplished through mentoring, where the software leads mentored the new employees. They did not provide software training for the mission project managers.

The remaining two Centers of the three with more coordinated training programs had a structured approach to their software organizational training. Both of the Centers developed their own process training and a series of other classes or workshops, which were offered in the classroom setting. In addition, they conducted short lunch-time seminars on selected topics. There were other required classes in addition to the NASA-required classes at these Centers. One of the Centers mentioned that they had started doing more on-line classes and the other said some of their classes had been video-taped so that students could watch the classes when they needed them. Both of these Centers had developed software awareness classes for project managers. One of the Centers had begun a flight software certification program consisting of three classes, spanning 24 hours of instruction. Topics in Class 1 included data through-put, theories of computing, and finite state machines; Class 2 focused on best practices, design principles, and lessons learned and Class 3 covered managing software risks via the use of good coding standards, static analysis and modeling. The flight software area at the other Center had a very structured mentoring program in place where an employee new to a role was assigned a seasoned mentor who ensured that the mentee mastered a list of pre-defined skills for the role.

5.5.3. Observations

- Most organizations used a variety of training delivery methods, depending on the training topics and the experience of the personnel. Classroom training with

exercises was more likely to be used with new employees while video and on-line training seemed to be preferred by more experienced students, particularly if training is required periodically.

- Alternative training methods including mentoring, on-the-job training, and participation in events like peer reviews, and lessons learned sharing played a key part in producing a well-trained workforce. The most impressive organizations with respect to training used structured approaches to mentoring such as the use of checklists or verification of skills during mentoring and alternative training opportunities like peer reviews where newer employees can benefit from the experience of more seasoned employees.
- Setting the expectations for the software teams and training to meet those expectations is a key aspect of supporting a well-trained, process-oriented software workforce. Organizations that included training on expected customary practices seemed more likely to define their processes as “just the way they do business.”
- Software training is needed at a variety of levels. Certain types of training like IT security and ethics were generally required of everyone. Managers often were given software process training and awareness training on software development and issues. More critical software areas often had required certification requirements. Many organizations had required periodic training.
- For the NASA Centers, the OCE/Software-sponsored training and the STEP program training greatly enhanced a Center’s ability to provide the needed software training. In a few cases, the training provided by OCE/Software and STEP were essentially the bulk of software training available for software personnel at those Centers.
- The use of on-line training, videos and the practice of breaking training into small modules to be offered separately, allows the training to be provided exactly when it is needed and with minimal disruption to project schedules. In one case, project members received regular modules of training during team meetings and considered that part of their regular work without even considering them part of a training program.
- Two of the organizations doing mission critical software considered it important to have a certification program for flight software personnel. The flight software unique aspects of doing mission critical or safety critical software and the flight software lessons learned were emphasized in these programs.

5.5.4. Recommendations

TR1 Continue to develop and enhance the NASA OCE Software Engineering Curriculum classes and provide them on a regular basis. These types of classes form a good training basis and greatly enhance the software training program at most NASA Centers.

TR2 Develop some of the software classes for the more experienced software developers as on-line training, videos, or small separate modules of training that can be offered as needed throughout a project. Particular skills can be learned or reinforced at the point where they can immediately be put to use on the project, with the fresh knowledge in mind; project members are more likely to apply consistent approaches for project activities in those skill areas.

TR3 Include training on desired set of expected practices in all classes, particularly process classes. Then process isn't thought of as "something extra", but it becomes "the way the organization does business."

TR4 Develop some guidelines for a more structured approach to some of the non-classroom training opportunities, such as mentoring, peer reviews, lessons learned sessions and other on-the-job training opportunities.

5.6. Metrics

5.6.1. Questions

- Could you share some of your metrics programs currently in place?

The question above was the only one consistently asked regarding metrics and it was asked primarily for the CMMI high maturity organizations. In many cases, the topic of metrics came up in the discussions, so information was also collected from most of the other organizations interviewed, including those who were not organizations with a high level of CMMI maturity.

5.6.2. Discussion

As noted in the Questions section above, the information on metrics was not collected consistently across all of the organizations, and in some cases, follow-up calls were made to get additional information. Since this area was not a primary focus area of the benchmarking, the information in this section is incomplete for some organizations, but

overall, the interviews yielded a great deal of information on the metrics practices of the various organizations. In several cases, organizations commented that a strong metrics collection and analysis program is a key factor for successful software projects and references were made to “managing projects by metrics” as an ideal state.

The table below is intended to show common trends in the types of metrics being collected and in the practices performed.

Table 5: Commonly Collected Metrics and Metrics Practices

	Organizations																	
	N1	N2	N3	N4	N5	I1	I2	I3	I4	I5	G1	G2	G3	G4	U1	U2	U3	U4
Metric Collected																		
KSLOC	X	X	X	X	X	X	X	X	X		X	X	X	X		X	X	X
# Requirements/size	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
Defects	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
Defect density				X	X	X	X	X	X				X	X				X
Planned vs. actual size				X	X	X	X	X	X			X	X	X				X
Peer review data	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X
Peer review coverage						X		X								X		
Phase containment					X	X	X	X	X				X					
% reuse				X	X	X		X	X				X					
% new				X	X	X		X	X				X					
Planned vs. actual effort		X	X	X	X	X	X	X	X			X	X	X				
Cost data				X		X	X	X	X	X		X	X	X			X	X
Effort /phase				X		X		X	X			X	X					
Effort/activity or process					X	X		X		X		X	X	X				
Tests performed/passed	X	X	X	X	X	X	X	X	X		X	X	X	X		X		X
Requirements verified	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Schedule variance	X	X	X	X	X	X	X	X	X	X		X	X	X				
Cycle time						X			X	X		X						
# change reports		X	X		X	X	X	X		X								
Requirements volatility	X	X	X	X	X	X		X	X				X	X				
Productivity				X	X	X	X	X	X				X					
QA audit findings	X	X	X	X	X	X		X	X		X		X	X				X
Development progress	X	X	X	X	X	X	X	X	X		X	X	X	X		X		X
Resource utilization		X	X	X	X	X	X		X									
Non-compliance delinquency				X		X							X	X				
Test coverage (% code tested)										X								
Metrics Advanced Practices																		
Historical database				X	X	X	X	X	X			X	X					X
Organizational set of measures				X		X	X	X	X			X	X					
Process performance baselines				X		X	X	X	X			X	X					
Prediction models				X					X			X	X					
Collection tools		X	X	X					X	X		X	X					
CMMI level - at interview	2	2	2	3	3	3	5	3	3	-	#	2	5	3	-	-	-	3
Previous CMMI level 5 org.						X			X			X						

Nx	NASA Centers	Ux	University, Univ. R&D Labs
Ix	Industry Organizations	Gx	Defense Services

Note: X's in matrix indicate practices or metrics mentioned in interviews. Lack of X's in boxes DO NOT indicate metric is not collected or practice is not performed. The chart should be viewed merely as an indication of the practices mentioned often and the metrics that seemed to be commonly collected. For U1, no information was received. # indicates organization had a previous CMMI Maturity Level 2 rating, which was expired at time of interview.

5.6.2.1. Aerospace Industries

As a group, the aerospace industry organizations seemed to have the strongest metrics programs, with most organizations focusing on a multi-level metrics approach. The top level metrics program consisted of having some senior management, high level or cross-project measurement objectives and a corresponding set of metrics, collected by all of the projects in the organization. This set of metrics usually consisted of six to 20 items and focused on items of interest to the business interests of the organization such as: effort and cost data, on-time delivery data, cycle time, productivity, defects/KSLOC. These top-level metrics were typically used to improve cost estimation and to assist with the delivery of on-time, high-quality software, as well as to improve cycle time and productivity. One organization mentioned that their 14 organizational metrics were collected consistently across 12 of their sites.

Four of the five aerospace industry organizations had historical databases of cost data and mentioned having other process performance baselines, such as the expected number of defects/KSLOC at key points in the development cycle. One organization stated that they were able to always get within 10% of their planning parameters, doing their cost estimation based on their historical database. One organization mentioned having a home-grown tool and some common spreadsheets to help collect the metrics required at the organizational level, as well as a specific class on analyzing the metrics data collected.

A lower level tier of metrics collection was typically done at the project level and focused on the types of items that enabled a project manager to measure the health of his project. Typical items collected and analyzed at the project level included: software planned versus actual size (KSLOC), percent of reuse versus new KSLOC, requirements volatility, phase containment data for defects, defects open/closed, test completion/requirements verification data, peer review data, development progress or earned value data, QA audit findings, delinquency in audit closures (and in error closures), and resource utilization. Project level measures were typically reviewed on a monthly basis to identify any potential issues that needed attention.

For one aerospace industry organization, there was not as much metrics information available, but they mentioned that their projects relied heavily on peer reviews, and that they tracked such items as code coverage during testing, and discrepancies and change requests, along with corresponding severity levels.

5.6.2.2. Universities and University Research and Development Labs

There was very little commonality among the university organizations interviewed concerning metrics practices. Measurement was not a focused topic of discussion with any of the university organizations, so the data presented here is partial at best.

There seemed to be more emphasis on metrics collection and analysis at the two organizations that were actually separate laboratories affiliated with universities, but not in the main university setting. In one of the university laboratories, the organization had been collecting metrics for over ten years and had a large amount of data including defect data, costing data, effort data on activities and audit information. However, they indicated that they had not really done much analysis on the data and felt that it would benefit them to spend more time analyzing what they had collected. Not much metrics data was received from the other university laboratory organization, but they indicated that they collected error data, peer review data and data on requirements verified. They commented that they have metrics, but felt they could be improved.

One of the university organizations indicated that they tracked defects and actual cost of the project versus the proposal cost. No other information was discussed. The other university organization indicated that the projects were done strictly in a classroom exercise type of setting with an instructor as a mentor. Each project chose their own metrics to collect and they varied greatly by project.

5.6.2.3. Defense Services

Among the defense services organizations, there was less commonality among the four organizations interviewed than there was among the aerospace industry organizations. One defense services organization stood out as the best example of an organization with a strong metrics program. They also used a tiered metrics program approach similar to that of the aerospace industry organizations and they showed us several examples from their project historical performance database. This organization had developed many baseline models and was able to use them to predict the expected performance of their projects in many areas. The measures they mentioned collecting were similar to the ones mentioned for the aerospace industry organizations, with the addition of process adherence, customer satisfaction and defect removal rate. They used an on-line measurement collection system to ensure collection of a consistent set of organizational measures.

The other three defense services organizations seemed to collect fewer measures but all mentioned collecting peer review metrics, defect data, and audit results. Two of the organizations mentioned software volatility, latency in resolution of errors, cost and effort data, and progress tracking measures such as tests run/completed, code review

metrics. One defense services organization was just beginning to do some collection of metrics at an organizational level, another had been collecting them for about three years and the third had a fairly extensive historical database, including some process performance models. One of the organizations mentioned using a web-based metrics collection tool for consistency and another used a process dashboard that collected time spent on activities. One of the organizations described a set of their projects that were using the Software Engineering Institute's (SEI) Team Software Process (TSP) and talked about the detailed set of metrics they keep with TSP. Their metrics included the amount of time each programmer spent on each activity, defect data, cost history, source lines of code (SLOC)/hour, average time/cost to close defects, and rework. A special tool called TIMER helped the programmers track their activity times.

Within the defense services organizations, two mentioned metrics tailoring for small projects. The organization whose metrics stood out the most used a different set of metrics for the small projects, where the set was smaller and more suitable for the smaller teams. One of the other organizations had the small projects record their metrics in a log as they went along so there was less overhead involved in collection.

5.6.2.4. NASA

The NASA Centers interviewed are at varying levels of maturity in their metrics programs. Generally they compare well with the other defense services organizations. No NASA Center has a metrics program as extensive as the defense services example mentioned earlier, but one NASA Center has a maturing set of organizational measures, along with a fairly extensive historical database and some initial process baseline models. The projects at that Center are collecting metrics similar to those mentioned by the aerospace industry organizations and in compliance with *NPR 7150.2*. Another one of the NASA Centers has a consistent set of metrics that are collected across their flight software branch and a historical database for cost estimation. Using this set of metrics, they have been able to do some analysis across projects and see some organizational trends. Their projects also collect *NPR 7150.2* compliant metrics. The other three NASA Centers collect and analyze metrics mostly on a project by project basis, with the set of metrics collected varying by project. Two of the Centers have a specified set of metrics for the projects to collect and have spreadsheets that assist their projects with this collection. In general, project metrics are presented at monthly reviews and discussed to identify any needed corrections.

Typical metrics for NASA projects to collect are: requirements volatility, progress data (planned milestones versus actual, units of code designed, completed, tested, etc., functionality delivered versus planned, defects open/closed, defect severity, software

size, resource utilization, peer review data). Several Centers collect defect containment data.

5.6.2.5. CMMI and Metrics

The aerospace industry organizations and the defense services organizations that were CMMI Maturity Level 5 or had been CMMI Maturity Level 5, as well as some of the more mature CMMI Maturity Level 3 organizations stood out in terms of the quality of their organizational measurement programs. In general those organizations had historical databases that included metrics data from previous projects, including information such as cost data, effort data, cycle time, productivity, defect data, time to close defects reports, etc. This data enabled these organizations to get more realistic cost estimates as well as a better picture of the quality and performance characteristics of their projects across the organization. They also had sufficient data to enable a determination of the areas where process improvement would yield the most benefit.

5.6.3. Observations

- A good metrics program provides key critical objective information needed to adequately manage projects. The majority of the organizations interviewed, with the exception of the university organizations, had strong metrics collection and analysis programs within their projects.
- The organizations with CMMI higher maturity levels had better measurement collection and analysis programs at the organizational level that included historical databases for cost estimation, and data for establishing project performance trends. This information allows the projects to compare their project's performance with the typical performance of similar types of projects in order to highlight potential problem areas and to predict the project's future performance, based on the current project data and typical patterns of previous projects.
- The measurement and analysis programs in place at the higher CMMI level organizations provided them with the ability to measure trends in their organizational performance and identify areas where process improvement activities would provide the most benefit.
- Several organizations mentioned that a strong metrics collection and analysis program at an organizational level is a key enabler for achieving higher CMMI maturity levels.
- Tool support is helpful in collecting measures, particularly in collecting a consistent, organizational set. Many organizations had developed specific tools

to collect the metrics they wanted or had tailored their development tools to capture the desired metrics as a part of performing the development work.

- Some metrics, such as percent of code tested (test coverage) and peer review coverage, were not collected by the majority of the organizations interviewed. This may be a topic for follow-up investigation.

5.6.4. Recommendations

ME1 Continue to improve measurement activities at the Centers, at both the Center organizational level and at the project level. This will support better management of software throughout the life cycle and provide the organizational information on current software capabilities and potential improvement opportunities. Provide training to the projects on analyzing metrics data. Ensure that key metrics are a part of project reviews.

ME2 Establish a set of consistent software metrics at the Agency level that extend the current inventory metrics so key trends can be identified and models can be established:

- Determine what the real objectives for measurement are.
- Identify and collect a few key metrics that can be collected consistently across the Agency to help answer questions such as: Are software costs increasing or decreasing? Is productivity increasing or decreasing? Is defect containment improving? Is NASA software cost estimation improving? Are NASA defect rates increasing or decreasing? How many defects/KSLOC should be expected in each phase of testing? How accurate is NASA cost estimate at initial concept? At project approval? At SRR? At PDR? At CDR?

ME3 Investigate the use of tools to help collect a more consistent set of organizational measures. A consistent set of tools and basic metrics will allow the development of project performance baselines and cost baselines.

ME4 Provide organizational measurement feedback to projects so they understand the benefits of an organizational metrics program and use this information more effectively to benefit their projects and to add value to their project reviews.

5.7. CMMI

5.7.1. Questions

- Who decided and how did your organization decide to ascertain your current CMMI maturity level?
- How were impacts to policies, requirements, training, and other organizational structure handled?
- Have you been able to measure or identify any benefits at previous and current maturity levels?
- What are your top three areas of improvement and what impact did they have?
- How have you overcome major challenges that you have faced with pursuing maturity levels?
- How have you been able to reduce the cost of appraisals?
- Does your CMMI include or exclude small projects?
- (For Maturity Levels 4 and 5): Could you share some of your measurement program currently in place?
- Does it cost less to operate at a higher maturity or are other benefits more significant?

5.7.2. Discussion

All but one of the organizations benchmarked had at least considered CMMI and most had a current or previous CMMI rating. Of those who did not have a rating history, the organizations had compared their practices against CMMI and were familiar enough with it so that the CMMI best practices made a good common basis for discussions on practices. Several of the organizations benchmarked have achieved CMMI ratings since they were interviewed and several organizations interviewed had previous CMMI Level 5 ratings which had either expired or had been renewed at a lower CMMI maturity level. In Figure 3 below, the blue bars show the ratings of the organizations when the interviews were done, and the red bars indicate where organizations held different ratings previously.

