Introduction to Software Engineering

ISWE

Public

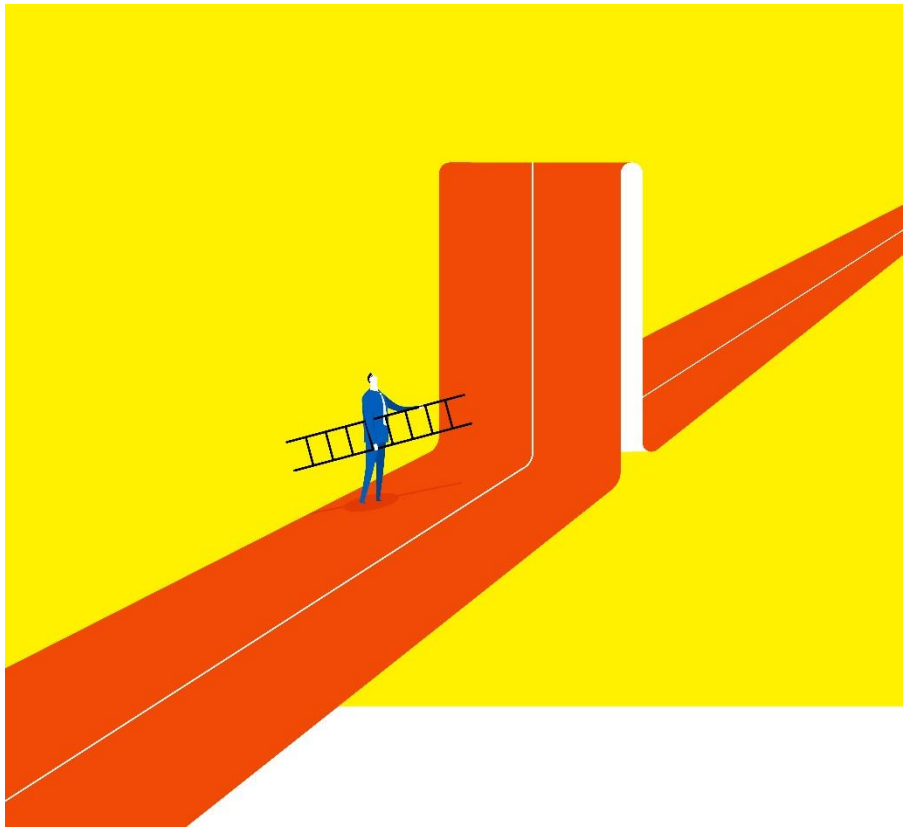
Introductions and Logistics

- Introductions of instructors
- Logistics
  - Sign-in sheet: Be sure to initial sheet everyday
  - Location of bathrooms, kitchen area
  - Exit in case of fire, etc.
  - Class evaluation process

Course Action Plan Slides

Training alone won't change performance

ISWE



Without monitoring, support and reinforcement, there's a chance that only a fraction of training is applied back on the job:

10% - 34%

(Brinkerhoff, 2006; Saks & Belacourt, 2006)

Developing new habits takes time.

Conventional Wisdom:
21 days

Research shows:
18-254 days; Average = 66 days

To change our behavior we need a system...

...like an Action Plan

Sources:

Brinkerhoff, R. O. (2006). *Telling training's story*. San Francisco, CA: Berrett-Koehler.

Lally, P. Van Jaarsveld, C. H. M., Potts, H. W. W., & Ardle, J. (2010). How habits are formed: Modelling habit formation in the real world. *European Journal of Social Psychology, 45*, pp. 998-1009.

Saks, A. M. & Belacourt, M. (2006). An investigation of training activities and transfer of training in organizations. *Human Resource Management*, *45*(4), 629-648

ISWE

Action Plans help you apply what you learn in order to improve performance

Before the Course	During the Course	After the Course
• Download the Action Plan template and example • Seek input from your project/task/branch manager • Draft preliminary Action Plan based on your expectations of what you will learn	• Create/refine your Action Plan as you learn things you can apply (you will be given a chance to do this during the course)	• Print and place your Action Plan in a conspicuous place • Inform an "accountability partner" about your Action Plan • Track your progress • Refine your Action Plan as needed

Introduction of Students

- Name
- Current Job Assignment
- What you want to get out of this class

Course High Level Objectives

- To provide an introduction to NASA software engineering skills
  - Not intended to be low level or “technical”
- To help non software engineers, system engineers and project managers understand the software development processes and considerations
- To help NASA engineers make better software related decisions by knowing where to get information and guidance

Key Course Objectives

Course Name	All Course Objectives	Key Course Objectives
APPEL - Introduction to Software Engineering	Upon completion of this course participants will be able to: 1. Explain software's role in and importance to NASA programs. 2. Properly interpret and apply NASA software engineering policies requirements templates tools checklists and guidelines. 3. Recognize and respond to early warning signs from software measurement data analysis and use results for effective decision making. 4. Formulate pertinent software measurements and reporting for senior management. 5. Explain the relationship between software development lifecycle phases and the project development lifecycle. 6. Identify the requirements for and the best practices of each phase in the software development lifecycle. 7. Describe methods to build good software products. 8. Describe the importance of software engineering support activities such as software configuration management software assurance software independent Verification and Validation software cost estimations software risks and software acquisition.	Upon completion of this course participants will be able to: 1. Properly interpret and apply NASA software engineering policies requirements templates tools checklists and guidelines. 2. Explain the relationship between software development lifecycle phases and the project development lifecycle. 3. Describe methods to build good software products. 4. Describe the importance of software engineering support activities such as software configuration management software assurance software independent Verification and Validation software cost estimations software risks and software acquisition.

Evaluation Pilot Courses Critical Behaviors

ISWE

Course Name	Key Course Objectives	Critical Behaviors
APPEL - Introduction to Software Engineering	Upon completion of this course participants will be able to: 1. Properly interpret and apply NASA software engineering policies requirements templates tools checklists and guidelines. 2. Explain the relationship between software development lifecycle phases and the project development lifecycle. 3. Describe methods to build good software products. 4. Describe the importance of software engineering support activities such as software configuration management software assurance software independent Verification and Validation software cost estimations software risks and software acquisition.	When they return to their jobs course attendees will: 1. Accurately interpret reported pertinent software measurements 2. Determine whether or not the software organization on their project is using the proper software requirements, and following the best practices of each phase in the software development lifecycle. 3. Determine if a software product is adequate 4. Assess if the tailoring options used on the software requirements is correct for the project risk level

Class Plan

Software's Role and Importance in NASA Missions

NASA Software Engineering & Assurance Policies, Requirements and Resources

Software Planning Requirements and Considerations

- Software Documentation
- Software Costing
- Software Processes
- Software Assurance
- Software Safety-Critical
- Software IV&V
- Software Classifications
- Software Reuse and Internal Sharing
- Software Cybersecurity
- Software Lifecycles and Reviews

Software Life-cycle Requirements

- Software Requirements
- Software Architecture
- Software Design
- Software Coding
- Software Testing
- Software Maintenance

Software Development Supporting Requirements

- Software Configuration Management
- Software Risks
- Software Peer Reviews
- Software Measurements
- Software Defect Management
- Software Bi-Directional Traceability
- Software License Management
- Software Acquisition

Why do we do these things? Software Failures

Software's Role and Importance in NASA Missions

Class Questions

- Can you name any examples of how software has affected your life (good or bad examples)?
- Do you think we can fulfill NASA's mission without software involvement?
- Why do you think software is important on NASA Missions?

Software's Role and Importance in NASA Missions

- The importance of software to NASA missions has grown steadily since NASA was formed.
- The first spacecraft launched by the United States in 1958 had no software at all, while the Mars Science Laboratory (MSL) launched in 2011 with well over 3 million lines of code.
- Contemporary NASA spacecraft have basically become flying computers.
- Software has become important on all NASA missions.
- Software percentage of a mission's budget ranges from 2% to 20%, with all missions needing high reliability software delivered on time and on budget.
- Flight software is typically the only item that can be changed or modified after launch
- Late or unreliable software threatens the entire mission, potentially causing launch delays and even mission failure.

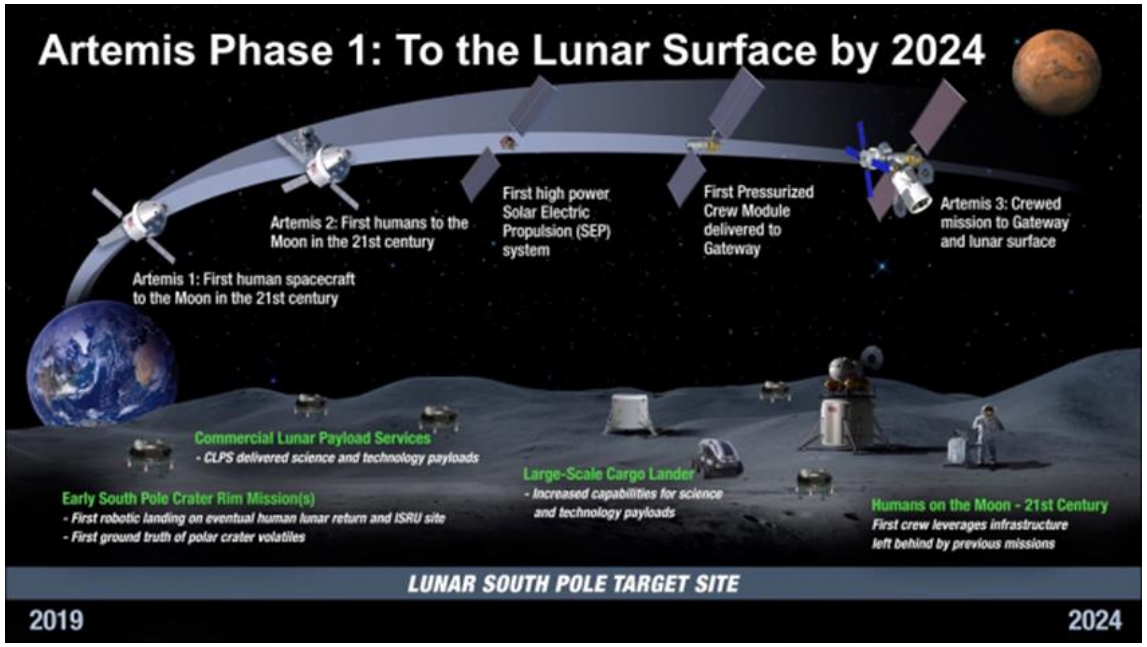
The result is that NASA is currently one of the 100 largest developers and procurers of software in the world.

NASA Software Workforce Trends

- More people are working software in 2021 than in previous years – **demand is high**
- OPM series is reporting less than are working software as reported by centers, and is **trending down, against the need**
  - This is due to software becoming more ubiquitous

Software Engineering Trends

- Space missions are increasingly dependent on correctly functioning software
- Software applications are growing in size and complexity
  - Rapidly increasing code size for all mission software
  - An increasing reliance on multi-threaded code
  - A gradual move from simpler to more complex languages
  - Increased reliance on COTS
- This brings two conflicting trends:
  - A growing importance of safe and reliable software
  - A shrinking ability to thoroughly test software
- This also leads to consistent underestimation of software development and assurance costs