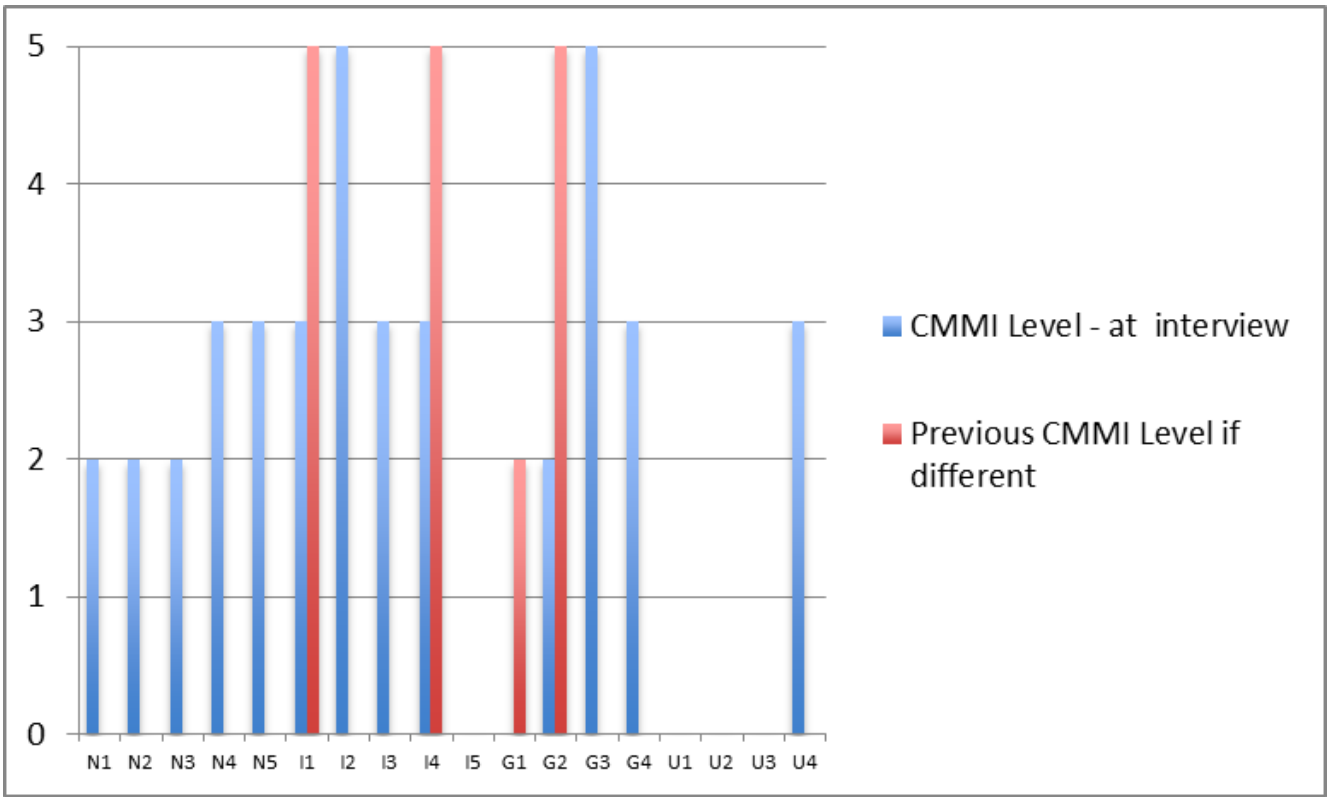


Figure 3: CMMI Levels of Organizations Pre-Interview and At Interview

The majority of the organizations had initially decided to work towards CMMI ratings based on either agency or DoD requirements/mandates or because their senior management felt that the CMMI ratings were a good business discriminator. One organization began to work on improving their practices and achieving a CMMI rating following difficulties with some of their projects. In the organizations with CMMI ratings, there was a good level of support by the senior management.

Within the aerospace industry organizations, three of the four with CMMI ratings achieved the CMMI rating in order to have a business advantage over their competitors. The fourth industry organization achieved the CMMI rating because it was required to do DoD work. The fifth industry organization did not have a CMMI rating and said they had not investigated CMMI enough to determine whether it would provide sufficient benefit for their organization. However, they did state that the CMMI knowledge they had provided them with a common language when dealing with its customers.

Table 6 below lists benefits and challenges that were mentioned by a number of organizations.

Table 6: CMMI Benefits and Challenges Mentioned in Interviews

	Organizations																	
	N1	N2	N3	N4	N5	I1	I2	I3	I4	I5	G1	G2	G3	G4	U1	U2	U3	U4
CMMI Benefits																		
Projects do better planning		X	X			X												
Better cost estimation (more credibility)		X	X		X							X	X					
Quicker start-up of projects (less effort with tools and templates)			X			X								X				
Better progress tracking			X			X							X					
Errors found earlier (peer reviews and inspections)		X			X	X	X		X		X	X						X
Better cost and effort predictability								X				X	X	X				X
More personnel flexibility with common processes						X								X				
Systematically done testing							X											
Causal Analysis Resolution (CAR) helps prevent future errors							X		X									
More standardization/consistency of processes across projects							X		X									
Standard Quality Assurance (QA) checklists/better QA			X	X	X	X	X											X
Helps optimize cost savings							X		X			X	X					
Better project performance/lower error rates						X			X			X	X	X				
Process ingrained in organization						X	X											
Many common templates and tools across projects			X	X	X	X	X	X			X							
Less rework, improved cycle time												X	X					
Business advantage						X	X						X					X
More control over requirements changes		X		X								X						X
CMMI Challenges																		
Having projects in right stage to appraise		X							X									
Getting good management support/funding	X		X			X					X		X	X				
Having adequate funding to make progress		X							X		X							
Need buy-in from projects		X	X		X	X						X	X					
Getting projects to follow consistent or common processes			X			X					X							
Need to evaluate cost/benefit of correcting every weakness to avoid over-achieving for rating				X			X											
Subjective interpretation of CMMI practices across Lead Appraisers			X		X		X						X					
change)						X							X	X				
Changes in CMMI model									X			X	X					
CMMI on small projects	X															X		
Collection of good quantitative measures (in lower maturity organizations)			X	X														
CMMI Levels																		
CMMI level - at interview	2	2	2	3	3	3	5	3	3	-	#	2	5	3	-	-	-	3
Previous CMMI level 5 org.						X			X			X						

Nx	NASA Centers	Ux	University, Univ. R&D Labs
Ix	Industry Organizations	Gx	Defense Services
	- Not Applicable		

Note: X's in matrix indicate practices mentioned in interviews. Lack of X's in boxes DO NOT indicate the lack of the characteristic. The chart should be viewed merely as an indication of the characteristics mentioned. For N1, benefits of CMMI were not discussed during the interviews. # indicates organization had a previous CMMI Maturity Level 2 rating, which was expired at time of interview.

5.7.2.1. Aerospace Industries

Of the four aerospace industry organizations with CMMI ratings, three were either CMMI Maturity Level 5 or had previously been CMMI Maturity Level 5. In addition to Table 6, comments on CMMI benefits include:

- **Well-disciplined project teams:** Processes are engrained in the project teams so that they are followed as part of their everyday work. One organization commented that the teams strived "to be perfect." Process training was part of the regular training programs and training was provided periodically for many roles, including roles such as configuration control board (CCB) members, and managers.
- **Better planning, quicker start-up:** Standard templates and tools enabled the projects to start-up more quickly and provided more flexibility for project team members to be able to move from one assignment to another with less learning curve time necessary on the new project.
- **Systematic testing and peer reviews:** One of the organizations felt some of their largest quality increases from CMMI practices were due to their process of doing inspections and re-inspections using checklists in conjunction with a systematic testing approach. After a round of testing had identified and removed some errors, the team went back and did a re-inspection before resuming testing. When errors were found, the team checked to see if the error was in any generic parts of the code or if it might appear in other similar places in the code.
- **Errors found earlier:** One organization stated that they had metrics on the number of errors found in different phases and numbers of phase escapes. They were finding fewer phase escapes and were identifying more errors in earlier phases of the projects after following CMMI practices.
- **Cost savings:** One organization estimated the cost savings with CMMI to be about 15%.
- **Management by metrics:** One organization felt CMMI's emphasis on useful measures contributed significantly to good project management since the collected measures provided the project managers with the information to qualitatively evaluate their status and progress. The measures collected at the organizational level provided the necessary information to determine what areas could benefit the most by process improvements.
- **Agile:** One organization used Agile/SCRUM for some of its projects but said that their processes were "overweight" for a typical Agile project. Their projects had made adjustments in the typical Agile methodology to ensure that they could meet even the CMMI Level 5 practices. One of the areas necessary to expand

was the emphasis on metrics. They felt this approach contributed to the quality of their software.

- **Common processes:** Four of the five industry organizations had some level of common processes across their organizations or business units. One of the organizations listed standard processes and standard software assurance checklists as one of their top areas of improvement. Another organization used common processes over multiple sites. They also had over 100 common tools and many common templates. Another organization stated that they have consolidated processes across all their business units. They had retired many of their local processes in favor of the organizational ones. Where they had specific local processes that were considered of benefit to the organization, they were allowed to keep the local processes. The organizations felt that having these common practices had many benefits including more consistency across projects, easier start-up, more flexibility of staffing, and lower costs to maintain processes.

All of the industry organizations with CMMI ratings stressed that strong support from higher-level management was a key factor in achieving CMMI maturity levels. Funding for the infrastructure and process maintenance is also necessary for continued CMMI compliance. One of the organizations that had decided to renew its rating at a CMMI Maturity Level 3 instead of CMMI Maturity Level 5 commented that they needed additional funding for someone to help with the analysis of their metrics in order to achieve CMMI Maturity Level 5. At CMMI Maturity Levels 4 and 5, the measures across projects need to be analyzed in order to determine process performance and to determine those areas where process improvement would provide the most benefit. This was generally a function that had been supported by the organization's SEPG, but without adequate funding they had been unable to continue it. The organization commented that they missed the additional insight into the performance of their projects and organizational processes that had been provided with the level of metrics emphasis they had while they were rated CMMI Maturity Level 5.

Another industry organization commented that they might not try to renew their current CMMI Maturity Level 5 rating because they didn't think maintaining Level 5 was cost effective. They thought a majority of the benefit had been achieved by obtaining the rating initially and that they could maintain the benefits achieved without renewing the rating at the same level. They felt the processes had been ingrained into the organization to such a degree that they would continue without the burden of the appraisal at Level 5.

5.7.2.2. Universities and University Research and Development Labs

CMMI was much less of a factor in the software development areas of the universities and university laboratories. One of the university organizations had not heard of CMMI. The other university, where projects were done in the context of a class, gave the students a class in CMMI before the project. Thus the students were familiar with the CMMI process areas, goals, and practices, but they were allowed to choose their own set of processes and were not required to follow those in CMMI.

One of the university laboratories did not have a CMMI rating, but had compared their practices to the practices in CMMI. They felt that their practices were adequate and they considered themselves a “best in class” provider. The interviewees stressed the difficulties and expense of trying to obtain and maintain a CMMI rating for very small projects that are devoting all of their limited time to their software projects. They felt the requirements in *NPR 7150.2* were too restrictive for an organization of small projects like theirs.

The other university laboratory had a CMMI Maturity Level 3 rating that had been driven by a senior management decision to enable them to be more competitive. Before deciding to implement CMMI, this organization had started to make the types of improvements they felt would help them develop better quality software. Once they started to implement the CMMI practices, they found that they didn’t really have to change very much because the practices they were already implementing were the types of practices needed for CMMI. They had small projects and have been able to tailor their processes well for CMMI. Other than the business advantage, they did not talk about the specific benefits of CMMI for them.

5.7.2.3. Defense Services

Of the defense services organizations interviewed, one was CMMI Maturity Level 5. One had a previous CMMI Maturity Level 5 that had expired and was working toward a CMMI Maturity Level 3. One organization was a CMMI Level 2 and obtained a CMMI Maturity Level 3 shortly after the interview. The fourth organization had an expired CMMI Maturity Level 2 and was working to regain that level. All of the organizations listed a number of benefits to obtaining a CMMI level and many were very similar to those mentioned by the industry group of interviewees. Other comments received in addition to those noted in Table 6 include:

- **Better cost/quality/schedule performance:** Several of the organizations claimed their projects:
    - Produced a “high quality product in a timely fashion.”
    - Completed “within budget and on schedule.”

    - Saw “improvements in cost and cycle time.”
- **TSP and CMMI:** One organization using SEI’s TSP in conjunction with CMMI recovered the $100K cost of establishing TSP within their first 18 month cycle period due to the increased productivity they measured.
- **Better project management:** One project said that the use of CMMI had made their personnel more aware of cost and effort management and the people became more vigilant in managing their projects.
- **Errors found earlier:** One organization reported that they had a 240% improvement in the number of defects captured within phase as they moved from CMMI Maturity Level 3 to CMMI Maturity Level 5.
- **Small projects and CMMI:** One organization reported that they didn’t see a business case for moving small projects from CMMI Maturity Level 3 to CMMI Maturity Level 5. The additional support of the common templates and tools at CMMI Maturity Level 3 was worthwhile for their projects, but the additional levels of metrics collection and analysis didn’t seem to provide enough benefit for the small projects.
- **Better reporting and progress tracking:** The projects in the CMMI Level 3 or higher organizations collected a good set of metrics on their project status and the organizations provided cost models, prediction models and performance models, allowing the projects to do better assessments of their status and probable future performance. Regular management status reports reported these metrics to provide better visibility into project performance.

Several of the defense services organizations mentioned the change management aspect of implementing CMMI as a major challenge. One of the ways they addressed the change management was to employ a number of full time process consultants, sometimes including certified CMMI Lead Appraisers. These consultants could then teach classes or serve as coaches for the projects to help get the processes institutionalized. One organization thought the use of tools to support the processes was helpful and they had used Process Max for process control and document review. One of the organizations commented that the efficiencies achieved by using CMMI had allowed them to reduce their support level significantly. One of the organizations said moving from CMMI Maturity Level 2 to CMMI Maturity Level 3 was much more difficult for them than achieving the original CMMI level because CMMI Maturity Level 3 required more infrastructure.

5.7.2.4. NASA

Of the NASA Centers that were interviewed, three had CMMI Maturity Level 2 ratings for most or part of their organizations. The other two NASA Centers had Maturity Level 3 ratings for most or part of their organizations. One of the Maturity Level 3 organizations said they did not feel they had enough data collected to quantify all of the benefits yet, but they did list some subjective improvements they had noticed. Other Centers also had limited quantitative data, but listed a number of areas where they had noted improvements. Some of the benefits they mentioned include:

- Available templates and tools improved project consistency and allowed projects to start up more quickly.
- The scope of software assurance has expanded and over time the number of findings is declining. Software development and software assurance are working together better.
- Projects were doing better planning and were able to track their progress with more accuracy.
- One Center commented that they were finding errors earlier in the life cycle with their peer reviews and another Center felt that they were seeing a huge benefit from the inspections.
- Several Centers said their cost estimation had improved and that their cost estimates had achieved more credibility with the mission project managers.
- Several Centers have measured increased productivity since implementing CMMI practices, but one of the Centers observed that other factors may also have contributed to the improvement.

Some of the challenges noted by the NASA Centers were funding and resource issues to develop the tools, templates and processes and to mentor the projects on their use. Another major issue for some Centers was the availability of projects for inclusion in an appraisal. Many projects were cancelled following the change of direction away from Constellation. One Center noted that it was more of a challenge to apply some of the CMMI practices on small projects, but several of the Centers had included one or more small projects in their appraisals.

5.7.2.5. CMMI Maturity Level 5 Organizations

Across the whole set of organizations interviewed, there were certain common characteristics noted in the organizations with the higher CMMI maturity levels. These were particularly obvious in the organizations that were rated CMMI Maturity Level 5 and in those with previous Level 5 ratings. Characteristics noted include:

- **Strong metrics programs:** All of the high maturity organizations had organizational metric programs that provided much more insight into the status of their projects as well as information on their project characteristics such as cost, effort, cycle time, and productivity. These organizations had historical databases of past project costs to use in their cost estimation for future projects. They also had data such as expected number of defects across the life cycle, based on the size of the project so their projects could compare their performance with the typical number for a project similar to theirs. Other process performance models allowed the prediction of items like time and effort to complete the project, expected expenditure, etc. Because these organization measurement assets were available, the projects were really able to "manage by metrics." The projects were able to use the organizational resources to do a better analysis of the implications of the metrics they were seeing on their own projects.
- **Comprehensive training programs:** The higher maturity organizations had training programs that provided a variety of delivery types (on-line classes, classroom classes, lunch-time seminars, structured mentoring and OJT, etc.) Training was typically required periodically and was targeted for certain roles. Possibly the most important aspect seen was that, in most cases, these organizations arranged to have the training available when needed, and sometimes the training was built in as a part of the project's activities.
- **Process-oriented personnel:** The higher maturity organizations had made clear their expectations in terms of process behavior and their personnel made comments like, "Process isn't extra – It's the way we do things." One small project member commented that it was just easier to follow the standard process, even on the small projects.
- **Organizational assets and tailoring to suit their projects:** These organizations had standard sets of process, templates, checklists and tools designed for their types of projects and a method of tailoring the assets for particular projects.
- **Strong peer review culture:** All of the higher maturity level organizations relied heavily on the use of peer reviews to help find errors early in the life cycle.

5.7.3. Observations

- Across all of the organizations with CMMI experience, the top three most mentioned benefits were:
    - Errors were found earlier in projects.
    - Many common templates and tools were used across multiple projects.

    - Better QA on projects, as well as QA checklist.
- Although many of the organizations interviewed listed challenges in the implementation of CMMI maturity levels, including obtaining funding, and getting project buy-in and senior level support, the organizations also realized many benefits after achieving CMMI maturity levels.
- The benefits of achieving CMMI level seemed to increase as the CMMI maturity levels went from 2 to 3 and from 3 to 5. The organizations at CMMI Maturity Level 5 (or previously at CMMI Maturity Level 5) exhibited characteristics not typically found in the other organizations interviewed. (See previous section for characteristics).
- Several higher maturity organizations did not feel that it was cost effective to maintain their CMMI Maturity Level 5 rating.
- A number of the organizations had developed common processes, often including common tools, templates, checklists, etc., to help obtain consistency across projects, to lower overall process maintenance costs and training costs, to enable faster project start-up, and to provide more personnel flexibility.
- NASA Centers have improved in a number of areas since implementing CMMI.

5.7.4. Recommendations

CM1 Continue to require CMMI for critical NASA projects as a method of promoting high quality mission software. Also use CMMI as a standard yardstick to measure the capability of organizations who are/will be developing NASA software.

CM2 Develop/consolidate/collect common processes, principles and other assets across the Agency in order to provide more consistency in software development and acquisition practices, and to reduce the overall cost of maintaining or increasing current NASA CMMI maturity levels.

CM3 Pursue collaborations with other organizations that have strong programs in areas where NASA could benefit from additional improvement and continue to improve NASA programs in those areas. Areas where NASA could improve include:

- An organizational metrics program.
- Improved cost estimation for both in-house developments and for in-house estimates for acquired software.
- More consistency of process performance across projects.
- A more comprehensive, training program with modules available

whenever needed.
- Development and application of appropriate levels of rigor for small projects.

5.8. Small Projects

5.8.1. Questions

- Please describe the scope (size and criticality) of small projects.
- Are small projects (all criticality levels) required to comply with software policies and requirements? If so, what are some of the key ways in which small projects are able to comply? If small projects are not required to comply, how are small projects governed?
- Does your CMMI statement of work include or exclude small projects?
- Does your organization have any infrastructure to support a collection of small projects?
- What methods or tools have you found that work well for small projects?
- How does your organization satisfy good software practices with limited resources and funds allocated to small projects?

5.8.2. Discussion

For the purposes of obtaining consistent information in the interviews, small projects were defined as projects with five Full Time Equivalents (FTEs) or less. Most of the organizations used similar definitions for their small projects. Variations in the definition ranged from less than ten FTEs to less than two FTEs in a 24-month period. Of the 18 organizations interviewed, only one said they really didn't have any small projects and four of the organizations had only small projects. One large organization said three-fourths of their projects were small projects. Table 7 shows some of the characteristics of the organizations interviewed. It is interesting to note that the large majority of the organizations with both large and small projects followed essentially the same set of processes for both the large and small projects.

Table 7: Organizational Practices for Small Projects

	Organizations															
	N1	N2	N3	N4	N5	I1	I2	I3	I4	I5	G1	G2	G3	G4	U1	U2	U3	U4
Organizational Characteristics																		
Had no small projects					X			X										
Had only small projects										X					X	X	X	
Excluded small projects from CMMI							X							X				
Included small projects in CMMI appraisals		X	X	X		X			X			X						X
Don't know if small projects were included in appraisals	X										X		X					
Practices for Small Projects																		
Used same processes (tailored) for small projects as large projects		X	X	X		X	X		X		X	X	X					
Pre-tailored by organization							X											
Under large project umbrella							X											
Tailoring help from Technical Authority	X	X		X	X				X		X							X
People indoctrinated in process-"They just follow it"						X							X					
Used umbrella SMP						X												
Tool support (availability, set-up, sys.admin.)	X		X	X	X	X			X	X	X							
Outside support for testing, peer reviews						X			X		X			X		X		
Extra documentation support		X	X			X												
Close relationship with hardware team			X							X								X
Heavy reliance on peer reviews										X			X			X		
Process tailoring workbook				X							X			X				
Special small project templates													X	X				
Special small project tools			X						X	X	X						X	
Use of wiki for communication	X									X					X		X	
Less formal reviews, reduced scope or combining of documentation	X	X	X	X		X				X		X	X	X	X			X
Single manager for multiple projects												X						
Use Of TSP/PSP												X						

Nx	NASA Centers	Ux	University, Univ. R&D Labs
Ix	Industry Organizations	Gx	Defense Services

	- Not Applicable		

Note: For U1, each project chooses its own processes
For N5, very few small projects in organization

5.8.2.1. Aerospace Industries

Of the five aerospace industry organizations interviewed, one had no small projects and one had only small projects. Two aerospace industry organizations reported that the majority of their small projects were research projects such as pilot projects or independent research and development (IRAD), with only a few actually developing the more critical software. The small projects that were not research (defined as less than 1000 hours or one to two people) were still required to follow the same processes, but their processes were pre-tailored by the organization to reduce the overhead. In one case, the smaller projects were put under the umbrella of larger projects to reduce their

paperwork. Then the larger project would include information for the small project in most of its documentation, for example, in the Software Management Plan (SMP). For the other industry group, an umbrella SMP was developed that covered several of the smaller projects to reduce the paperwork overhead.

One of the aerospace industry organizations with small projects had several approaches in place that seemed to work well for their small projects:

- Although the small projects used the same basic processes as the large projects, the Technical Authority (at the level of a NASA Branch Head) worked closely with them to help tailor the process. It was the responsibility of the Technical Authority to make sure that the tailored processes met all the required key points and any project risks were reviewed with the Technical Authority regularly.
- The organization provided tool support for the small projects including not only the purchase of the tools, but also the initial set-up and tool administration throughout the project. Generally one tool guru supported several small projects.
- Small projects still had an independent tester for formal tests and they performed peer reviews. Special arrangements were made to have an external person assigned to support these activities.
- Software assurance was still involved in these small projects doing process audits, review and signature of documents, and after-the-fact review of peer reviews.
- Process training seemed to be a key for the small projects. The personnel had been well-trained and indoctrinated into the culture of the process used for the large projects and they commented, “It’s just the way we do business.”

The aerospace industry organization with all small projects also had some practices that were very supportive of the small project environment. Some of their characteristics are:

- Personnel for small teams were carefully chosen to ensure that team members had the best possible skills for the project. New team members were carefully mentored and monitored before allowing them to submit code.
- The small project software team worked closely with the hardware team. Software and hardware requirements were refined iteratively.
- Team members reviewed each others’ code and the whole team relied heavily on team peer reviews. This organization felt that the constant communication and shared familiarity with the project details enabled the team to keep documentation and formal reviews at a minimum.
- The teams used tools to assist them in many areas of the project. They mentioned tools to track and plan their work, as well as tools for automated unit

testing, program documentation, static analysis, configuration management, issue tracking, peer reviews and code coverage.

5.8.2.2. Universities and University Research and Development Labs

The group of university and university laboratory organizations had mostly small projects, often with only one or two people on a project. One organization commented that small teams can often help them do projects quickly without losing quality. One university laboratory described a set of projects done by one person, who participated in their CMMI appraisal. Several key characteristics were noted:

- The software person worked very closely with the hardware team and requirements were developed iteratively.
- The software person documented very carefully so someone else could take over, if necessary.
- The software requirements specification was started early in the life cycle and became the primary multi-purpose document as the project developed.

Several of the university organizations commented that NASA processes, reviews, and CMMI put too much burden on the small projects. Two of the university organizations mentioned using different levels of rigor on their small projects, depending on the types of projects. One of the university laboratories commented that they used inspections heavily and that they sometimes used developers as testers, but not for their own code. Other developers were often brought in for reviews and operations people often did the independent testing.

5.8.2.3. Defense Services

The defense services organizations with small projects all essentially expected their small projects to follow the same processes as the large projects with some tailoring. One organization said they did not require their smaller projects to participate in the CMMI activities and another organization said they expected the small projects to operate at CMMI Maturity Level 3 instead of CMMI Maturity Level 5. Several of the organizations described the tailoring for small projects as a matter of scope (“Doing the same thing, but in a simpler manner”) or a matter of degree (where an SMP may have the same number of sections, but they are “lighter-weight”.)

Two of the defense services organizations mentioned having a process tailoring workbook or customization workbook that listed what could be tailored and described how much it could be tailored. Typically, the project then documented exactly what they

planned to use in their tailoring and this project tailoring needed approval or a waiver. The tailoring workbook might say that bugs needed to be tracked and the project might specify that they planned to use Bugzilla. Types of tailoring mentioned included less documents or documentation with less detail; combined reviews or less formal reviews; little or no regular software assurance; less independence of testing.

One defense services organization assigned one technical manager to manage multiple small projects. Common support for the small projects was also provided, such as a common set of tools, a common configuration management group or in the case of another organization, a common test group. It was noted that it is more difficult for one manager to manage multiple small projects than one larger one.

One defense services organization used Team Software Process/Personal Software Process (TSP/PSP) on a number of their small projects. It was noted that this is a very structured way of working with small projects and they felt they got excellent results with the process. They did comment that TSP/PSP probably wouldn't work well for everyone since it is quite metrics intensive and requires a fair amount of personal diligence in record-keeping.

5.8.2.4. NASA

The NASA Centers interviewed expressed some of the same issues with small projects as many other organizations interviewed. Their projects felt challenged to maintain the same level of rigor as the large projects when they were very resource-constrained. All of the NASA Centers interviewed except one had small projects with mission critical software and these projects were required to follow the same processes as the large projects. Three of the Centers had small projects that shared tools, or were able to get tools from the organization or a tool service. One Center had developed a number of tools that were designed for use by the smaller projects. In all these cases, the availability of appropriate tools for the small projects was considered beneficial in helping them maintain a greater level of rigor with less overhead.

Two of the Centers allowed the smaller projects to tailor their processes and provided some guidance for tailoring in their process documents. One Center wanted to write a tailoring guide, particularly for small projects, but had not yet done so. The other Center also commented that more tailoring guidance would benefit their small projects.

A third Center used an approach similar to one of the industry organizations interviewed. Their Branch Head, who was also the Technical Authority, worked with the small projects in the branch to help them develop a reasonable tailoring of their processes. Consolidation of documentation was one example of the tailoring cited. The projects at one Center were given some additional documentation support to help alleviate their resource problems.

5.8.3. Observations

- The issue of maintaining rigorous processes in small projects with limited resources is not a NASA-unique issue. The majority of the organizations interviewed had small projects that followed processes similar to those of large projects, but usually were tailored and often the small projects received additional support from their organizations.
- When tools appropriate for small projects are made available, it is easier for the projects to follow a rigorous process with less overhead.
- Organizations often give their small projects other forms of support such as providing common services like configuration management, additional documentation support, a common test team, or an extra resource for peer reviews.
- Tailoring of processes for small projects is key and can be accomplished in several ways:
    - With the use of assets pre-tailored for the small projects.
    - Through the use of process tailoring workbooks or a customization workbook.
    - Through the use of a Technical Authority close to the small projects, who can help develop a tailoring plan that works well for the project.

5.8.4. Recommendations

SM1 Develop an Agency-level set of recommendations on tailoring for small projects or develop a process tailoring workbook that lists those items that can be tailored and describes allowable tailoring.

SM2 Provide tool support for small projects. Small projects should be able to get access to the tools they need for rigorous processes, as well as support for their use and administration. Most small projects do not have the budget to purchase many of the tools that would benefit them and they do not have the expertise or manpower to set up the tools for their projects and perform the needed administrative activities.

SM3 Focus on educating technical authorities or their designees and advisors who may be managers closer to the small projects so that they can assist the small projects in developing a good set of tailored processes for their project.

5.9. Tools

5.9.1. Questions

- No specific questions were asked regarding this topic.

5.9.2. Discussion

A listing of the software tools discussed during the NASA Software Engineering Benchmark meetings is included in Appendix F. This table should not be considered all-inclusive, nor is it an endorsement of any particular tool by NASA. The purpose of this table is to provide a list of the tools reported to be used by participants in the software engineering benchmark study. General noteworthy software tool lessons from interviewees during the software engineering benchmark include:

Availability

- Having software tools available for small projects is very important.
    - Most organizations have been successful in having centralized tools for small teams to use since small projects lack the resources to acquire tools.
    - Small projects also benefit from institutionally-provided tool set-up and administration.
    - On-line tools are very helpful for small projects.
- It is important to coordinate with information technology (IT) organizations to create locations on servers that can be used to host software tools.
- Industry organizations provide more institutional software tool support.
- Some organizations are pushing towards standard tools for all programs.
- When multiple tools are available that can be used, and options are available, do a risk analysis to make a choice and to determine whether multiple tools are needed.

Analysis and Testing Tools

- Industry is looking at tools that will help with continuous integration and automated testing. For example, verifying the 1553 communications at the bit level is time consuming.

- NASA IV&V has been very helpful in running static analysis tools and sorting through all of the false positives.
- Most organizations use two compilers with all warnings turned on.
- Most organizations have developed some of their own specialized tools for performing testing and regression testing.
    - Most organizations don't use many automated test tools.
- Home grown tools are used for software testing… as opposed to automated commercial tools.

COTS and Open Source Tools

- Some organizations use a decision analysis resolution (DAR) process to determine whether to use COTS tools.
- In-house tools may be considered better than having COTS tools, since having someone local who knows the code and the tool capabilities is important.
- Defects in COTS tools have the potential to ripple into the software being developed.
- Need to have someone in-house who is very familiar with the tool, so use is not limited by reliance on the vendor.
- Vendors do go out of business; have access to the source code in case a vendor goes out of business.
- Some organizations test software tool kits, library software, and open source software at same level as in-house developed software.
- Increased use of open source software tools was seen, particularly by small projects.

Training and Additional Comments

- For a new project and or new tools, provide structured training on tool usage.
- Starting to see an increase in use of integrated software tools performing multiple functions.
- Organizations are interested in using a NASA "tool shed" concept and would like to be given access to the more expensive static analysis tools.

5.9.3. Observations

The following observations arose from the benchmark visits on the topic of software tools:

- An increased use of open source software tools was noted however, testing requirements, approach and guidelines were not clearly defined.
- Static analysis tools are commonly used across the software engineering community.
- Common software tool repositories are considered a benefit when used on most projects.
- Larger and more mature organizations are pushing towards standard software tools for all programs with a standard schema for the tools, and for most tools to be organizationally supported.

5.9.4. Recommendations

Recommendations for NASA to improve existing software tools include:

TO1 Assess the option of providing Agency-wide licenses for high-cost commonly used software test tools and static analysis tools.

TO2 Assess the need for a NASA policy on the use of Open Source software tools.

5.10. Programmable Logic Devices

5.10.1. Questions

- No specific questions were asked regarding this topic.

5.10.2. Discussion

Programmable Logic Devices (PLDs), also known as complex electronics (CE), policies and requirements were not a primary focus on the software engineering Benchmark activities, but the subject did get discussed in some of the Benchmark visits. Enough consistent data was not gathered to have any findings or recommendations in this area. In the few organizations that did talk about PLDs, the approaches were mixed on how the devices are handled from a policy, requirements, and process perspective. Based on the limited data received, it was observed that PLDs were developed by a number of different organizational approaches. Some were developed in hardware electronics groups, some were developed in software organizations and some used a hybrid approach with both software and hardware organizations participating in the development processes.

NASA usage of the term PLDs encompasses programmable and designable complex integrated circuits. They can be programmed by the user and range from simple chips to complex devices capable of being programmed “on the fly.” “Designable” logic devices are integrated circuits that can be designed but not programmed by the user. The design is submitted to a manufacturer for implementation in the device. Some of the primary types of programmable devices used are:

- Field programmable gate array (FPGA).
- Complex programmable logic device (CPLD).
- Application-specific integrated circuit (ASIC).
- System-on-chip (SoC).

A PLD is an electronic component used to build digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture and is defined prior to use. An FPGA is one of the most commonly used PLDs in space flight applications. The FPGA design is captured using a hardware description language (HDL) which defines how the chip will work, equivalent to a circuit schematic diagram. A fuse or bit file is created using the HDL design and used to configure the FPGA to perform its intended functions. Typically, modern FPGAs that have been qualified for space flight contain thousands of logic gates and memory cells, and they can perform highly complex digital logic functions. At a systems-engineering level, there are many similarities between a PLD integrated circuit development process and a software product development process; however there are also many critical differences between the two products. A recent NASA Engineering and Safety Center (NESC) assessment found there is a great deal of effort being made by the PLD engineering community and safety community to ensure robust PLD development. However, the lack of consistency in terminology and/or definitions, the various project specific or Center-specific development requirements, as well as differences between the software engineering processes and the PLD engineering processes put robust development of PLDs at risk. The issue is a combined hardware and software issue that needs to be evaluated from a systems perspective. The NASA NESC study developed a recommended approach for policy, requirements, and/or guidance that should be applied at the NASA level to ensure robust development of these types of devices used in future space flight systems. The NASA NESC assessment team recommended that a NASA-level PLD handbook be created. The NASA-level PLD Handbook is being developed using existing NASA Center PLD documentation. Similarly, a community of practice (CoP) of subject matter experts has been established to clarify and document best design practices and to improve communication and sharing between the NASA Center’s PLD experts in this dynamic technical discipline.

Currently, NASA has established a PLD CoP, under the NASA Avionics Technical Fellow to enhance informal networks between NASA Centers and other government agencies, industry, and academia to encourage communications, transfer knowledge, and share lessons learned and peer reviews. NASA has started the development of a NASA PLD Engineering Handbook through the PLD CoP. NASA has also developed a NASA Complex Electronics Handbook for Assurance Professionals. This handbook provides an overview of complex electronics, the design process, and assurance activities.

5.10.3. Recommendations

PL1 Continue with the NASA proposed plan.

6. Benefits

Among the anticipated benefits, the study also produced two unexpected results: requests to collaborate with NASA and feedback on working with NASA.

6.1. Feedback

The university and university R&D labs, and at least one industry organization provided feedback on working with NASA. The NASA SWG regularly participates in surveys to collect NASA Center feedback, but it does not have a mechanism to collect feedback from contractors or partnerships that provide software to NASA projects. This feedback is helpful information that suggests a need to understand and seek some amount of feedback to improve the overall software process while potentially improving the results/costs associated with software acquisitions. Some of this feedback is incorporated into the resulting recommendations and actions.

A few comments were collected regarding participation in the improvement of NASA's requirements and expectations. Requests were primarily:

- To be able to provide comments on the NPR's while in development so the NPR requirements are also a reasonable set for the contractors to implement.
- For NASA representatives to make sure the standard Mission Assurance Requirements and NPRs are in agreement.[7]
- To consolidate or coordinate NASA audits and surveys to minimize potential schedule impacts to contractor's schedules.