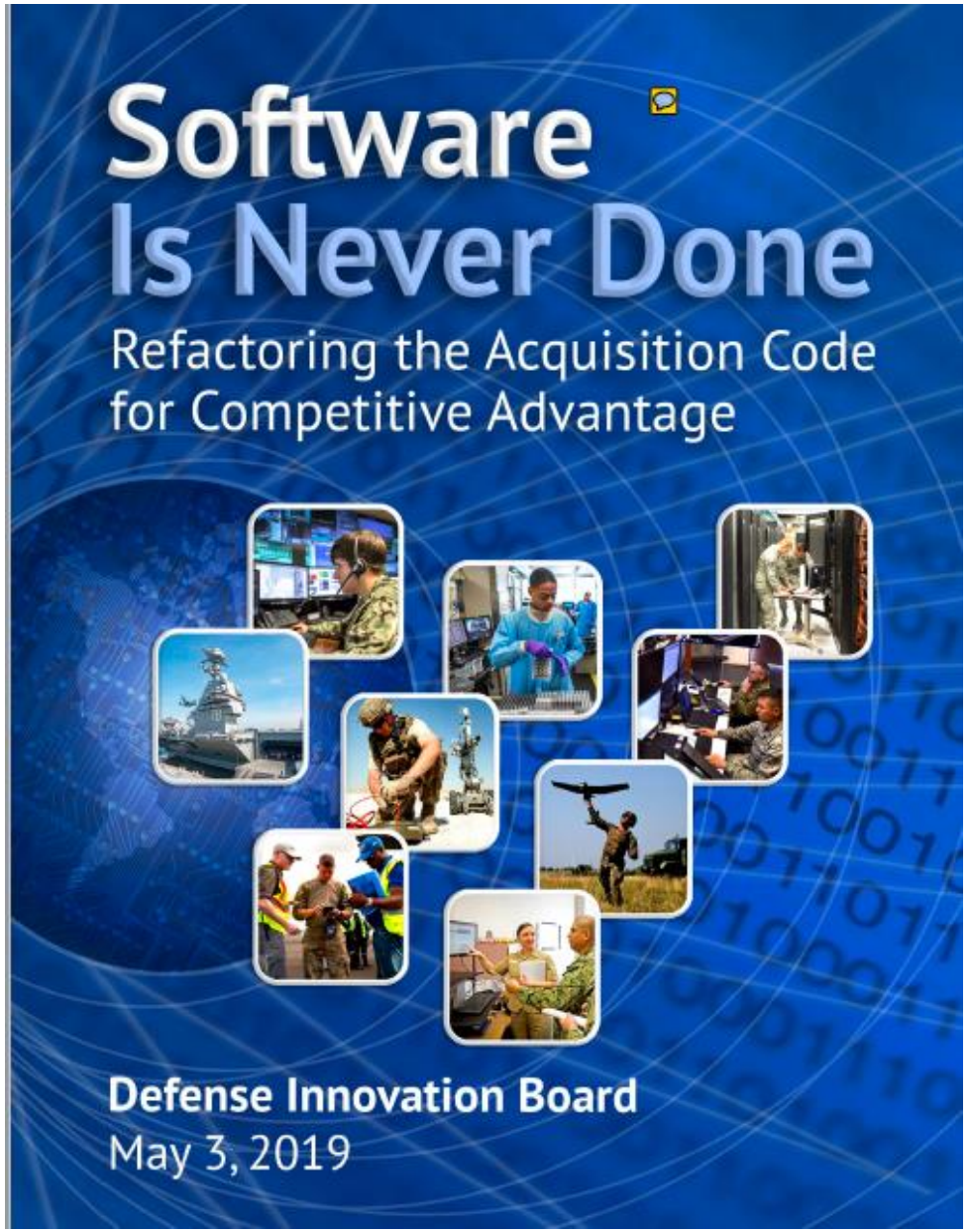
The increased demands placed on mission systems to implement NASA future mission portfolio will undoubtedly be answered in large part through functionality provided by software.

Software Engineering Capabilities Needed for Future Missions

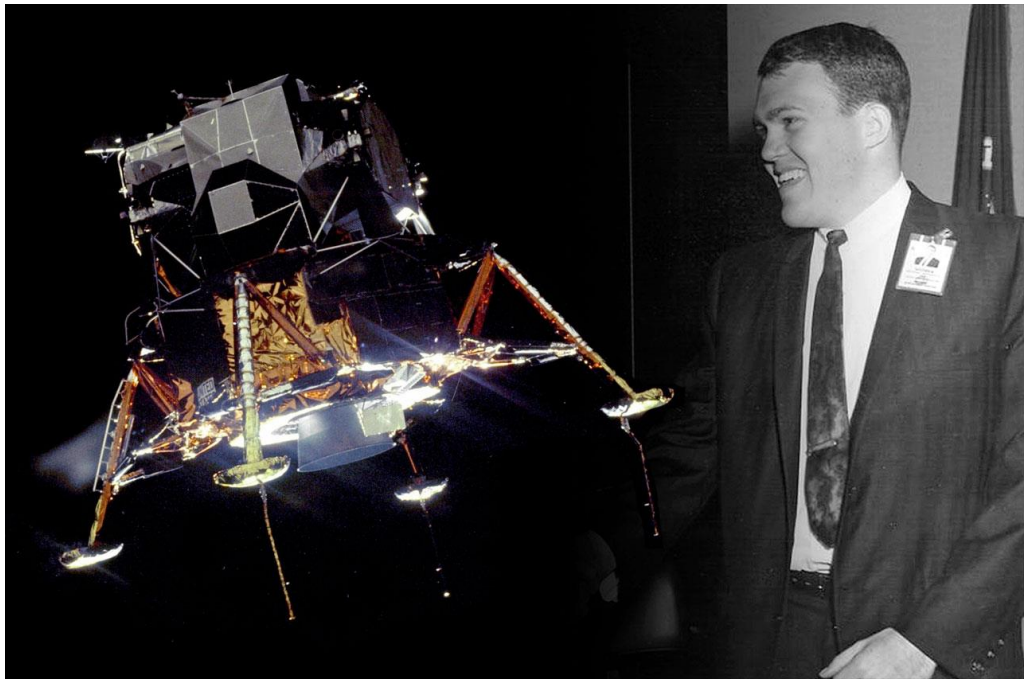
Future missions will require more software development and increased autonomous behavior in the software functions.

- More efficient and effective development and assurance practices required to meet the rapid increase in software size and complexity.
- Improved software acquisition practices (Make-Buy-Reuse).
- Maintaining a capable and well-trained workforce.
- Advancement in the design, development, verification and validation of autonomous behaviors
- Increased simulation capabilities
- Improved system design and requirements
- Determining metrics of software development effort and software product quality

What do you see as needed software capabilities for future launch systems?



“Software is different than hardware (and not all software is the same). Hardware can be developed, procured, and maintained in a linear fashion. Software is an enduring capability that must be supported and continuously improved throughout its life cycle.”

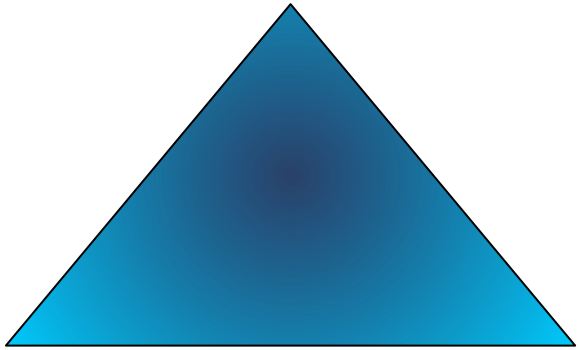


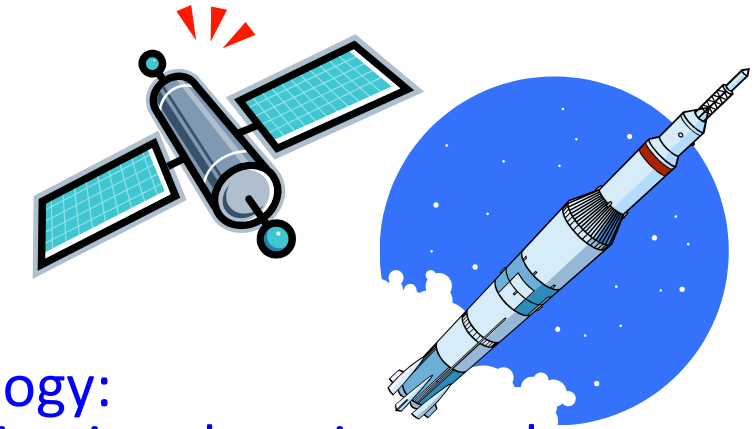
Software is the easiest to change .... but in change, it is the easiest to compromise.”

The "Bug" Heard 'Round the World by John R. “Jack” Garman October 1981

The Three Elements of Project Success

Processes and Requirements:
a defined method involving steps or operations





People:
Skills, Training, Management

Technology:
Application domains, tools, languages, information, environments

Improved Process + Competent Workforce + Appropriate Technology
=
Reduced Risk, Higher Productivity, and Better Quality

Catching Software Faults Early Saves Money

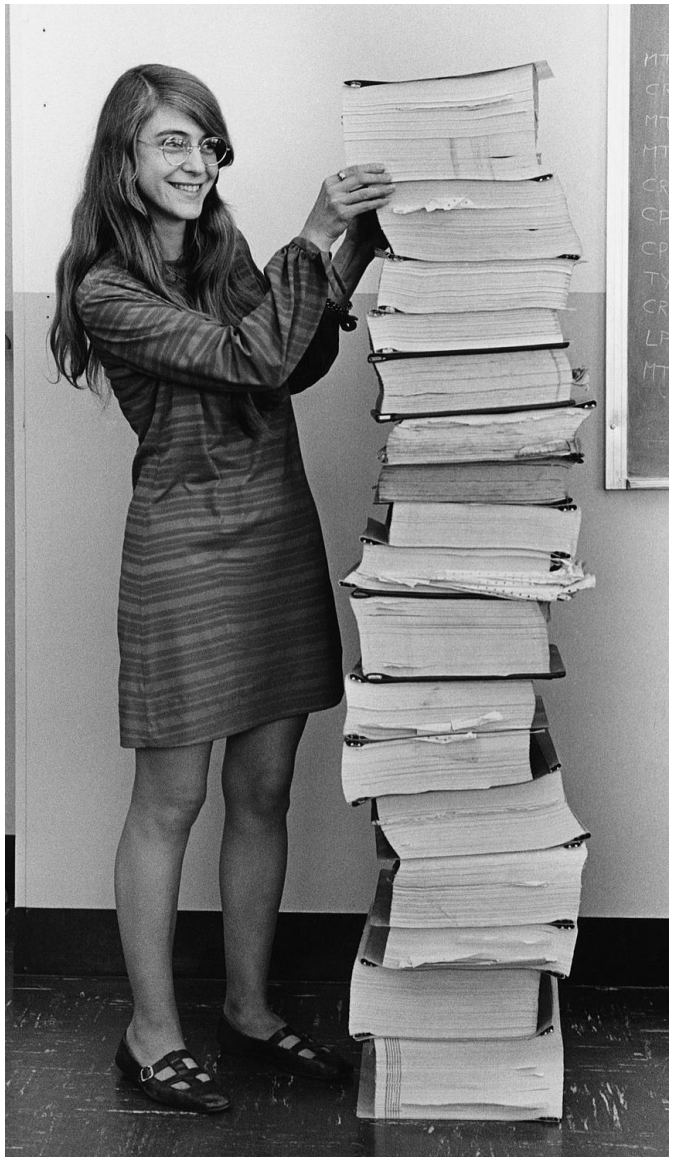
Faults accounts for 30-50% percent of total software project costs

What Is Software Engineering?

Software Engineering is not programming!

"a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software" IEEE

The term was coined by Margaret Hamilton in 1963-1964, director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for NASA's Apollo program.



"It was a memorable day when one of the most respected hardware gurus explained to everyone in a meeting that he agreed with me that the process of building software should also be considered an engineering discipline, just like with hardware." Margaret Hamilton

NASA's Software Definition (From IEEE)

Software is defined as:

(1) computer programs, procedures and possibly associated documentation and data pertaining to the operation of a computer system

(2) all or a part of the programs, procedures, rules, and associated documentation of an information processing system

(3) program or set of programs used to run a computer

(4) all or part of the programs which process or support the processing of digital information

(5) part of a product that is the computer program or the set of computer programs

This definition applies to:

- Software developed by NASA,
- Software developed for NASA,
- Software maintained by or for NASA,
- Commercial off-the-shelf (COTS) software,
- Government off-the-shelf (GOTS) software,
- Modified off-the-shelf (MOTS) software,
- Reused software,
- Auto-generated code,
- Embedded software,
- The software executed on processors embedded in programmable logic devices (see NASA-HDBK-4008, Programmable Logic Devices (PLD) Handbook),
- Legacy software,
- Heritage software,
- Application software,
- Open-source software components,
- Configuration Data