Many of the comments are extensions of the issues that NASA had identified and is expecting to gain new ways to resolve through this study. What is helpful about these comments is that they provide a more end-to-end picture of the problem and the impacts. The conversations also led to information gathered in the previous discussion sections on how these organizations work to meet the requirements and resolve issues that are problematic. For example, two organizations suggested the need for NASA to accept non-traditional documentation on small projects. There is merit in considering new ways to provide the required documentation if the rigor and necessary content is maintained.

[7] NASA Office of Safety and Mission Assurance and the Office of Chief Engineer are working together to eliminate redundancies and harmonize the updates of *NASA-STD-8719.13*, NASA Software Safety Standard and *NASA-STD-8739.8* NASA Software Assurance Standard with *NPR 7150.2* NASA Software Requirements. Approved updates of the two standards are scheduled for early 2013, while *NPR 7150.2*'s update will occur in 2014.

- NASA reviews mean taking people away from a very small development and test team and have them create and present review materials. NASA reviews typically want Microsoft PowerPoint presentations. Data is available in other (original) formats that are more detailed and would require less time to use.
- Some organizations would like to see NASA scale for smaller things; NASA Is requiring more than necessary for smaller, cheaper efforts. For example, expect a combined document instead of multiple documents to meet the requirements.

Another topic that has been an issue for NASA, and can be found on the Top Software Issues list, is cost estimation. This topic was not included in the scope of this study since cost estimation is competitively sensitive information and was not expected to be shared.

A few of the organizations had concerns about their cost estimates not being accepted or possibly not trusted at NASA. Quite frequently this also happens at NASA Centers where project representatives may pressure the software organizations to provide the software for less than requested costs. The non-NASA organizations that had these concerns all claimed to have excellent ability to estimate software costs. If this is true, it would benefit NASA to be able to estimate costs better for internal and external software development to ensure the work can be done properly and on schedule. Having better cost estimation for in-house software would help project managers with cost and schedule decisions and also help determine if in-house or external software development is the right development choice. Without strong cost estimation at NASA, these comments imply negative impact to the quality, on-time delivery, and cost of software on a project, including:

- Concerns about NASA projects challenging (and cutting) the amount of code, time, or schedule even with a measurable process estimation that has only a 10% variance.
- Concerns about requirements creep and requirements definition. NASA forces projects into a “smaller” box then forgets who made the decisions.
- Concerns that NASA starts with a budget to develop a project instead of finding out how much budget will be required for the project.

6.2. Collaboration and Continued Opportunity

The conversational aspect of the interviews allowed for comments and feedback in an open and objective manner. Some of the comments were requests to help improve internal processes or NASA processes. Several organizations asked to collaborate with

NASA, which would help improve the organization's internal processes, and more closely align their processes with NASA's. The key collaborative requests were focused on:

- Metrics
- Software training (specifically software safety)
- NASA’s tool shed
- Static analysis tools
- Software processes
- NASA's Engineering Network, which hosts NASA’s software CoP
- CMMI expertise

Several of the interviewed organizations have requested to work with NASA on specific subjects. One organization suggested a Memorandum of Understanding (MOU)/Memorandum of Agreement (MOA) to enable further sharing of information and partnership development which could include sharing of Lead Appraisers for CMMI certifications. These relationships can lead to further success for NASA’s software engineering. The interest to work with NASA or utilize NASA resources suggests that NASA can be viewed as a leader or at least a strong contributor in these fields.

Anticipated benefits were also achieved. The objectives of the effort have produced data that can be used to formulate action plans to further improve NASA’s software engineering. Enough data was collected to pursue many new avenues beyond those originally scoped for this project. With this study producing results that are incorporated into positive changes to current software engineering organization and processes, interaction with external organizations should be continued to find new ways to do business from the external environment and obtain objective viewpoints.

7. Comparisons and Trends

To help determine what information from this study could be pursued in the near term, NASA's position among the benchmarked organizations can be analyzed to find the largest differences between NASA and the external software engineering environments. On the topics studied, NASA generally fared well among the organizations that participated. The Benchmark Team rated NASA Centers as either among the strong performers or between the strongest and weakest performers. For the topics where NASA was in-between the strongest and weakest performers, the difference between NASA and the strongest organization was most noticeable for the areas of software acquisitions and metrics. Although the difference between NASA and the other organizations is not as significant in other areas, there were still many valuable ideas to consider and potentially pursue.

Overall, NASA, industry and defense services organizations were noted as more mature than the universities and university R&D laboratories. This maturity was characterized by stronger and better organized policies and processes and effective mechanisms to communicate and ensure policies and processes were followed. Training programs and the provision of tools were also noted as more organized and consistent; with the better training programs able to routinely provide training through a useful variety of mediums.

Defense services organizations were stronger in software acquisitions, metrics and testing than NASA. The Defense services organizations were better at ensuring software acquisitions comply with software engineering policies and requirements. Industry interviewees demonstrated a wide spectrum of acquisition capabilities: from levying software engineering requirements to not needing any software acquisition strategy since software was not subcontracted. (Note: It's likely that universities and university R&D laboratories were not as strong due to the limited number/lack of acquisitions they executed.) Both groups had standard contracting language to ensure compliance. Industry and some defense services organizations also had excellent cost estimation capabilities to ensure the contracts were appropriately priced. These same organizations were also better at using metrics to help manage testing and costs, giving them an edge on cost estimation and helping to establish the credibility of their cost estimates with the projects. This correlation is consistent with a higher CMMI maturity level which was found in many of the industry and some of the defense organizations. For those that had good metric and cost estimation abilities, the software representatives had respected positions within a project such that project management did not challenge them, but actually sought inputs from them to assist in determining comprehensive project costs and schedules.

NASA software assurance was mature and appeared to be comparatively strong by virtue of its well-established policies, processes, and training. No gaps were perceived between NASA's maturity and implementation of software assurance verses other strong organizations, however a few of the industry organizations appeared to perform basic software assurance functions more efficiently, using fewer people.

Although there was mixed information obtained for small projects and complex electronics, the insight gained has helped confirm forward plans already in work to address these topics.

A more substantial analysis could be performed comparing NASA to the benchmarked organizations but only a few and only general comparisons were made. The value of this analysis (comparisons) is not in how well NASA performs relative to others but in what was learned and how NASA can continue to improve.

8. Recommendations

This software benchmark effort collected a wealth of data which translates into opportunities that will result in key improvements to advance software engineering at NASA. These improvements will come through near-term execution of the resulting recommendations, summarized in section 8.2. There are many ideas and best practices that were not incorporated into this final set of recommendations but are available for development at a later time for additional actions or further study.

The data that leads to the suggested resulting recommendations, results from the topic areas that were identified as areas for needed improvements, per the top software issues. Once the data was gathered from the benchmarked organizations, both internal and external, the comparative assessment pinpointed the most noticeable differences between NASA and other organizational best practices. These differences highlight the topics which should be focused on for near-term plans. Pulling this information together leads to the specific recommendations that can be executed individually but preferably will be executed collectively to maximize the benefits of any one action since there are strong relationships and dependencies among the suite of actions presented.

These recommendations encompass a comprehensive forward plan which contributes to satisfying the NASA Software Engineering Improvement Initiative and improvement plans directed by *NPR 7150.2A,* sections 1.2.1 and 1.2.2, and fulfilling the original purpose of this study “to identify, review and employ best practices relevant to the software that supports NASA’s missions.”

8.1. Software Benchmark Study Recommendations from Topic Sections

This section contains the complete set of recommendations that were suggested for each of the topic areas. This set of topic recommendations was consolidated, eliminating duplication and grouping the actionable areas together to form the final set of actionable recommendations that appear in the Executive Summary and in Section 8.2.

		Recommendations
Policy	PO1	Improve NASA Policies and processes with regard to universities. Although this is a small sample, there is an important distinction between University versus University R&D Laboratory software providers. It can't be assumed that university graduates have a firm foundation in the awareness and use of common software engineering practices. This has implications for co-op assignments and in-house training of new hires by NASA. Recommendations include: • NASA should be proactive in filling the knowledge gap in common software engineering practices for new hires and co-ops. • Be aware of the risk of ad hoc software development practices when evaluating proposals from universities. • Work through the Science, Technology, Engineering, and Mathematics (STEM) program with universities in strengthening education in the use of common software engineering practices and standards.
	PO2	Improve the flow down of NASA's SW NPR to Center xPRs. Establish an Agency-wide 1 year time limit/grace period on flowing down approved NASA procedural requirements (NPR) updates into approved Center level direction via xPRs (i.e. NASA Center 'x' procedural requirements).
	PO3	Improve contracts. Recommendations include: • NASA should examine the details of how defense services

		organizations maintain consistency in the flow down of software requirements through contract vehicles, then create and implement standard language with respect to the Agency's software requirements. • Since in-house SQA plays a key role in compliance with policies and standards, NASA request for proposals (RFP's) should request information on supplier's QA capabilities and use it as one of the evaluation factors in contract awards.
	PO4	Establish corporate/enterprise-wide processes. The corporate-wide software engineering strategy communicated by two aerospace industries should provide a model for future NASA software engineering improvements (with appropriate tailoring to ensure it provides benefits within NASA's environment). [8]
	PO5	Collect and publish a set of well documented software engineering principles/rules from the NASA Centers, to promulgate software lessons learned in a natural periodic manner available to all NASA Centers and projects.[8]
Acquisition	AQ1	Standardize contract language for software. Develop standard NASA contracting language to ensure software requirements are consistently flowed down on contracts that include software development or maintenance. Defense services and the ESA have contract language for software that should be examined in the development of standard NASA language. Contracting language should reflect any pre-tailored requirements to enable allowances for alternative documentation and review approaches (e.g., The approach of using a standard set of compliant data requirements documents was mentioned by a couple of NASA Centers, as were PDLM tools approaches.). The contracting language should address subcontracting situations to ensure software requirements are flowed down to subcontractors from primes. This effort should leverage and enhance the software acquisition guidance provided in *NASA-HDBK-2203 Software Engineering Handbook* (Topic 7.3). While the Handbook provides needed guidance among the

[8] The NASA Headquarters Office of Chief Engineer began an initiative called Agency Processes and Principles for Software (APPS) under the leadership of Sally Godfrey and Steve Larson to develop a NASA capability in this area.

		software community, the adopted standard contract language needs to be institutionalized within the NASA acquisition community set of common practices.
	AQ2	Provide accurate and trusted software cost estimates. Improve the fidelity of NASA's cost estimates for software and utilize it to reach agreement with industry partners. NASA needs to enhance its capability in trusted software cost estimates to accurately evaluate contractor software estimates and make smart project trades. Adequate planning funds also need to be put in place by project management to ensure there are resources to do perform better software cost estimates on the defense services side.
	AQ3	Improve the NASA contracting process to adequately address software: • Introduce a check in the contracting process to ensure adequate software requirements have been included. • When contracting for systems, knowledgeable software personnel need to be involved to ensure adequate agreements are put in place. • Ensure representation or advice from software experts in the acquisition of systems that depend on software. • When applicable, use CMMI to better communicate NASA's software needs and expectations on contracts. • Secure and maintain adequate budgets to fund trusted software estimates. • Clarify what is acceptable to NASA in terms of "equivalence" and "meets or exceeds" in the area of software requirements. Leverage the work performed in 2012 under the special NASA task which mapped the Agency's software requirements to voluntary consensus standards.
	AQ4	With regard to training and guidance, recommendations include:, • Improve NASA acquisition training as it relates to supplied software (train personnel in standard software contracting language, applicable requirements, and cost estimation). • Utilize the *NASA Software Engineering Handbook* and its guidance on acquisition and contracts. Ensure the

		Handbook is available to contractors to facilitate better communication and understanding of NASA's expectation in meting software requirements. • When applicable, use CMMI to better communicate NASA's software needs and expectations on contracts. Including the expectation of increased accuracy in software cost estimates from organizations at higher maturity levels (3 and above). • Better awareness and use of the FAR supplemental clauses concerning software and data rights (at the Agency and Center levels) available to place on contracts.
	AQ5	Clarify NASA's Software Requirements. In the 2014 update of *NPR 7150.2, NASA Software Requirements*, it is recommended that inherently governmental requirements be clearly labeled to eliminate confusion when *NPR 7150.2* requirements are put on contracts.
Testing	TE1	Develop a set of predictive software defect data and a process for assessing software testing metric data against it. Use the set to status progress during NASA software testing phases and in software test reviews.
	TE2	Identify a recommended set of test metrics for NASA software development. Assess whether code coverage is a viable metric for use in software testing reviews. Better use of software metrics could highlight areas where software testing productivity and software quality could be improved.
	TE3	Review options for improving the NASA software COTS testing requirements and guidelines when the NPRs for software engineering and the associated NASA handbook are updated in FY14.
	TE4	Assess the option of buying Agency-wide licenses for commonly used software test and static analysis tools.
	TE5	Clarify the role of software assurance in NASA's software testing activities.

	TE6	Perform a workforce assessment of the software simulation skills of NASA personnel and provide recommendations based on the findings from the assessment.
Assurance	AS1	Follow up this benchmark study with a deeper look into what organizations perceive as the scope of software assurance (SA), the value they expect to obtain from it, and the shortcomings they experience in the current practice.
	AS2	Improve SA awareness at universities as well as among project managers and engineers at organizations interfacing with NASA: • Work needs to be done to better communicate the definition and scope of software assurance, with an emphasis on its value to projects, so that adequate resources can be allocated to it. • Consider using the NASA STEP program and other means to reach out to workplace training and university software engineering/computer science programs. This is an opportunity for NASA to infuse its view of software assurance, educate its suppliers, and cultivate the next generation of practitioners and users.
	AS3	Identify the tools and metrics needed to do a better job of software assurance, and work out with software engineering the best way to share the metrics that focus on project and process quality, safety and reliability while improving on defining and collecting metrics for improvements in software assurance.
	AS4	NASA RFPs should request information about the supplier's software assurance capabilities and use that information in deriving risk exposure and assessing the degree of supplier surveillance needed.
Training	TR1	Continue to develop and enhance the NASA OCE Software Engineering Curriculum classes and provide them on a regular basis. These types of classes form a good training basis and greatly enhance the software training program at most NASA Centers.

	TR2	Develop some of the software classes for the more experienced software developers as on-line training, videos, or small separate modules of training that can be offered as needed throughout a project. Particular skills can be learned or reinforced at the point where they can immediately be put to use on the project, with the fresh knowledge in mind; project members are more likely to apply consistent approaches for project activities in those skill areas.
	TR3	Include training on desired set of expected practices in all classes, particularly process classes. Then process isn't thought of as "something extra", but it becomes "the way the organization does business."
	TR4	Develop some guidelines for a more structured approach to some of the non-classroom training opportunities, such as mentoring, peer reviews, lessons learned sessions and other on-the-job training opportunities.
Metrics	ME1	Continue to improve measurement activities at the Centers, at both the Center organizational level and at the project level. This will support better management of software throughout the life cycle and provide the organizational information on current software capabilities and potential improvement opportunities. Provide training to the projects on analyzing metrics data. Ensure that key metrics are a part of project reviews.
	ME2	Establish a set of consistent software metrics at the Agency level that extend the current inventory metrics so key trends can be identified and models can be established: • Determine what the real objectives for measurement are. • Identify and collect a few key metrics that can be collected consistently across the Agency to help answer questions such as: Are software costs increasing or decreasing? Is productivity increasing or decreasing? Is defect containment improving? Is NASA software cost estimation improving? Are NASA defect rates increasing or decreasing? How many defects/KSLOC should be expected in each phase of testing? How accurate is NASA cost estimate at initial concept? At project approval? At SRR? At PDR? At CDR?

	ME3	Investigate the use of tools to help collect a more consistent set of organizational measures. A consistent set of tools and basic metrics will allow the development of project performance baselines and cost baselines.
	ME4	Provide organizational measurement feedback to projects so they understand the benefits of an organizational metrics program and use this information more effectively to benefit their projects and to add value to their project reviews.
CMMI	CM1	Continue to require CMMI for critical NASA projects as a method of promoting high quality mission software. Also use CMMI as a standard yardstick to measure the capability of organizations who are/will be developing NASA software.
	CM2	Develop/consolidate/collect common processes, principles and other assets across the Agency in order to provide more consistency in software development and acquisition practices, and to reduce the overall cost of maintaining or increasing current NASA CMMI Maturity Levels.
	CM3	Pursue collaborations with other organizations that have strong programs in areas where NASA could benefit from additional improvement and continue to improve NASA programs in those areas. Areas where NASA could improve include: • An organizational metrics program. • Improved cost estimation for both in-house developments and for in-house estimates for acquired software. • More consistency of process performance across projects. • A more comprehensive, training program with modules available whenever needed. • Development and application of appropriate levels of rigor for small projects.
Small Projects	SM1	Develop an Agency-level set of recommendations on tailoring for small projects or develop a process tailoring workbook that lists those items that can be tailored and describes allowable tailoring.
	SM2	Provide tool support for small projects. Small projects should be able to get access to the tools they need for rigorous processes,

		as well as support for their use and administration. Most small projects do not have the budget to purchase many of the tools that would benefit them and they do not have the expertise or manpower to set up the tools for their projects and perform the needed administrative activities.
	SM3	Focus on educating technical authorities or their designees and advisors who may be managers closer to the small projects so that they can assist the small projects in developing a good set of tailored processes for their project.
Tools	TO1	Assess the option of providing Agency-wide licenses for high-cost commonly used software test tools and static analysis tools.
	TO2	Assess the need for a NASA policy on the use of Open Source software tools.
PLD	PL1	Continue with NASA's proposed plan.