Software Is Not All the Same

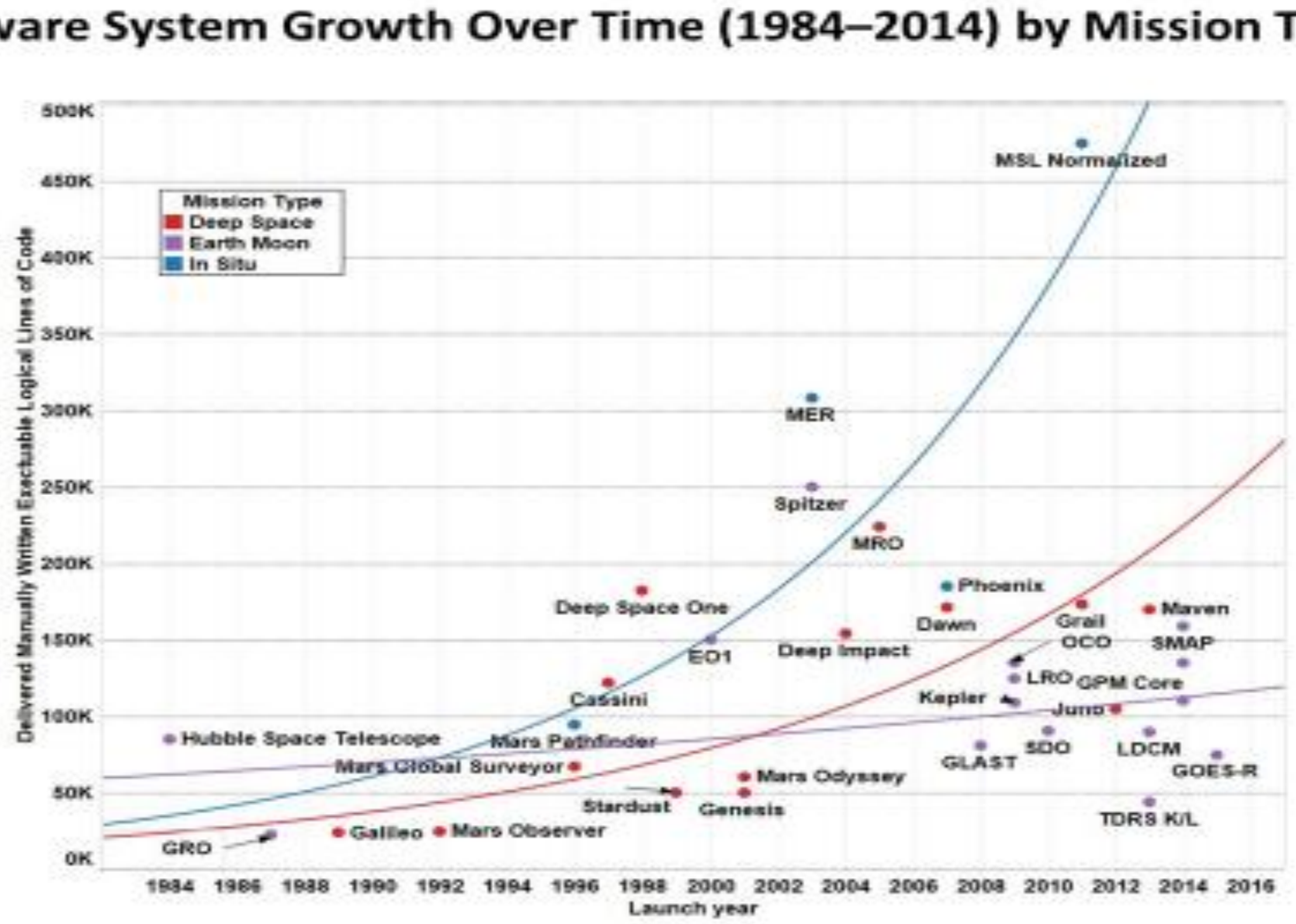
flight software ≠ *Non-flight software*

engineering software ≠ *general purpose software*

safety critical software ≠ *non-safety critical software*

… and it shouldn‘t be treated the same!

NASA flight software systems have grown as measured by SLOC



How Big is a Million Lines of Code?

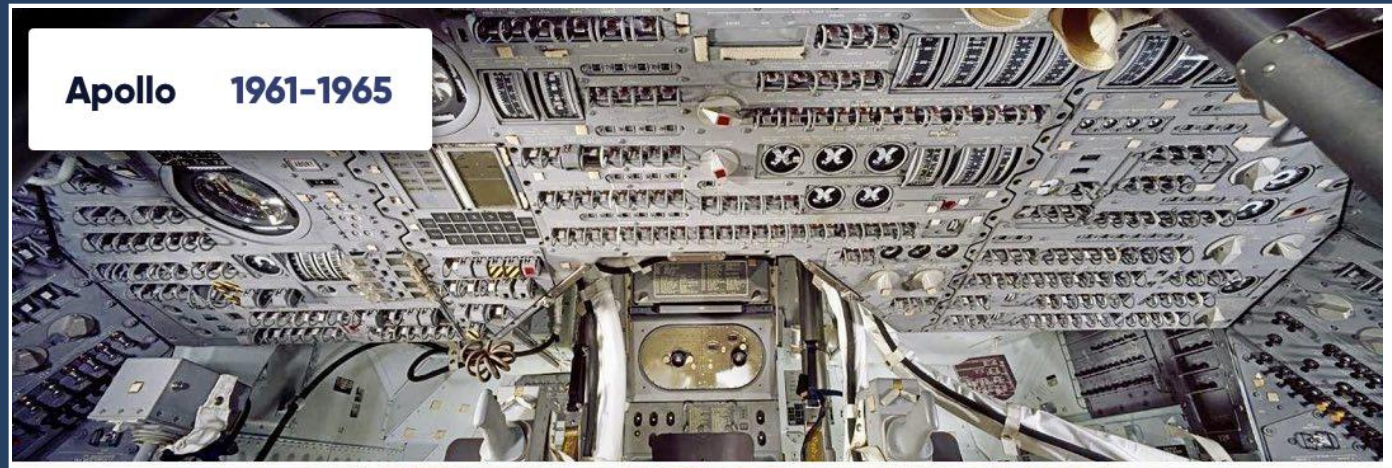
A novel has ~500K characters
(~100K words × ~5 characters/word)

~2.5 Million Lines of Code in the KSC GSDO program

~3.5 Million Lines of Code in the GSFC/Raytheon JPSS Core Ground System

A million-line program has ~20M characters
(1M lines × ~20 characters/line), or about 40 novels

Source:
Les Hatton, University of Kent, Encyclopedia of Software Engineering, John Marciniak, editor in chief