8.2. Recommended Forward Plan

To facilitate the development of coordinated action plans that address several aspects of a topic area, these final recommendations are consolidated and grouped into several areas discussed below.

8.2.1. Project Management

To improve software engineering as an integral part of a project and not as a stand-alone aspect of a project, the discussions in several areas led to best practices that could have substantial impacts and improvements in software development but also in project costs, schedules, and overall success.

1. Develop and implement standard contract language for software procurements (AQ1, AQ3, AQ4, AQ5, PO1B, PO3, SA4).

   By developing standardized language for procurement of software, the purchased product will more efficiently meet NASA's software engineering requirements. To successfully develop and implement standard contract language, changes to contracting process and to NASA's software requirements, communication and training will need to be addressed.

2. Advance accurate and trusted software cost estimates for both procured and in-house software and improve the capture of actual cost data to facilitate further improvements (AQ2).

   If NASA Centers are able to accurately estimate software costs, software procurement costs could be confirmed and both procured and in-house software costs could be used in project decisions and ultimately to deliver software as planned (improved baseline to negotiate requirement changes). An essential piece of accurate and trusted cost estimates is to accurately track actual costs.

3. Establish a consistent set of objectives and expectations, specifically types of metrics with examples at the Agency level so key trends and models can be identified and used to continuously improve software processes and each software development (ME2, CM3).

   With a consistent set of information, projects will be able to rely on the software engineering process to reduce risk of defects and failures and provide affordable software. This information also enhances the accuracy of software and project cost estimates.

4. Maintain CMMI Maturity Level requirement for critical NASA projects and use CMMI to measure organizations developing software for NASA (CM1).

CMMI maturity levels (ML3 and higher) displayed stronger and more mature software engineering processes and products, which is consistent with the quality and dependability of software that NASA needs.

5. Consolidate, collect and, if needed, develop common processes, principles, and other assets across the Agency in order to provide more consistency in software development and acquisition practices, and to reduce the overall cost of maintaining or increasing current NASA CMMI maturity levels (CM2, PO5).
6. Provide additional support for small projects that includes: (a) guidance for appropriate tailoring of requirements for small projects, (b) availability of suitable tools, including support for tool set-up and training, and (c) training for small project personnel, assurance personnel and technical authorities on the acceptable options for tailoring requirements and performing assurance on small projects (SM1, SM2, SM3).

   There is enough information captured through this study to provide guidance on small projects, focused on tailoring requirements and processes, training personnel on level of details/documents required and ensuring small projects have access to needed tools. These small changes will improve the consistency and success of small projects.

8.2.2. Processes, Practices, Training and Tools

Specific practices were found that could improve software engineering internally, producing better software (quality and cost) thus increasing the success of the project. These practices were focused on the process, people and tools.

7. Develop software training classes for the more experienced software engineers using on-line training, videos, or small separate modules of training that can be accommodated as needed throughout a project (TR2).

   Skills can be learned and then practiced immediately, increasing retention and proper implementation while using easily accessible training mediums to reduce time removed from a project for training.

8. Create guidelines to structure non-classroom training opportunities such as mentoring, peer reviews, lessons learned sessions and OJT (TR4).

   Non-classroom training methods can take advantage of available resources to improve software engineering skills across Centers and NASA. Guidelines would introduce and strengthen the use of these methods which can be implemented through development and performance plans.

9. Develop a set of predictive software defect data and a process for assessing software testing metric data against it. Use the set to status progress during NASA software testing phases and in software test reviews (TE1, ME2).

   One of the best practices seen was to status software during reviews. This recommendation suggests having statuses at software test reviews. To do this a common set of data and a process must be available. An expected result of this practice is to reduce software defects and development/testing costs. Once in place, this process and information will strengthen the software engineering cost and schedule inputs since they can be tracked and accurately estimated and in the longer term provide the ability to explain/defend software cost and schedule and the impact of changes. Although there are several areas within the software life cycle where this approach could be applied, testing is a suggested starting point and once the process is successful, the approach can be expanded.

10. Assess Agency-wide licenses for commonly used software tools (TE4, TO2, ME2).

    Since there were many tools identified in the study, further review and assessment is needed but it's clear that common software test and static analysis tools can provide improvement to software development and testing, potentially reducing software testing time; also tools to capture metrics would be in line with the findings regarding development of metrics. Additionally, having a common set of tools available across the Agency would then contribute to the remedy of tool availability for small projects.

11. Fill the knowledge gap in common software engineering practices for new hires and co-ops (PO1A, TR1, TR3, TR4).

    Based on data collected through this study, university graduates may not have a firm foundation in the awareness and use of common software engineering practices. This has implications for new hire/co-op assignments and requires NASA to share common software engineering knowledge with new hires and co-ops.

12. Work through the STEM program with universities in strengthening education in the use of common software engineering practices and standards (PO1C).

    A corresponding approach to the previous action is to provide knowledge and expectations of common software engineering practices to university software engineering programs instead of waiting until the software engineer is employed and in NASA training. This recommendation lends itself to other Agency goals of outreach and involvement in the software engineering community of practice (education).

13. Follow up this benchmark study with a deeper look into what both internal and external organizations perceive as the scope of software assurance, the value they expect to obtain from it, and the shortcomings they experience in the current practice. This should include mutual action, with software assurance making its risk mitigation capabilities better known as well as taking customer and contractor inputs (SA1, TE5).

    The software assurance role across NASA and other organizations was not consistent with many of the software activities, including participation/role in reviews, testing, acquisition, software safety, and reliability. The differing expectations should be reviewed to see if one implementation is optimal thus improving the quality while streamlining the software assurance work.

8.2.3. Collaboration and Further Interactions

There is one additional recommendation that should be considered. It captures the success of this effort propelling it forward to create a continuous improvement mechanism and keeps NASA Software Engineering at the forefront of the discipline.

14. Continue interactions with external software engineering environment through collaborations, knowledge sharing, and benchmarking.

    The NASA participants in this study felt that the data collected and the process of collecting, connecting with other NASA Centers and external organizations, was beneficial to their knowledge and understanding of the software engineering community. It is an excellent catalyst for improvement and innovation. Many of the external participants were excited to participate and were interested in NASA's results, suggesting that there are benefits for them as well. In addition to sharing knowledge and awareness of what other organizations are doing, there are opportunities for sharing other types of assets, such as services of personnel with specific training or certifications, training classes, and even assets such as process descriptions, templates and tools for improving software engineering practices.

8.3. Summary

With software being a critical element of spaceflight, NASA must continue to advance software engineering just as NASA continues to advance hardware technologies and the ultimate prize of pushing the boundaries of flight. This benchmark provides the guidance on where investment--not just dollars, but also attention and resources--will continue the growth and innovation necessary to help keep NASA's missions safe and

successful. The NASA Benchmark Team believes in continued improvement and will be advocates and change agents where necessary. This report captures data that was the most relevant and compelling for NASA's use in today's environment.

The report also analyzes the data so that it can be used, or more directly stated, put into place by following through with the resulting recommendations. These recommendations are within the scope of the NASA software engineering community and the accompanying organizations such as the SWG, Mission Software Steering Committee (MSSC), SAWG, NESC, NSC, and will be taken into consideration along with the most current top software issues, and other software recommendations to be folded into the yearly planning cycles for the various groups. As budgets allow, the recommendations will be transformed into executable plans (action items with assignees and due dates) to improve the overall state of software engineering at NASA. The NASA Software Benchmark Team and the SWG look forward to resolving some of NASA's current software challenges and seeing what software engineering will look like in the future.

APPENDIX A – ACRONYMS AND ABBREVIATIONS

Acronym	Definition
AAO	Audits and Assessments Office
APPS	Agency Processes and Principles for Software
ASIC	Application-Specific Integrated Circuit
CAR	Causal Analysis Resolution
CASRE	Computer Aided Software Reliability Estimation
CBT	Computer-based Training
CCB	Configuration Control Board
CDR	Critical Design Review
CE	Complex Electronics
CIO	Chief Information Officer
CMMI	Capability Maturity Model Integration
CoP	Community of Practice
COTS	Commercial off-the-shelf
CPLD	Complex Programmable Logic Device
CSRM	Certified School for Risk Managers
DAR	Decision Analysis Resolution
DAU	Defense Acquisition University
DFM	Design for Manufacturability
DID	Data Item Description
DoD	Department of Defense
DOORS	Dynamic Object-Oriented Requirements System
EPL	Eclipse Public License
ESA	European Space Agency
ESD	Electro-static Discharge
FAA	Federal Aviation Administration
FAR	Federal Acquisition Regulations
FMEA	Failure Mode Effects Analysis
FPGA	Field Programmable Gate Array
FTA	Fault Tree Analysis
FTE	Full-time Equivalent

FY	Fiscal Year
GOTS	Government Off-the-Shelf
GPR	GSFC Procedural Requirement
GRC	Glenn Research Center
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HDL	Hardware Description Language
HQ	Headquarters
IEEE	Institute of Electrical and Electronics Engineers
IRAD	Independent Research and Development
ISO	International Standards Organization
IT	Information Technology
IV&V	Independent Verification and Validation
JPR	JSC Procedural Requirement
JSC	Johnson Space Center
KSC	Kennedy Space Center
KSLOC	Thousands (k) of Source Lines of Code
LDRA	Liverpool Data Research Associates
MIL	Military
ML#	Maturity Level
MOA	Memorandum of Agreement
MOTS	Modified Off-the-Shelf
MOU	Memorandum of Understanding
MSSC	Mission Software Steering Committee
NESC	NASA Engineering and Safety Center
NHB	NASA Handbook
NMI	NASA Management Instruction
NPD	NASA Policy Directive
NPR	NASA Procedural Requirements
NSC	NASA Safety Center
OCE	Office of the Chief Engineer
OCIO	Office of the Chief Information Officer
OJT	On-the-Job Training
OSMA	Office of Safety and Mission Assurance

PAL	Process Asset Library
PDLM	Product Data and Life Cycle Management
PDR	Preliminary Design Review
PLD	Programmable Logic Devices
PPQA	Process and Product Quality Assurance
PSP	Personal Software Process
QA	Quality Assurance
QAARS	Quality Audit, Assessment and Review
R&D	Research and Development
RFP	Request for Proposal
RTOS	Real-time Operating Systems
SA	Software Assurance
SAWG	Software Assurance Working Group
SCM	Software Configuration Management
SEB	Source Evaluation Board
SEER-SEM	Software Evaluation and Estimation of Resources - Software Estimating Model
SEI	Software Engineering Institute
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SME	Subject Matter Expert
SMP	Software Management Plan
SoC	System-on-Chip
SQA	Software Quality Assurance
SRR	Systems (or Software) Requirements Review
STD	Standard
STEM	Science, Technology, Engineering and Mathematics
STEP	Safety and Mission Assurance Technical Excellence Program
SW	Software
SWE	Software Engineering
SWEBOK	Software Engineering Body of Knowledge
SWG	Software Working Group
TSP	Team Software Process
V&V	Verification and Validation

VCS	Voluntary Consensus Standards
VHDL	Very High-level Design Language
xPR	Center-level Procedural Requirements

APPENDIX B –TOP SOFTWARE ISSUES AT NASA

B.1. Top NASA Software Issues of 2010

The following list comes from the Microsoft PowerPoint report entitled *Top NASA Software Issues of 2010* (August, 2010) compiled during discussion at the Software Working Group (SWG) meeting at Kennedy Space Center (KSC) in April, 2010 and from the SWG/MSSC Joint Sessions at Plum Brook in August, 2010.

1. Internal NASA-wide requirements [NPDs, NPRs, and NASA Standards (STD)].
2. Software Cost Estimation.
3. Software workforce level.
4. Systems engineering/software engineering interface
5. Small project implementations.
6. Empowerment of software personnel.
7. Software requirements.
8. Complex electronics.
9. Training and skill development.
10. Insufficient attention to software on contracts.
11. Prototype to operations transition and software class increase.
    (Rolled up into issue #1 as this item's concern is essentially the increase in requirements when prototypes provide successful demonstrations.)
12. Lessons learned (software).
    (Rolling the lessons learned from previous programs back into the NASA standards, procedures, etc.)
13. Software architecture analysis and review*.
    [Effective application of software architecture and architecture analysis/review (from the Flight software Complexity study).]
14. Commercial off-the-shelf software.
    [Incorporating COTS and open source products into mission critical software developments (e.g., flight software, ground software, and the software development environment) while maintaining rigorous processes.]
15. Model Based software Development*.
    [Incorporating model-based development into the existing development and design review process. Incorporating model-based development into requirements, test, and software assurance and hazard analysis. For reviews (PDR, CDR, etc.).]

Notes:*Top issue at one of the NASA Centers

B.2. Top Software Engineering Challenges for NASA over the last 8 years

Software Engineering Challenges 2013

Software workforce issues
Software requirements issues
Investment in software engineering issues
Small project issues
Software costing issues
Software reuse issues
Acquisitions that include software issues
Software security & cyber security issues
Software metrics
It related issues
Open source utilization
Model based software development issues

Others Potential Challenges in 2013

Programmable logic devices (complex electronics)
Software architecture analysis and review
Fault management
Improving the system engineering and software engineering interface
Agile software development methods
Improving the interface between software engineering and the center software release authorities interface
Software engineering technical authority
Safety-critical software

Top Software Issues 2010

Internal NASA-wide requirements (NPD, NPR, & standards)
Software cost estimation
Software workforce level
Systems eng. / software eng. Interface
Small project implementations
Empowerment of software personnel
Software requirements
Complex electronics
Training & skill development
Insufficient attention to software on contracts
Software architectural analysis & review
Model based software development

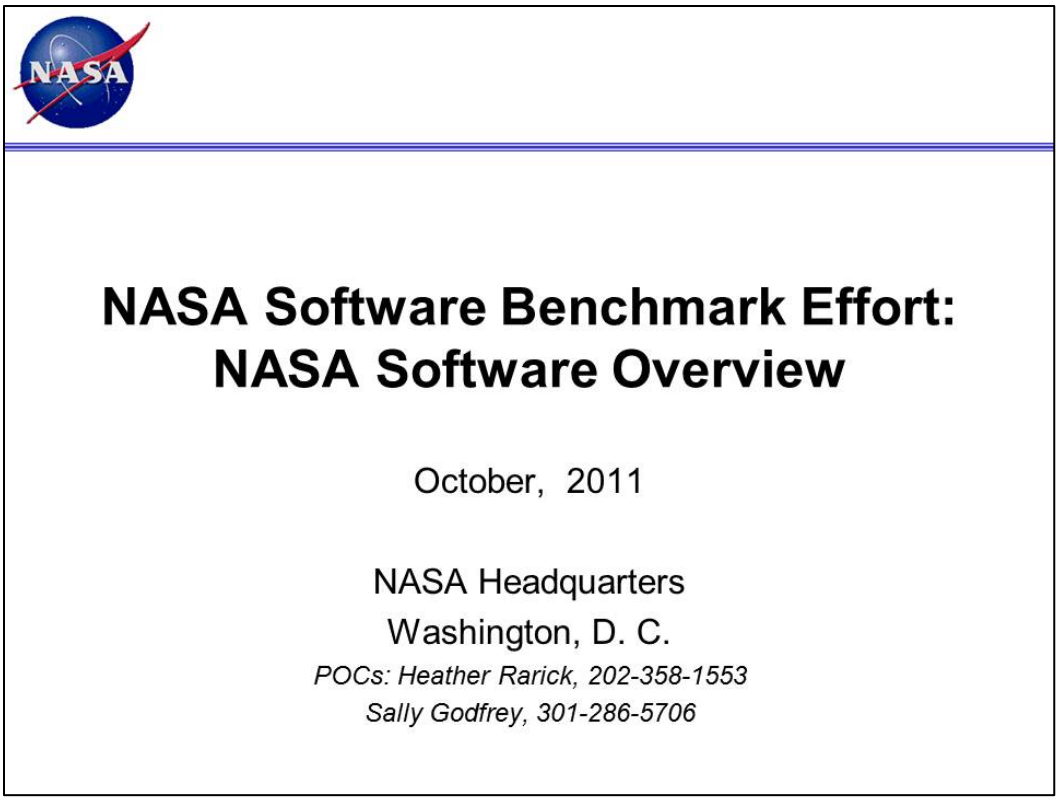
Others Identified Challenges in 2010

Prototype to operations transition & software class increase
Software engineering lessons learned
Software architecture, analysis & review
COTS software usage
Model-based software development

Top Software Issues 2007

Software requirements
Internal NASA-wide requirements (NPD, NPR, & standards)
Software engineering training & skill development
Programmable logic devices (complex electronics), FPGA, PLD, etc. (blurring of hardware – software boundary)
Insight/oversight of contractor software development
Software engineering tools
Empowerment of program/project software personnel
Software metrics/measurement
COTS software usage
Software cost estimation - need a standard approach
Small project implementations
Empowerment of software personnel
Software requirements
Complex electronics
Training & skill development
Insufficient attention to software on contracts
Software architectural analysis & review
Model-based software development
Internal NASA-wide requirements (NPD, NPR, & standards)
Software cost estimation
Software workforce level
Systems eng. / software eng. interface

APPENDIX C – NASA PRESENTATION TO PARTICIPATING ORGANIZATIONS



NASA Software Benchmark Effort: NASA Software Overview

October, 2011

NASA Headquarters
Washington, D. C.

POCs: Heather Rarick, 202-358-1553
Sally Godfrey, 301-286-5706

Content

- NASA's Mission and Organization
- NASA's Software Engineering
- NASA's Software Assurance
- Software Benchmarking
- Agency Directives
  - Software and Software Assurance Directives, Requirements and Standards
  - Agency Software Requirements
  - Agency Software Assurance and Software Safety Standards
  - Center Software Requirements
  - Software Classifications
  - Requirements Mapping Matrix
- CMMI at NASA
- Software Training
- NASA's Top 10 Software Issues

Slide 2

NASA's Mission & Organization

NASA's mission is to pioneer the future in space exploration, scientific discovery and aeronautics research.

NASA conducts its work in four principal organizations, called mission directorates:

- **Aeronautics**: pioneers and proves new flight technologies that improve our ability to explore and which have practical applications on Earth.
- **Exploration Systems**: creates capabilities for sustainable human and robotic exploration.
- **Science**: explores the Earth, solar system and universe beyond; charts the best route of discovery; and reaps the benefits of Earth and space exploration for society.
- **Space Operations**: provides critical enabling technologies for much of the rest of NASA through the space shuttle, the International Space Station and flight support.

NASA's Organization Chart

NASA Headquarters, Washington DC, provides overall guidance and direction to the agency, under the leadership of the NASA Administrator/Charlie Bolden. It includes a Mission Support Directorate and Administrative Offices. The Mission Support Directorate includes Human Capital Management, Strategic Initiatives, HQ Ops, Agency Ops and NASA Shared Services Center. The Administrative Offices include Chief Finance, Chief Information Technology, Chief Engineer, Chief Technologist, Chief Scientist, Chief Safety & Mission Assurance, etc.

Ten field centers and a variety of installations conduct the day-to-day work, in laboratories, on air fields, in wind tunnels and in control rooms.

Ames Research Center - Moffett Field, CA
IT, fundamental aeronautics, bio and space science technologies

Dryden Flight Research Center - Edwards Air Force Base, CA
Flight research

Glenn Research Center - Cleveland, OH
Aero-propulsion and communications technologies

Goddard Space Flight Center - Greenbelt, MD
Earth, the solar system, and Universe observations

Jet Propulsion Laboratory - Pasadena, CA
Robotic exploration of the Solar System

Johnson Space Center – Houston, TX
Human space exploration

Kennedy Space Center - Cape Canaveral, FL
Prepare and launch missions around the Earth and beyond

Langley Research Center - Hampton, VA
Aviation and space research

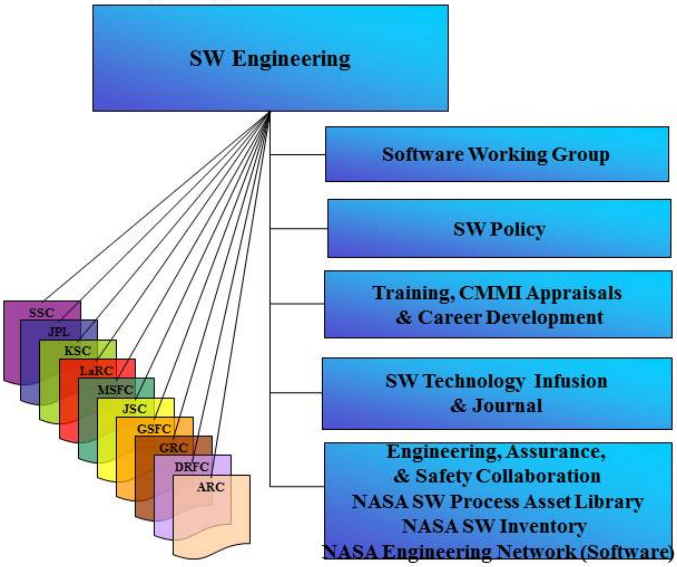
Marshall Space Flight Center - Huntsville, AL
Space transportation and propulsion technologies

Stennis Space Center - Bay St. Louis, Mississippi
Rocket propulsion testing and remote sensing technology

Slide 3

Software Engineering

"The NASA Headquarters Office of the Chief Engineer shall lead, maintain, and fund a NASA Software Engineering initiative to advance software engineering practices." NPR 7150.2A



Also includes: NASA SW Process Asset Library, SW Metrics, NASA SW Inventory.

Slide 4

Software Assurance

Software Assurance is an umbrella risk identification and mitigation strategy for safety and mission assurance of all NASA's software, under direction of the Office of Safety and Mission Assurance.

SW Quality **SW Safety** **SW Reliability** **V&V** **IV&V**

Quality Engineering – analysis of the product
Quality Control – processes and products are in place
Quality Assurance – processes and products are the right ones

SW Assurance is a participant in the V&V function which is managed by the project.

A systematic approach to identifying, analyzing, tracking, mitigating and controlling software contributions to hazards and hazardous functions performed by software (data and commands) to ensure safer software operation *within a system.*

-Build in a certain reliability and/or fault level and/or failure level
-Find the problem areas and where to best fix them via analyses (e.g. FMEA, FTA, CSRM/DFM, SoftRel, CASRE, others?)
-Workmanship
-Prediction of problem areas

-- NASA IV&V applied only to selected critical projects and then only to most critical software
-- provides analyses and in depth look at product
-- Independent technically, managerially & financially

NOTE: Software security is managed by the Chief Information Office

Slide 5

Software Benchmark

- NASA Software Working Group Chair initiated a benchmarking effort for FY11 to help ensure NASA incorporates new techniques and tools; reviews and learns from external organizations and, is cognizant of the current and near-term environment.
- The NASA SW Working Group (which includes Software Assurance) helped identify the key target areas for the benchmark, consistent with the Top 10 Software Issues:
  - SW Policies, Level of Detail and Use of Industry Standards
  - How to Maintain Rigor for Small Projects
    - Small projects at NASA are defined as using five or fewer software engineers.
  - How to Maintain Organizational Requirements in SW Acquisitions
  - In-house Training Programs
  - Testing
  - CMMI Maturity Level Benefits and Benefits of Advancement
- The questions are targeted for organizations that develop and/or acquire software comparable to NASA flight software (criticality A, B and C) and includes software assurance functions.
- Three NASA centers are being interviewed as part of the benchmarking effort. In addition, the following organizations (approximately 3 of each) are also being interviewed
  - US Government Agencies
  - NASA Contractors
  - Universities working with NASA on projects
- A summary of findings will be documented and presented along with recommendations on how and where to use relevant information. Your organization is welcome to request our summary.

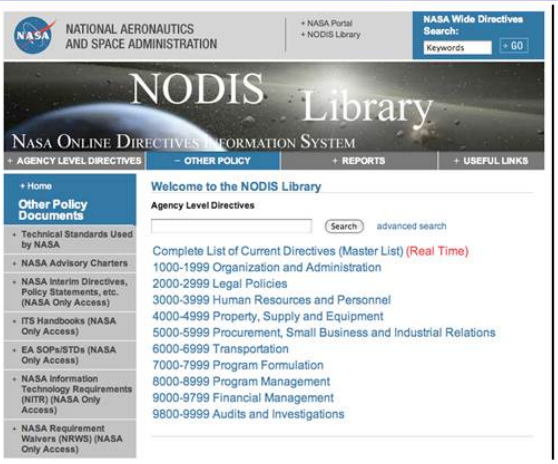
Slide 6

Agency Directives

NASA Directives
NASA Policy Directives (NPDs)
NASA Procedural Requirements (NPRs)
Standards

Key Directives for Software
NPD 7120.4D: NASA Engineering and Program/Project Management Policy
→ NPR 7150.2A: Software Engineering Requirements
NPR 2810.1A: Security of Information Technology
NPR 2210.1A: External Release of NASA Software

Key Directives for Software Assurance
NASA STD 8719.13: Software Safety Standards
NASA STD 8739.8: Software Assurance Standards



Slide 7

Agency Software Requirements
(NPR 7150.2A)

- Requirements in NPR 7150.2A are levied by classification of software and safety criticality
  - Classes A through E are mission related software
  - Classes F through H are non-engineering software
- Requirements are fairly high level
  - Most stringent requirements are levied on Classes A, B, and F
  - The SW Assurance Standards are invoked in NPR 7150.2A.
- Documentation requirements are specified
- Requirements identified for agency and for centers
  - Some requirements for lower classes of software allow for implementation flexibility at a center
  - Includes a required software improvement plan for each center
- Requirements contain many industry best practices as documented in CMMI Maturity Levels 2 and 3
  - CMMI requirement for Classes A, B, C
- Waiver process
  - Some requirements can only be waived at Agency level
  - Others can be waived at a center by Engineering Technical Authority (ETA) for Software
- Compliance
  - Center ETAs are required to keep records of waivers submitted, granted
  - Office of the Chief Engineer (OCE) surveys are conducted at Centers; HQ to check on compliance

Slide 8

Agency Software Assurance & Software Safety Standards

- NASA-STD-8739.8, SW Assurance applies to all software assurance activities during the **entire software life cycle** of the software developed or acquired for NASA, either internally or externally, including the incorporation of open source, COTS, GOTS, or MOTS into NASA systems.
  - Legacy and reuse software products are also covered with a focus on how they fit into the new systems.
  - The requirements of this Standard are applicable whenever NASA is either the acquirer or provider, and to the extent specified in the contract or other agreement such as Memorandum of Agreement.
- The Standard provides
  - a common framework for consistent SA practices across NASA programs
  - basic procedures for establishing and maintaining an effective software assurance program
  - specific acquirer and provider requirements for software assurance
  - specific requirements for each software assurance discipline (Quality, Reliability, Safety, Verification & Validation, IV&V)
  - tailoring of software assurance activities commensurate with software class, size, criticality, complexity, and risk
- NASA-STD-8719.13, SW Safety applies to all projects and defines how to
  - determine safety criticality of software
  - emphasize Software Safety as part of System Safety
  - integrate safety into the full software life cycle
  - analyze the software, system, and interfaces, from beginning to end
  - document safety plans, decisions, processes, and results
  - trace software safety requirements through all software phases
  - report and resolve problems and discrepancies
  - evaluate off-the-shelf software for safety
  - "Certify" the software for safe operations

Slide 9

Center SW Requirements

Agency NPD/R requirements flow to each Center . . . ,

Center procedural requirements are GPRs for GSFC, JPRs for JSC, etc.

In many cases, these center-level documents are part of the ISO documentation of a center

Most centers have their own specific set of software processes, procedures, usually in their Process Asset Libraries (PAL)

As an example, GSFC has flowed down NPR 7150.2 into 3 GPRs:

GPR 7150.1: Covers make/buy decision, software classification, safety criticality, sets up project, directs users to appropriate follow-on GPR

GPR 7150.2: Covers in-house software development and maintenance

GPR 7150.3: Covers requirements for acquisition projects and contracted software

Engineering Technical Authority

6.2.1 The designated Engineering Technical Authority(s) for requirements in this NPR, which can be waived at the Center level, shall be approved by the Center Director. [SWE-122]

- Note: The designated Engineering Technical Authority(s) for this NPR is a recognized NASA software engineering expert.
- Typically, Center Directors designate an Engineering Technical Authority for software
  - from their engineering organization for software classes A through E,
  - The designation of Engineering Technical Authority(s) is documented in the Center Technical Authority Implementation Plan.
  - Appendix D (last column) shows which requirements can be waived at the Center level.

Slide 10

NASA Software Classifications

	Classification	Characteristics
Engineering	**A - Human Rated Space Software Systems**	All space flight software subsystems (ground and flight) developed and/or operated by or for NASA that are needed to perform a primary mission objective of human space flight and directly interacts with human space flight systems.
	B - Non-Human Space Rated Software Systems	Flight and ground software that must perform reliably to accomplish primary mission objectives, or major function(s) in Non-Human Space Rated Systems.
	C - Mission Support Software or Aeronautic Vehicles, or Major Engineering/Research Facility Software	Flight or ground software that is necessary for the science return from a single (non-primary) instrument, or flight or ground software that is used to analyze or process mission data, or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems, or software used for the testing of space assets, or software used to verify system requirements of space assets by analysis, or software for space flight operations, that is not covered by Class A or B.
	D - Basic Science/Engineering Design and Research and Technology Software	Ground software that performs secondary science data analysis, or ground software tools that support engineering development, or ground software used in testing other Class D software systems, or ground software tools that support mission planning or formulation, or ground software that operates a research, development, test, or evaluation laboratory (i.e., not a Major Engineering/Research Facility), or ground software that provides decision support for non-mission critical situations.
	E - Small Light Weight Design Concept and Research and Technology Software	Software developed to explore a design concept or hypothesis, but not used to make decisions for an operational Class A, B, or C system or to-be built Class A, B, or C system, or software used to perform minor desktop analysis of science or experimental data.
Non-engineering	**F - General Purpose Computing Software (Multi Center or Project Use)**	General purpose computing software used in support of the Agency, multiple Centers, or multiple programs/projects, as described for the General Purpose Infrastructure To-Be Component of the NASA Enterprise Architecture, Volume 5 (To-Be Architecture), and for the following portfolios: voice, wide area network, local area network, video, data Centers, application services, messaging and collaboration, and public Web.
	G - General Purpose Computing Software (Single Center or Project Use)	General purpose computing software used in support of a single Center or project, as described for locally deployed portions of the General Purpose Infrastructure To-Be . . . ,
	H - General Purpose Desktop Software	General purpose desktop software as described for the General Purpose Infrastructure To-Be Component (Desktop Hardware and Software Portfolio) of the NASA Enterprise Architecture, Volume 5 (NASA To-Be Architecture).

Slide 11

Applicable and Non-Applicable Software Requirements

← OCE Classes → ←CIO Classes →

				Class A	Class B	Class C	Class D	Class E	Class F	Class G	Class H
SW Requirements Development	Document	49	Project	X	X	X	X	P (Center)	X	X	
	SW requirements	50	Project	X	X	X	X		X	X	
	Flow-down & derived req.	51	Project	X	X	X			X (not OTS)	P (Center)	
	Bi-directional trace	52	Project	X	X				X (not OTS)	P (Center)	
SW Requirements Management	Manage req. change	53	Project	X	X	X	X	P (Center)	X	X	
	Corrective action	54	Project	X	X				X	X	
	Requirements Validation	55	Project	X	X	X			X	X	
SW Design	Document design	56	Project	X	X	P (Center)			X (not OTS)	X (not OTS)	
	Architecture	57	Project	X	X	P (Center)	P (Center)		X (not OTS)	X (not OTS)	
	Detailed design	58	Project	X	X				X (not OTS)	X (not OTS)	
	Bi-directional trace	59	Project	X	X				X (not OTS)	X (not OTS)	
SW Implementation	Design --> code	60	Project	X	X	X	X		X (not OTS)	X (not OTS)	
	Coding standards	61	Project	X	X				X (not OTS)	X (not OTS)	
	Unit test	62	Project	X	X	X	P (Center)	P (Center)	X (not OTS)	X (not OTS)	
	Version Description	63	Project	X	X	P (Center)	X		X (not OTS)	X (not OTS)	
	Maintain Traceability	64	Project	X	X				X (not OTS)	X (not OTS)	
SW Testing	Plan, procedures, reports	65	Project	X	X	X	P (Center)		X	P (Center)	
	Perform testing	66	Project	X	X	X	X		X	P (Center)	
	Test for compliance	67	Project	X	X	X			X	P (Center)	
	Evaluate test results	68	Project	X	X	X	X		X	P (Center)	
	Doc. defect & track	69	Project	X	X	X	P (Center)		X	P (Center)	
	Models, simulations, tools	70	Project	X	X				X	P (Center)	
	Update plans & procedures	71	Project	X	X	X			X	P (Center)	
	Maintain Traceability	72	Project	X	X	X	X		X	P (Center)	
	Platform or Hi-Fidelity sim.	73	Project	X	X	X			X	P (Center)	

X = Required
Blank = Not Required
P(Center) = Center defined process (i.e. subset OK)

"Through the use of the Requirements Mapping Matrix, the number of applicable requirements and their associated rigor are scaled back for less critical software classes", - NPR 7150.2, Section P.2.1

12

NASA CMMI Requirement
(NPR 7150.2A)

[SWE-032] The project shall ensure that software is acquired, developed and maintained by an organization with a non-expired Capability Maturity Model Integration® for Development (CMMI-DEV) rating as measured by a Software Engineering Institute (SEI) authorized or certified lead appraiser as follows:

- **For Class A software: CMMI-DEV Maturity Level 3 Rating or higher for software, or CMMI-DEV Capability Level 3 Rating or higher in all CMMI-DEV Maturity Level 2 and Maturity Level 3 process areas for software.**
- **For Class B software: CMMI-DEV Maturity Level 2 Rating or higher for software, or CMMI-DEV Capability Level 2 Rating or higher for software in the following process areas:**
  - **a. Requirements Management.**
  - **b. Configuration Management.**
  - **c. Process and Product Quality Assurance.**
  - **d. Measurement and Analysis.**
  - **e. Project Planning.**
  - **f. Project Monitoring and Control.**
  - **g. Supplier Agreement Management (if applicable).**
- **For Class C software: The required CMMI-DEV Maturity Level for Class C software will be defined per Center or project requirements.**