Increasing Complexity of Software

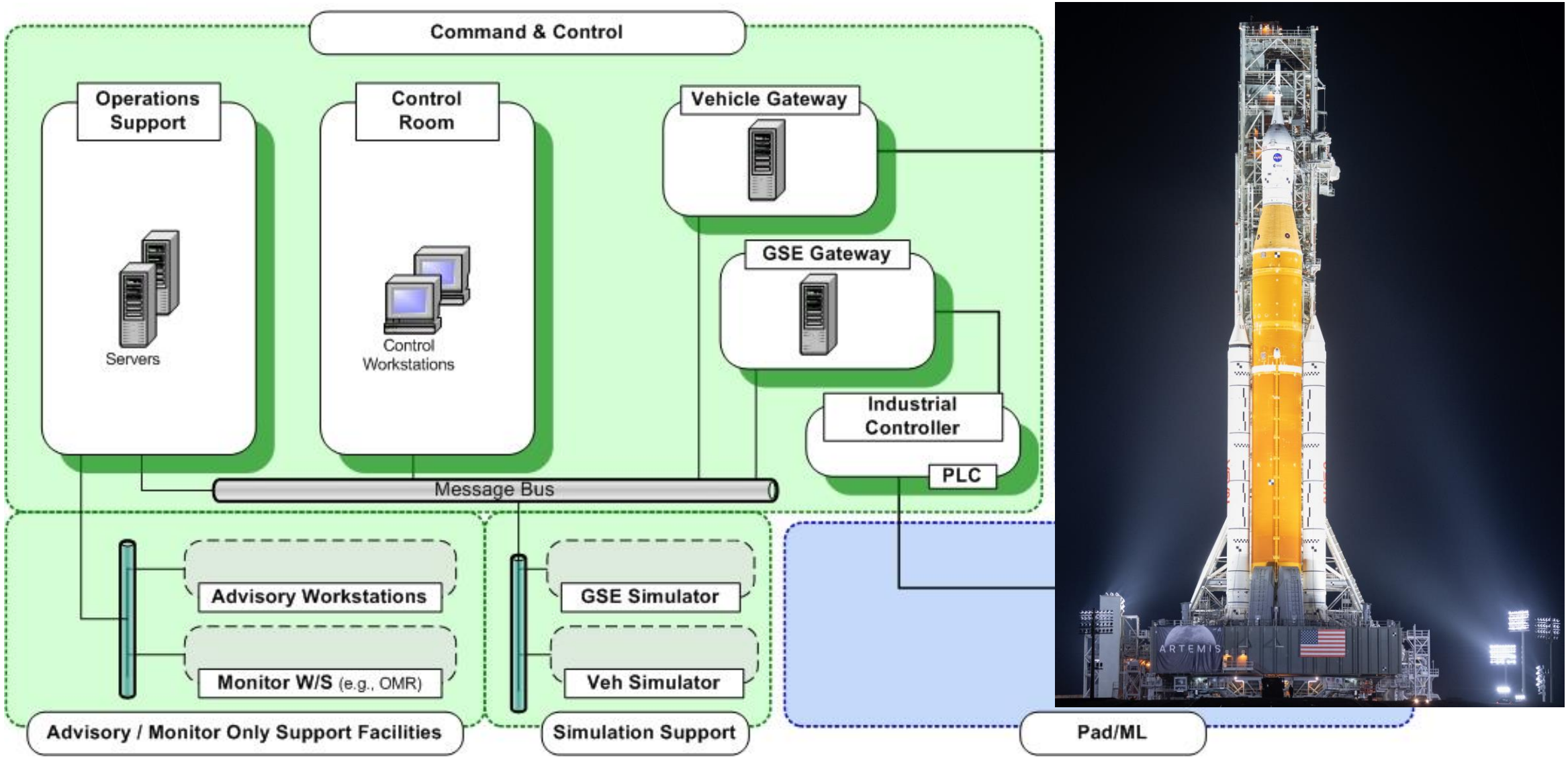
	KSLOCS
• Apollo	40
• Shuttle	440
• SLS	158
• EGS	1500
• Orion	1000+

What happened to the switches?

Other Types of Software Intensive Facilities and Operations

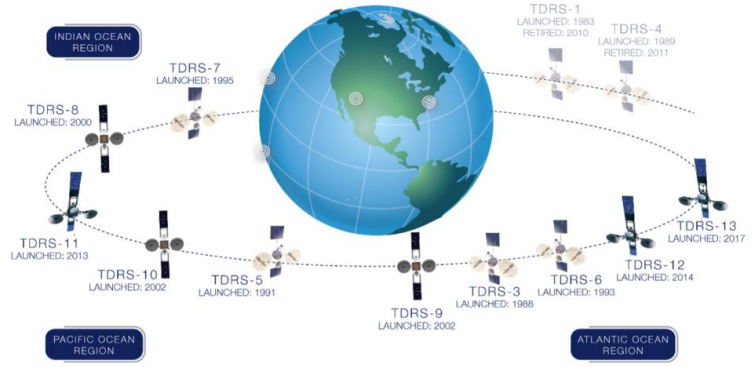
Spaceport Command and Control Systems

ISWE



Space-Ground Network Systems

- Major components of a space-ground network system include:
  - Antenna subsystem
  - Data processing equipment
    - Demodulates or modulates user data
    - Performs initial processing (synchronization, error detection and correction) and delivery to other Ground System Elements
  - Status/Scheduling subsystem
    - Provides means/mechanism to enable customer missions to schedule network services
    - Provides data quality and accounting information to customers
    - Not to be confused with mission planning and scheduling systems to control observatory operations and support science planning.



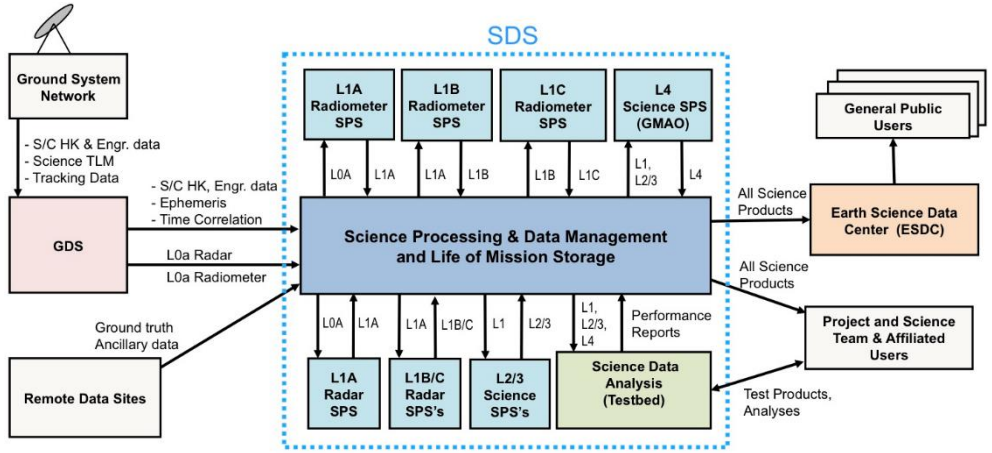
Mission Operations Centers

- Major components of a MOC include:
  - Real-time Telemetry and Command (RT T&C) subsystem
  - Mission Planning subsystem
  - Flight Dynamics subsystem/Attitude Ground subsystem
  - Trending subsystem
  - Automation/Alert subsystem
  - Data Storage and Distribution subsystem