13

Software Training

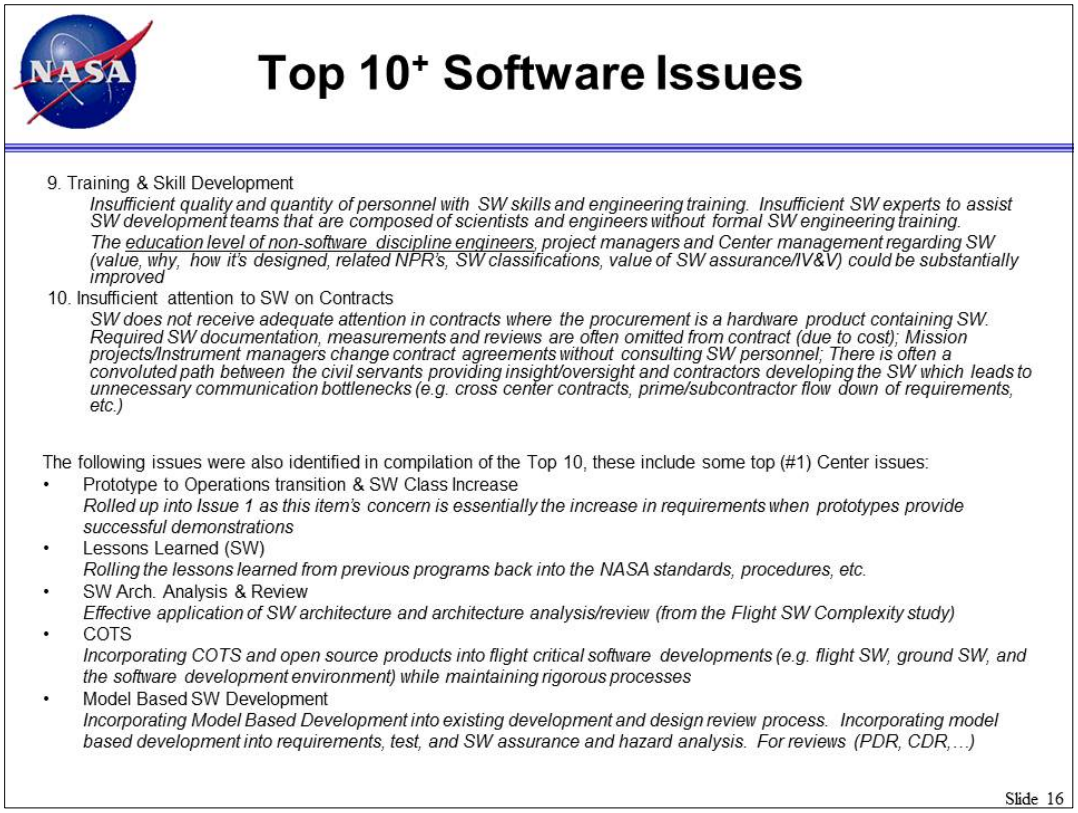
- Training needs are identified by the Agency and by the centers, where appropriate the Agency provides the training, otherwise the centers determine their priorities and schedule training. Many of the training classes are procured.
- NASA supports software training on the Agency level, both through Office of Chief Engineer and Office of Safety and Mission Assurance, some examples
  - SWE 301: Software Engineering Management: Intense week long residential class on software project mgmt
  - Perspective-Based Inspections
  - NASA Software Standards Class
  - Configuration Management
- Office of Safety and Mission Assurance supports Software Assurance training via the STEP Curriculum (Safety and Mission Assurance Technical Excellence Program), some examples which include web-based and instructor lead courses
  - Basics Of Software Assurance For Practitioners
  - Intermediate Software Quality Assurance
  - Introduction To Software Testing
  - Software Auditor Skills
  - Reliability and Maintainability training is worked collaboratively with DAU (Defense Acquisition University)
  - Introduction to Software Reliability
  - Software Safety courses; intro and practitioner levels
- Many of the Centers have extensive training programs:
  - MSFC brings in software classes about once a month
  - GSFC and JPL have regular "lunch-time" seminars on various software topics
  - Many centers have developed their own specific software training classes

Slide 14

Top 10+ Software Issues

1. Internal NASA-wide requirements (NPD, NPR, & Standards)
   Application of Software Directives, Requirements and Standards, can be complex: heritage SW, external providers, inter-center SW development. Application can be difficult for prototype software, for lower level classifications and for simulations. New requirements require new processes to be developed.
2. Software Cost Estimation
3. Software Workforce level
   Software Engineering and Assurance workforce level is insufficient to meet: NASA's SW Engineering, Assurance, and Safety requirements, NASA HQ software-related request and requirements (for various data, information, and actions) and addressing of important long-term strategic software issues (e.g., software costing issue, …).
4. Systems Software Interface
   Communication between the systems and SW levels on projects is weak (documentation, models, safety critical elements, etc.). Fundamental systems engineering decisions are made without adequate participation of SW engineering and are not synchronized.
5. Small Project Implementations
   Small projects (hence, small staff) are often challenged in understanding and complying with the requirements imposed on SW from both NPR 7150.2A and the Software Assurance Standard (NASA-STD-8739), in addition to requirements from the relevant project (NPR 7120.5) and NASA Systems Engineering (NPR 7123.1) requirements. The number of requirements and documents are disproportional to the funding and risk levels of small missions, technology demonstration projects (especially the new programs and projects sponsored by the NASA Office of Chief Technologist) and smaller robotics spacecraft with accepted high payload risk.
6. Empowerment of SW Personnel
   Getting projects to recognize importance of SW, to bring SW people in early and the importance of SW in large procurements.
7. SW Requirements
   The impact of requirements upon software is not consistently quantified and managed in development nor sustainment.
8. Complex Electronics
   Increasing tendency to put mission critical functionality in FPGAs because of the perception that there are cost and schedule benefits to doing this. Perception that complex electronics are not held to the same level of rigor in requirements, development, testing, and verification as software. Need combination of h/w and s/w approaches to developing complex electronics

Slide 15



Top 10+ Software Issues

9. Training & Skill Development
   Insufficient quality and quantity of personnel with SW skills and engineering training. Insufficient SW experts to assist SW development teams that are composed of scientists and engineers without formal SW engineering training.
   The education level of non-software discipline engineers, project managers and Center management regarding SW (value, why, how it's designed, related NPR's, SW classifications, value of SW assurance/IV&V) could be substantially improved
10. Insufficient attention to SW on Contracts
   SW does not receive adequate attention in contracts where the procurement is a hardware product containing SW. Required SW documentation, measurements and reviews are often omitted from contract (due to cost); Mission projects/Instrument managers change contract agreements without consulting SW personnel; There is often a convoluted path between the civil servants providing insight/oversight and contractors developing the SW which leads to unnecessary communication bottlenecks (e.g. cross center contracts, prime/subcontractor flow down of requirements, etc.)

The following issues were also identified in compilation of the Top 10, these include some top (#1) Center issues:

- Prototype to Operations transition & SW Class Increase
  Rolled up into Issue 1 as this item's concern is essentially the increase in requirements when prototypes provide successful demonstrations
- Lessons Learned (SW)
  Rolling the lessons learned from previous programs back into the NASA standards, procedures, etc.
- SW Arch. Analysis & Review
  Effective application of SW architecture and architecture analysis/review (from the Flight SW Complexity study)
- COTS
  Incorporating COTS and open source products into flight critical software developments (e.g. flight SW, ground SW, and the software development environment) while maintaining rigorous processes
- Model Based SW Development
  Incorporating Model Based Development into existing development and design review process. Incorporating model based development into requirements, test, and SW assurance and hazard analysis. For reviews (PDR, CDR,…)

Slide 16

APPENDIX D – QUESTIONS ASKED OF PARTICIPATING ORGANIZATIONS

D.1. High Level

This set of questions was provided to the participating organizations in preparation for the interviews. The list is an abridged version of the full list of questions generation by the Benchmark Team (D.2) and was shortened so that the participating organizations did not feel the need to spend significant time preparing for the interview.

These questions/answers will gather data and best practices that can then be shared throughout NASA and used/tested where they may be of benefit. The target areas include: Software Policies, Level of Detail and Use of Industry Standards; How to Maintain Rigor for Small Projects; How to Maintain Organizational Requirements in software Acquisitions; In-house Training Programs; Testing; and, CMMI Maturity Level Benefits and Benefits of Advancement. These questions are targeted for organizations that develop and acquire software comparable to NASA flight software (criticality A, B and C) including SQA and reliability functions. NOTE: Small projects at NASA are defined as using five or fewer software engineers.

Background

- Please describe your software organization, structure and projects supported. An example or two of projects being or previously worked would be beneficial.
    - Structure – management and engineering roles, team responsibilities and roles, team sizes and composition, support structure (integrated, matrixed, distributed); Include SQA, safety and reliability activities take place, when they are performed and who performs them.
    - How is the software engineering organization integrated with software assurance? Operations? Requirements development and management? Systems Engineering? Project Management?
    - How do you “classify ” your software? Explain the relationship between classifications and QA guidance/requirements?
    - How do you determine the criticality of your software and the safety required?
    - Projects – how many, types, sizes, length, maintenance vs new development, criticality of sw (include definitions of criticality levels), in-house vs acquisition.
    - CMMI Maturity Level.

  - Life cycle model and languages used, use of commercial, military, or government off-the-shelf (COTS/MOTS/GOTS) software.
  - Use of Agile, PSP/TSP.
- How does your organization train and develop software engineers and software quality engineers? Would you describe the 3 most beneficial classes for software engineers?
  - Organizational responsibility for development or acquisition of training curriculum.
  - Type of training given: in-house vs external training programs (or combined)? What has led to using in-house vs external training?
  - Who has responsibility for identifying the training needs, how and when training is given?
  - Is training given at individual, group or project level, career levels? If so, what kind and when?
  - Is training mandatory or optional? How much time is allowed or expected for training per person?
  - How is mentoring and OJT included in developing individuals (informal, structured, required)?
  - What are your preferred methods and media for training?
  - Describe how your training program addresses proficiency training, system engineering, metrics, risk management and project management.
  - Describe any training provided to management (line management and/or project managers)?
  - How does your organization manage training that might be needed for a specific project (just in time training)?
- Do any of your projects include software acquisitions as a deliverable or as a piece of the complete software project? If so, could you describe your acquisition process, specifically how it integrates into your software organization?

Software Policies

- Please identify and summarize software policies, directives or requirements you have that are implemented organization-wide (governing documents)?
- Identify the source of software policies, directives or requirements which are applied to your organization's software activities? Who (organizationally) is responsible for these high-level documents and how/when are they updated?
- Describe how your organization ensures compliance with these governing documents?
- Please explain how these documents are communicated to the users?

- How are policies and requirements included in contracted work (e.g., acquisitions)?
- How are project reviews and milestones coordinated with software reviews and milestones?

Rigorous Software Processes for Small Projects

- Please describe the scope (size and criticality) of small projects.
- Are small projects (all criticality levels) required to comply with software policies and requirements? If so, what are some of the key ways in which small projects are able to comply? If small projects are not required to comply, how are small projects governed?
- Does your CMMI statement of work include or exclude small projects?
- Does your organization have any infrastructure to support a collection of small projects?
- What methods or tools have you found that work well for small projects?
- How does your organization satisfy good software practices with limited resources and funds allocated to small projects?

Software Testing

- Please describe software testing: include strategy and scope, test plans, testing types, success/completion criteria.
- What is the organization and composition of your typical software test team?
- Please identify any tools used or autonomous testing performed?

CMMI Maturity Level

- Who decided and how did your organization decide to ascertain your current maturity level?
- How were impacts to policies, requirements, training and other organizational structure handled?
- Have you been able to measure or identify any benefits at previous and current maturity levels? What are your top 3 areas of improvement and what impact did they have?
- How have you overcome major challenges that you have faced with pursuing maturity levels?
- How have you been able to reduce costs of appraisals?

For CMMI Maturity Level 4 and 5:

- Could you share some of your measurement programs currently in place?
- Does it cost less to operate at a higher maturity level or are other benefits more significant?

D.2. Detailed

This set of questions was provided to the Benchmark Team in preparation and for use during the interviews. The list is an unabridged version of the list of questions provided to the participating organizations (D.1). This more extensive list was intended to ensure additional information and details were consistently discussed and noted.

These questions/answers will gather data and best practices that can then be shared throughout NASA and used/tested where they may be of benefit. The target areas include: Software Policies, Level of Detail and Use of Industry Standards; How to Maintain Rigor for Small Projects; How to Maintain Organizational Requirements in software Acquisitions; In-house Training Programs; Testing; and, CMMI Maturity Level Benefits and Benefits of Advancement. These questions are targeted for organizations that develop and acquire software comparable to NASA flight software (criticality A, B and C) including SQA and reliability functions. NOTE: Small projects at NASA are defined as using five or fewer software engineers.

Background

- Please describe your software organization, structure and projects supported. An example or two of projects being or previously worked would be beneficial.
    - Structure – management and engineering roles, team responsibilities and roles, team sizes and composition, support structure (integrated, matrixed, distributed); Include SQA, safety and reliability activities take place, when they are performed and who performs them.
    - How is the software engineering organization integrated with software assurance? Operations? Requirements development and management? Systems Engineering? Project Management?
    - How do you “classify ” your software? Explain the relationship between classifications and QA guidance/requirements?
    - How do you determine the criticality of your software and the safety required?

  - Projects – how many, types, sizes, length, maintenance vs new development, criticality of sw (include definitions of criticality levels), in-house vs acquisition.
  - CMMI Maturity Level
  - Lifecycle model and languages used, use of commercial, military, or government off-the-shelf (COTS/MOTS/GOTS) software.
  - Use of Agile, PSP/TSP
- How does your organization train and develop software engineers and software quality engineers? Would you describe the 3 most beneficial classes for software engineers?
  - Organizational responsibility for development or acquisition of training curriculum
  - Type of training given: in-house vs external training programs (or combined)? What has led to using in-house vs external training?
  - Who has responsibility for identifying the training needs, how and when training is given?
  - Is training given at individual, group or project level, career levels? If so, what kind and when?
  - Is training mandatory or optional? How much time is allowed or expected for training per person?
  - How is mentoring and OJT included in developing individuals (informal, structured, required)?
  - What are your preferred methods and media for training?
  - Describe how your training program addresses proficiency training, system engineering, metrics, risk management and project management.
  - Describe any training provided to management (line management and/or project managers)?
  - How does your organization manage training that might be needed for a specific project (just in time training)?
- Do any of your projects include software acquisitions as a deliverable or as a piece of the complete software project? If so, could you describe your acquisition process, specifically how it integrates into your software organization?

Software Policies

- Please identify and summarize software policies, directives or requirements you have that are implemented organization-wide (governing documents)?
  - Specify the documents and their purposes.
  - Describe the level of detail of these documents.

  - Are industry or international standards (International Standards Organization (ISO), Institute of Electrical and Electronics Engineers (IEEE), FAA, military, etc.) used, how so?
  - Are these policies applicable to IT (infrastructure projects) and R&D projects? Are there any exclusions or tailoring?
  - Describe any waiver process, authority for approving waivers and the frequency of use.
- Identify the source of software policies, directives or requirements which are applied to your organization's software activities? Who (organizationally) is responsible for these high-level documents and how/when are they updated?
  - Who drafts, reviews and approves these documents and their updates?
  - How often are these documents updated; are updates affected by major software issues, lessons learned, external changes, internal changes?
  - If you do have to accommodate multiple sources of software policies, what is your approach/strategy for accommodating the policies that apply to a specific software activity?
- Describe how your organization ensures compliance with these governing documents?
  - Who is responsible for compliance? Are QA groups used to ensure compliance?
  - How and how frequently is compliance checked: interviews, surveys, audits, etc?
  - Are there checks/balances in software processes to help ensure compliance?
  - How are non-compliances handled? What is the follow-up to ensure findings do not recur?
  - Are governing documents updated based on findings?
- Please explain how these documents are communicated to the users?
  - Do you use a communication plan? If so, could you explain?
  - Is there any additional training developed to support any changes in the documents; is training used to ensure adequate understanding of the documents?
  - Describe any sort of mentoring program you have to support the implementation of these documents?
- How are policies and requirements included in contracted work (e.g., acquisitions)?
  - How do you ensure that software development, safety, reliability and QA requirements are included in contracts? Is there standard contract language to flow down the requirements?

    - Are there additional costs expected with acquisitions to include compliance with governing documents?
    - If compliance with governing documents is not required, how do you ensure quality products?
- How are project reviews and milestones coordinated with software reviews and milestones?
    - If reviews are independent, how does the project review address the impact of software dependencies? How and when do the project reviews integrate the software? How do software reviews address integration issues/impacts with the project?
    - What criteria are project reviews and software reviews based (schedule, readiness, combination)? Who and how are assessments of readiness made for reviews to occur? Who and how are decisions made to proceed to next level of development for life cycle and for formal reviews?

Rigorous Software Processes for Small Projects

- Please describe the scope (size and criticality) of small projects.
    - NASA develops flight software for small sized projects and for larger projects that have small software engineering needs. For this discussion, small projects at NASA include this type of work and typically requires five or less software engineers.
- Are small projects (all criticality levels) required to comply with software policies and requirements? If so, what are some of the key ways in which small projects are able to comply? If small projects are not required to comply, how are small projects governed?
    - Are exclusions or modifications outlined within the policies/requirements?
- Does your CMMI statement of work include or exclude small projects?
    - If you do apply CMMI to small projects, do you tailor your CMMI-compliant processes for small projects? Do you find some CMMI requirements (say at the ML2 level) are not feasible/practical for some/all of your small projects?
- Does your organization have any infrastructure to support a collection of small projects?
    - Do you have additional tools or alternate resources?
    - How is this infrastructure supported (cost and resources)?
- What methods or tools have you found that work well for small projects?
    - Specific areas of interest: requirements management, risk management, measurement, documentation, configuration management

    - What are some of the most useful process areas to focus on for small projects and why?
- How does your organization satisfy good software practices with limited resources and funds allocated to small projects?

Software Testing

- Please describe software testing: include strategy and scope, test plans, testing types, success/completion criteria.
    - When in the life cycle do you start development of your software test plan(s)?
    - When in the life cycle do you start development of your detailed test procedures?
    - At what point in the life cycle does the application of your test plan(s)/procedures start? (i.e., the development is done, a feature is complete, etc.)
    - Do you have any institutional guidelines that are used to define your test strategy? (i.e., percentage of code coverage required during software testing, stress test time requirements, test requirements for different criticalities of software, etc.)
    - Do you use any predictive software defect data to assess software test adequacy?
    - How do you plan or layout your nominal software test timeline or schedule?
    - Do you have any institutional guidelines for testing software models and simulations?
    - What software test metrics do you use?
    - Do you have software assurance witness or review the results from formal software testing?
    - What are your criteria for test completion, success?
    - When in the software life cycle do you begin capturing defect data?
    - Do you a approach or strategy for testing commercial operating systems? autocode, open source, GOTS/MOTS/COTS?
- What is the organization and composition of your typical software test team?
    - Does it include users and/or operations persons?
    - Is it independent from the software developers?
    - If using an Independent V&V group, how is this organized and integrated into the process?
- Please identify any tools used or autonomous testing performed?

- Do you use automated testing techniques? If yes, are separate resources used for generating the automated tests? (i.e., converting detailed test procedures into automated scripts.)
- Do you use any static test tools and can you comment on their usefulness?

CMMI Maturity Level

- Who decided and how did your organization decide to ascertain your current maturity level?
  - Is your current maturity level the organizational goal? If not, what maturity level is your organizational goal and when do you expect to achieve it?
  - What milestones did you establish to achieve your goal maturity level and how were those determined?
  - Have there been any assessments to determine if a current level is sufficient or cost-effective?
  - What amount of support has management provided for pursuit of higher maturity levels?
  - What were the most useful techniques/strategies in implementing the CMMI practices? (training, mentoring, outside consultants, etc. ?)
  - How do you maintain your maturity level?
- How were impacts to policies, requirements, training and other organizational structure handled?
  - Were necessary changes developed in transition plans?
  - How was the investment in these changes managed (organizationally, resource, costs)?
  - What was the extent (schedule and resources) of the investment?
- Have you been able to measure or identify any benefits at previous and current maturity levels? What are your top 3 areas of improvement and what impact did they have?
  - What are some of the benefits that your organization is realizing at your current maturity level?
  - How quickly did you see these benefits? While transitioning, once complete, shortly after full implementation?
  - Have the investments been able to reduce errors, testing (test plans, strategies, cost of testing), or some other measurable aspect of software development (personnel, resources, etc.)? How?
- How have you overcome major challenges that you have faced with pursuing maturity levels?
- How have you been able to reduce costs of appraisals?

For CMMI Maturity Level 4 and 5:

- Could you share some of your measurement programs currently in place?
  - How are they organized and managed?
  - How do you interpret and use these measurements?
  - How have you been able to fund and provide training for these programs?
  - Which have been the most useful measures?
- Does it cost less to operate at a higher maturity level or are other benefits more significant?

APPENDIX E – NASA SOFTWARE ENGINEERING HISTORY – POLICIES AND PROCEDURAL REQUIREMENTS

The early occurrence and recognition of software issues (software faults caused computer restarts during the Apollo 11 lunar landing, 1969) as well as the increasing costs of software development encouraged NASA to address the software engineering approaches used by the Agency and its suppliers. NASA's first policy document on software was *NASA Management Instruction (NMI) 2410.6, NASA Software Management Requirements for Flight Projects*, 1979. A 1982 independent study conducted by MITRE Corporation[9] found *NMI 2410.6* to be too narrow in scope, brief, and lacking life-cycle direction. In 1991, NASA replaced the prior NMI with a broader policy document in *NMI 2410.1*, NASA Software Management, Assurance, and Engineering Policy. Figure 4 shows the history of NASA software policies, through various consolidations and transfers of responsibilities from Headquarters OSMA, to Office of Chief Information Officer (OCIO), to the current owner the OCE.

[9] NASA Software Development and Assurance: Survey of Problems and Practices, F. Mayo, F. G. Tompkins, D. L. Hill, NASA-CR-175502, (MITRE Corp.), Nov, 1982.

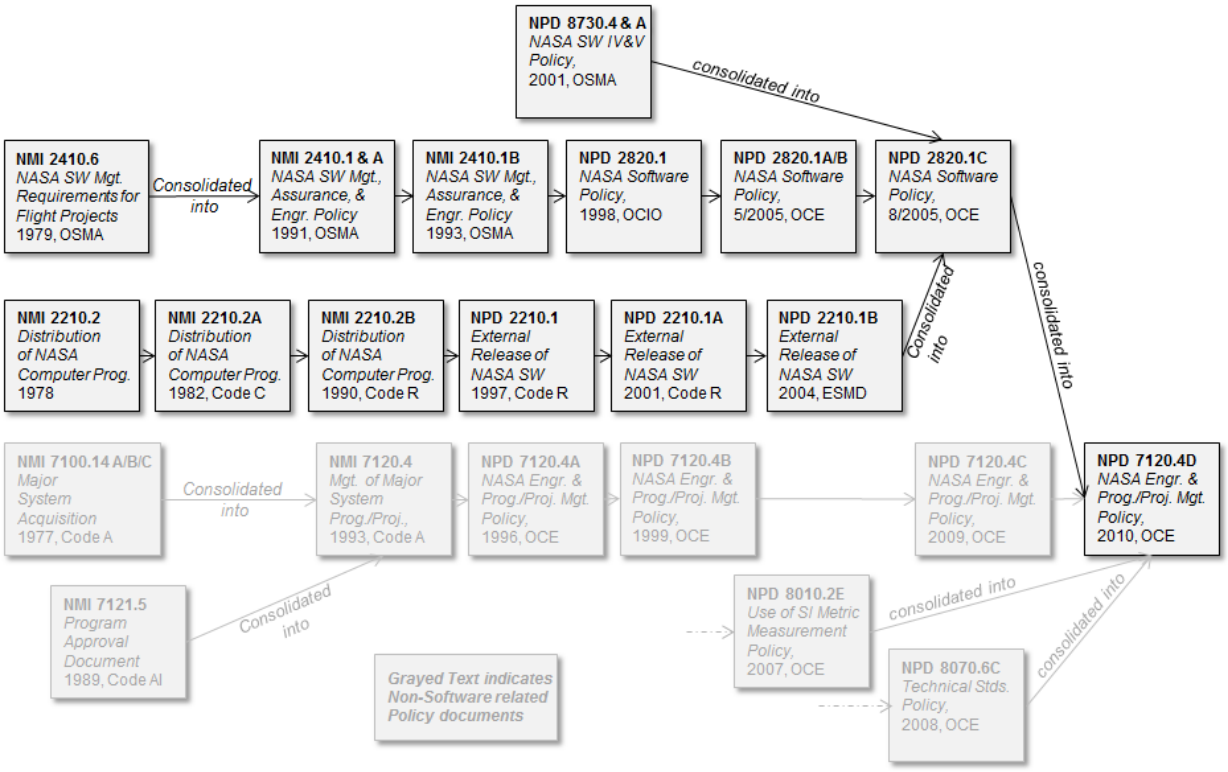


Figure 4: History of NASA's Software Policies

In addition to NASA policy, there are a number of software related NASA Procedural Requirements (NPR), Standards, Handbooks, and Guidebooks. Early NASA handbooks (NHB) included *NHB 2410.1A, Management Procedures for Automatic Data Processing Equipment*, 1970 and *NHB 2411.1, Computer Program Documentation Guidelines*, 1971. In 1976, the first NASA Software Engineering Workshop was hosted by NASA Goddard Space Flight Center, University of Maryland, and Computer Sciences Corporation to address software issues. The resulting Software Engineering Laboratory among these three participating representative government, industry, and academic entities made a number of contributions to the field of software engineering for over three decades including a series of guidebooks on software development. In the late 1980's NASA established the Software Management and Assurance Program (SMAP) which developed a set of document item description (DIDs) for software development and assurance. In 2002, The NASA Office of Chief Engineer, formed the NASA Software Engineering Initiative and chartered the NASA Software Working Group (SWG) to, "advance software engineering practices to effectively meet the scientific and

technological objectives of NASA". The NASA SWG produced the first Agency -wide procedural requirements for software, *NPR 7150.2*, NASA Software Procedural Requirements in 2004. The current NASA software documentation tree is provided in Figure 5.

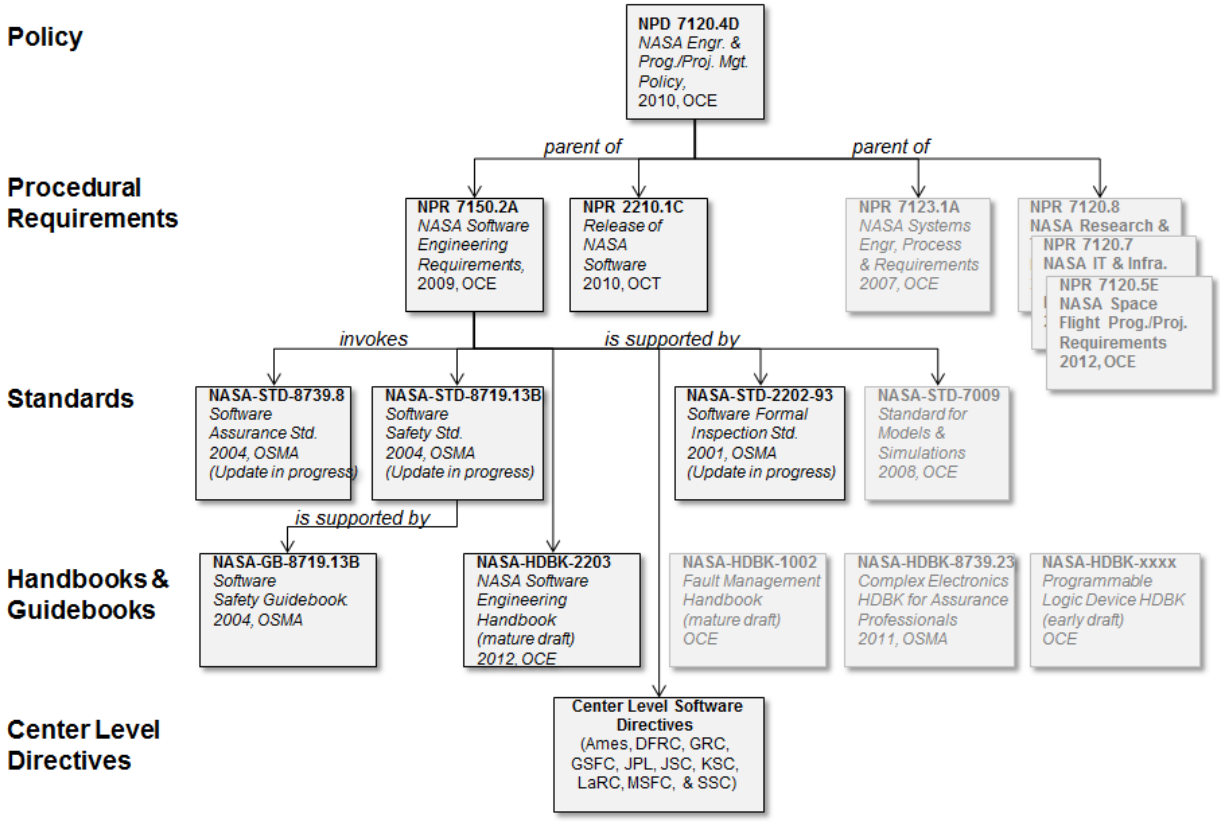


Figure 5: Current NASA Document Tree

APPENDIX F – SOFTWARE TOOLS IDENTIFIED

The following software tools were mentioned by participants during the NASA Software Engineering Benchmark discussions. This table should not be considered all-inclusive, nor is it an endorsement of any particular tool by NASA. The purpose of this table is to provide a list of the tools reported to be used by participants in the software engineering benchmark study.

Tool Name	Description
Bugzilla	Issue tracking software.
CodeCollaborator™	A code review tool.
CodeSonar®	A static analysis tool for C/C++ that detects bugs in critical embedded code that other source code analysis tools miss.
Confluence®	A collaboration tool.
Coverity Prevent™	This suite enables organizations to establish and enforce consistent standards for quality and security, across their internal teams and third-party software suppliers, and automatically test the code against those policies.
Cruise Control™	A Java-based framework for a continuous build process.
CURE	COTS Usage Risk Evaluation. Evaluates commercial of-the-shelf (COTS) software.
IBM® Rational® DOORS®	Dynamic Object-oriented Requirements System. A requirements management and traceability tool.
Doxygn	A C++ Source Code Documentation System.
E-room™ inspection	A peer review tool.

FindBugs™	A static code analysis tool.
Fortify® 360	Provides vulnerability assessments and application security solutions to help detect, test, prioritize, remove, remediate, and prevent vulnerability issues in software, whether applications are developed in-house, acquired from open sources, or procured from third-party vendors.
gCORE	A unit test coverage tool.
Gcov	A code coverage tool.
IBM® Rational® Rhapsody®	A collaborative design and development tool for systems engineers and software developers creating real time or embedded systems and software.
JIRA®	A bug tracking, issue tracking, and project management tool. JIRA tool has been very useful on small projects to track the task, cost estimates, inspections, etc.
Klocwork®	A source code analysis tool using static analysis and complete codebase inspection.
LabVIEW™	Laboratory Virtual Instrumentation Engineering Workbench. A system design platform and development environment for a visual programming language.
LDRA Testbed®	Liverpool Data Research Associates. This tool provides the core static and dynamic analysis engines for both host and embedded software.
LiquidPlanner®	A scheduling tool that allows fuzzy durations. Changing a duration or sequence of tasks, doesn't cause major effort in the tool (unlike Microsoft Project). It also syncs to iPhone, Android, online, etc.

Mantis Bug Tracker	An issue tracking tool.
MathWorks®	Mathematical computing software. Its major products include MATLAB and Simulink.
McCabe Software	Software quality, testing, security and configuration management tools
Microsoft Project®	A scheduling tool.
NASA software classification	A tool used to help define NASA software classification categories.
MathWorks ® Polyspace®	A static code analysis tool.
Process MAX	A project management tool for the development of business processes.
IBM® Rational® Purify®	A memory debugger program used by software developers to detect memory access errors in programs, especially those written in C or C++.
IBM® Rational® ClearCase®	A software configuration management (SCM) tool with version control.
RiskTrak™	Manages all forms of business risk on a project, program, or enterprise level.
SCRUB	Source Code Review User Browser. A tool for code reviews, brings together inspections and results of static analysis tools.
SEER- SEM™	A tool used for software cost estimation modeling.
Selenium	This suite of tools automates browsers. Primarily it is for automating web applications for testing purposes.
Sikuli	Visual technology to automate and test graphical user interfaces (GUI) using images (screenshots).
SPIN	A general tool for verifying the correctness of

	distributed software models in a rigorous and mostly automated fashion.
Splint	Secure Programming Lint. A programming tool for statically checking C programs for security vulnerabilities and coding mistakes.
STAX	An open source Eclipse Public License (EPL) project that enables users to create cross-platform, distributed software test environments.
Subversion (SVN)	An open source version control system.
Trac	An open source web-based project management and bug tracking system.
VectorCAST™	An integrated software test solution that reduces the time, effort, and cost associated with testing C/C++ software components necessary for validating safety- and mission-critical embedded systems.
Wind River Simics™ (formerly Virtutech)	A simulation of hardware boards that helps software engineers develop embedded software prior to actually having the boards.
Wind River VxWorks™ RTOS	Real-time operating system and tools.
Wiki	A tool used for documentation, development and reviews.

APPENDIX G – NASA PERSONNEL ON INTERVIEW TEAMS

Software Interviewers:

Heather Rarick/OCE/JSC (Benchmark Co-Lead) (Report Co-Author)
Sara (Sally) Godfrey/OCE/GSFC (Benchmark Co-Lead) (Report Co-Author)
John Kelly/OCE (Report Co-Author)
Robert (Tim) Crumbley/MSFC (Report Co-Author)
Kevin Carmichael/GRC
Elizabeth Strassner/JSC
Daryl Peltier/JSC
Patricia Benson/MSFC
Helen Housch/MSFC
Helen (Leann) Thomas/MSFC
Scott Morgan/JPL
William Van Dalsem/ARC
Laura Maynard-Nelson/GRC

Software Assurance Interviewers:

Martha Wetherholt/OSMA
Joel Wilf/JPL (Report Co-Author)
Cindy Naiman/NSC
Rosalynne (Roz) Strickland/MSFC
Susan Sekira/GSFC
Cyrus Chow/ARC
Renee Hugger/JSC
Cynthia Calhoun/GRC

APPENDIX H – LIST OF SITES/DATES/TEAMS

Access Restricted

APPENDIX I – SUMMARIZED OBSERVATIONS

Access Restricted

APPENDIX J – REFERENCES

1. *NASA Base Performance Review*, Cheevon B. Lau, Oct 21, 2010.
2. *Top NASA Software Issues of 2010*, Joint Software Working Group / Mission software Steering Committee Meeting at NASA Plum Brook, August, 2010.
3. *How Does NASA Estimate Software Cost? Summary Findings and Recommendations,* J. Hihn, et.al, 2012.
4. *ISO/IEC TR 19759:2005 Software Engineering -- Guide to the Software Engineering Body of Knowledge* (SWEBOK).
5. *NASA Software Development and Assurance: Survey of Problems and Practices*, F. Mayo, F. G. Tompkins, D. L. Hill, NASA-CR-175502, (MITRE Corp.), Nov, 1982.
6. *NASA-HDBK-2203 Software Engineering Handbook*, NASA Office of the Chief Engineer, 2013 (https://swehb.nasa.gov).
7. *SAE AS9100C, "Quality Management Systems - Requirements for Aviation, Space and Defense Organizations"*, 2009.
8. *NPR 7150.2A, NASA Software Engineering Requirements*, NASA Office of the Chief Engineer, 2009.
9. *NASA-STD-8739.8, NASA Software Assurance Standard*, 2004.
10. *NASA STD 8719.13 (Rev B w/ Ch1 of 7/8/2004), NASA Software Safety Standard*, 2004.