Science Data Systems

- Science Data System Functionality/Architectures are generally unique from mission to mission, and heavily dependent upon the science objectives to be satisfied for the mission
- Functionality generally included in Science Data Support Systems include:
  - Data Ingest: Receipt of raw instrument data from the mission or other data suppliers
  - Generation of Science products: Mission unique depending upon the type of science being performed.
  - Data Archive/Distribution: Includes both Active Archives (To serve data products to Science community, other interested users) and Deep Archives (To preserve a copy of the science products beyond the nominal mission lifetime).
  - Provide other features required by science/user community, including:
    - Data mining
    - Modeling
    - Visualization/animation tools

- Functionality optionally included in Science Data Support Systems include:
  - Science/Instrument Operations Centers
    - Plan and schedule instrument operations, generate commands to control instrument observations/operations, assess health/safety of the instruments.
- Location of Science Data System architectural components very much unique from mission to mission

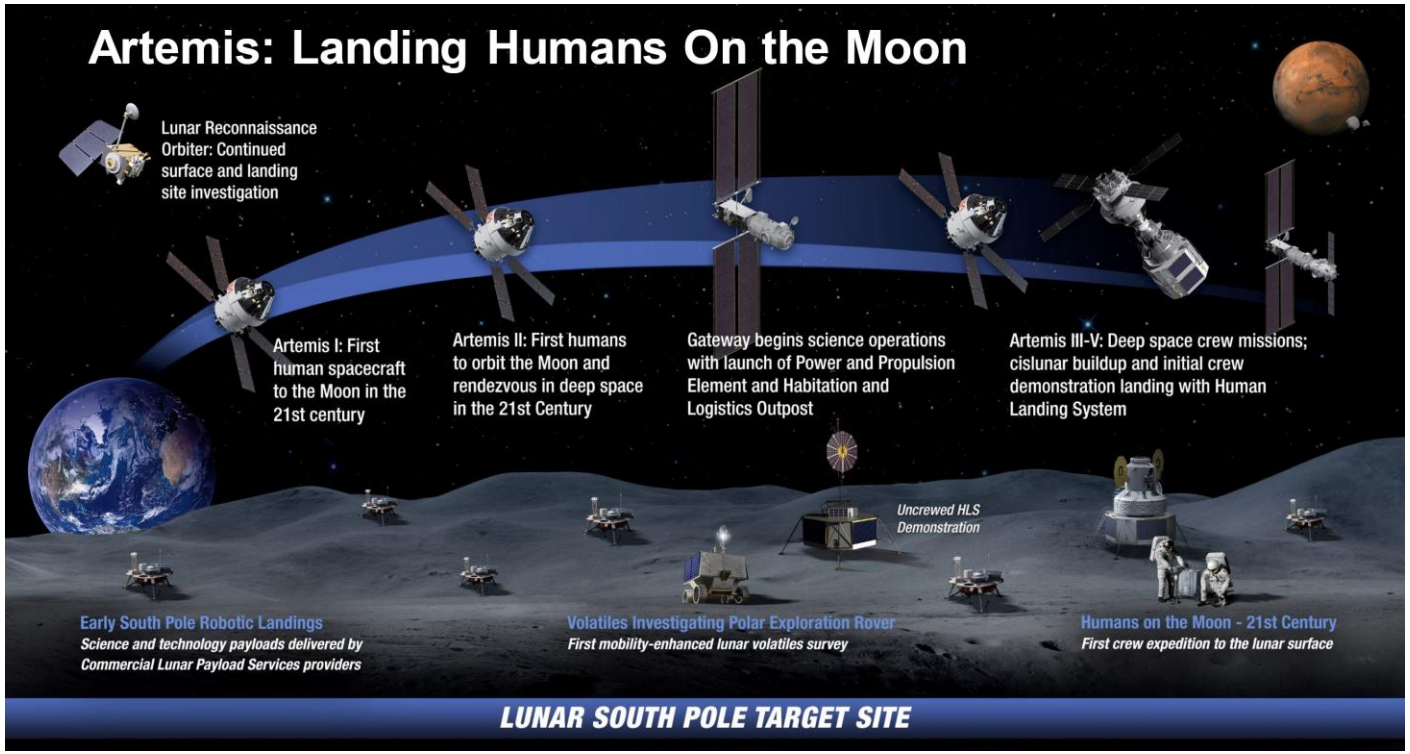


SPS – Science Production Software

The SMAP Science Data System (SDS) converts telemetry downloaded from the SMAP observatory into Science Data Products provided to the science community for research and applications.

Software's Role and Importance on NASA Missions

- **Software engineering and software assurance is a core capability** and a key enabling technology for NASA's missions and supporting infrastructure.
- All NASA missions have software involvement
- **NASA's success in increasingly dependent on software functions and capabilities.**
- NASA must become more efficient and effective in developing and validating quality software.



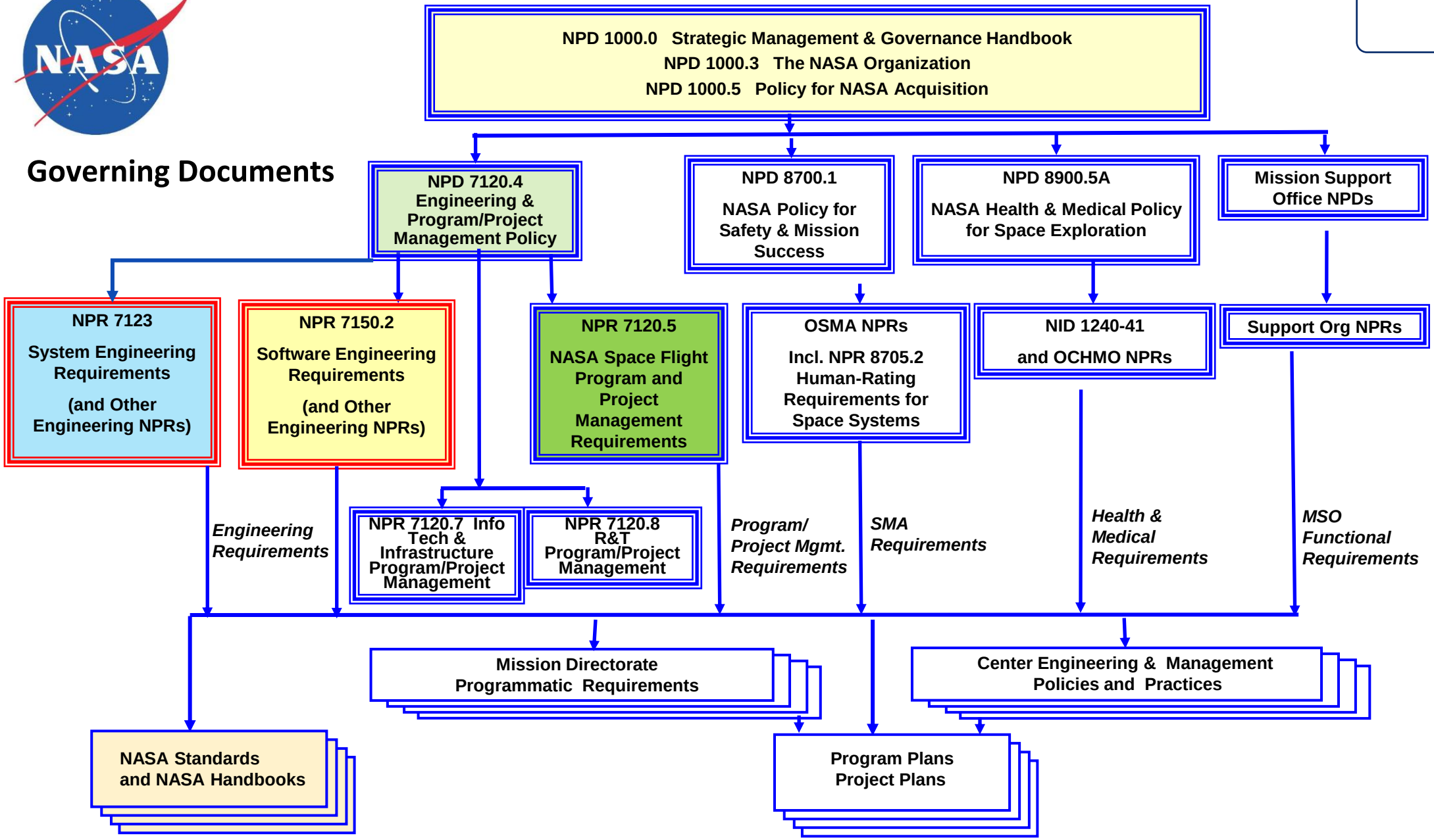
Future State

NASA missions will have more software, more complexity and more autonomous operations

We will need to invest in the software workforce to be able to support the NASA missions

NASA Engineering and Software Policies, including key NASA software standards

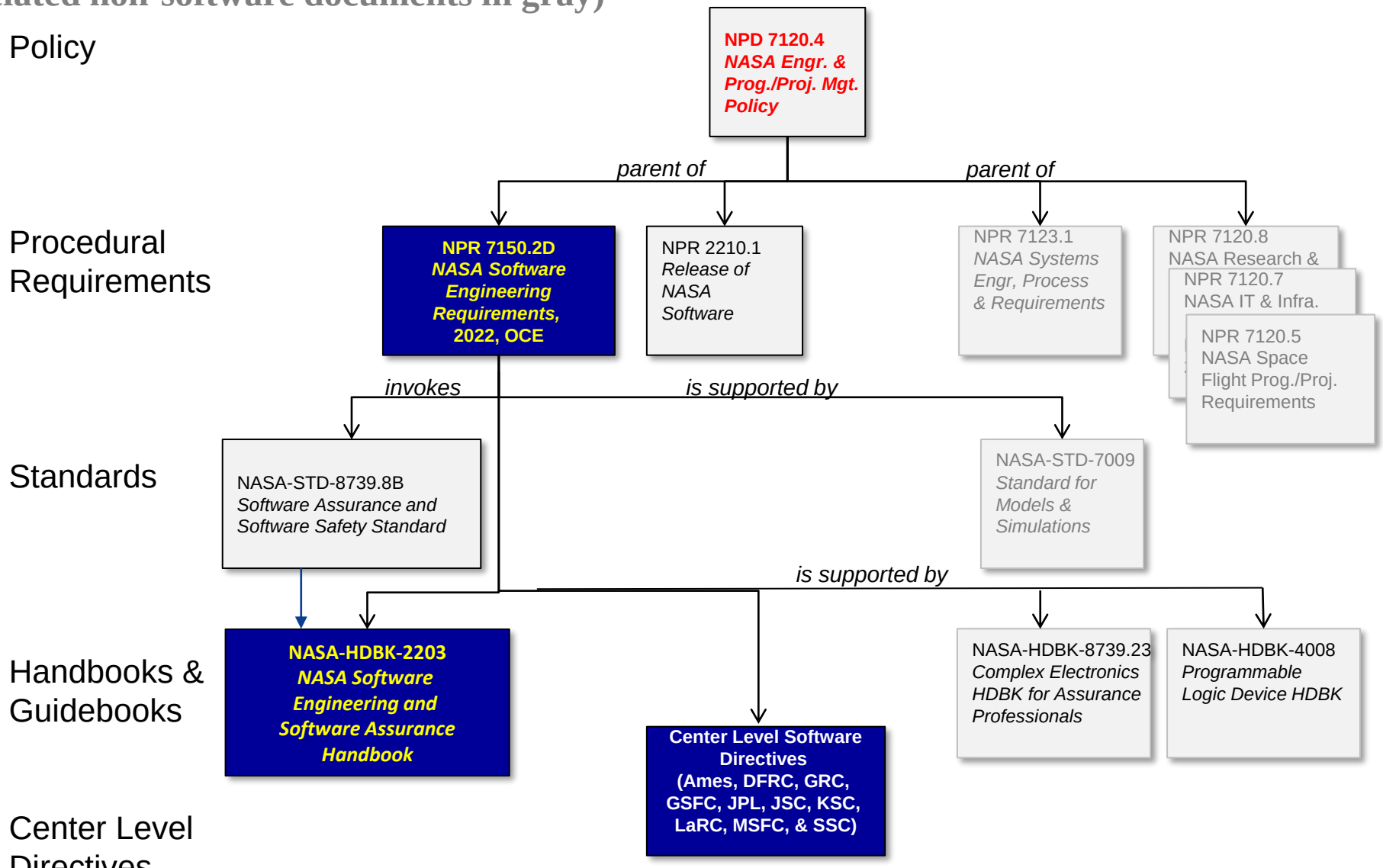
Governing Documents



Current NASA Software Documentation Tree

ISWE

(with a few related non-software documents in gray)

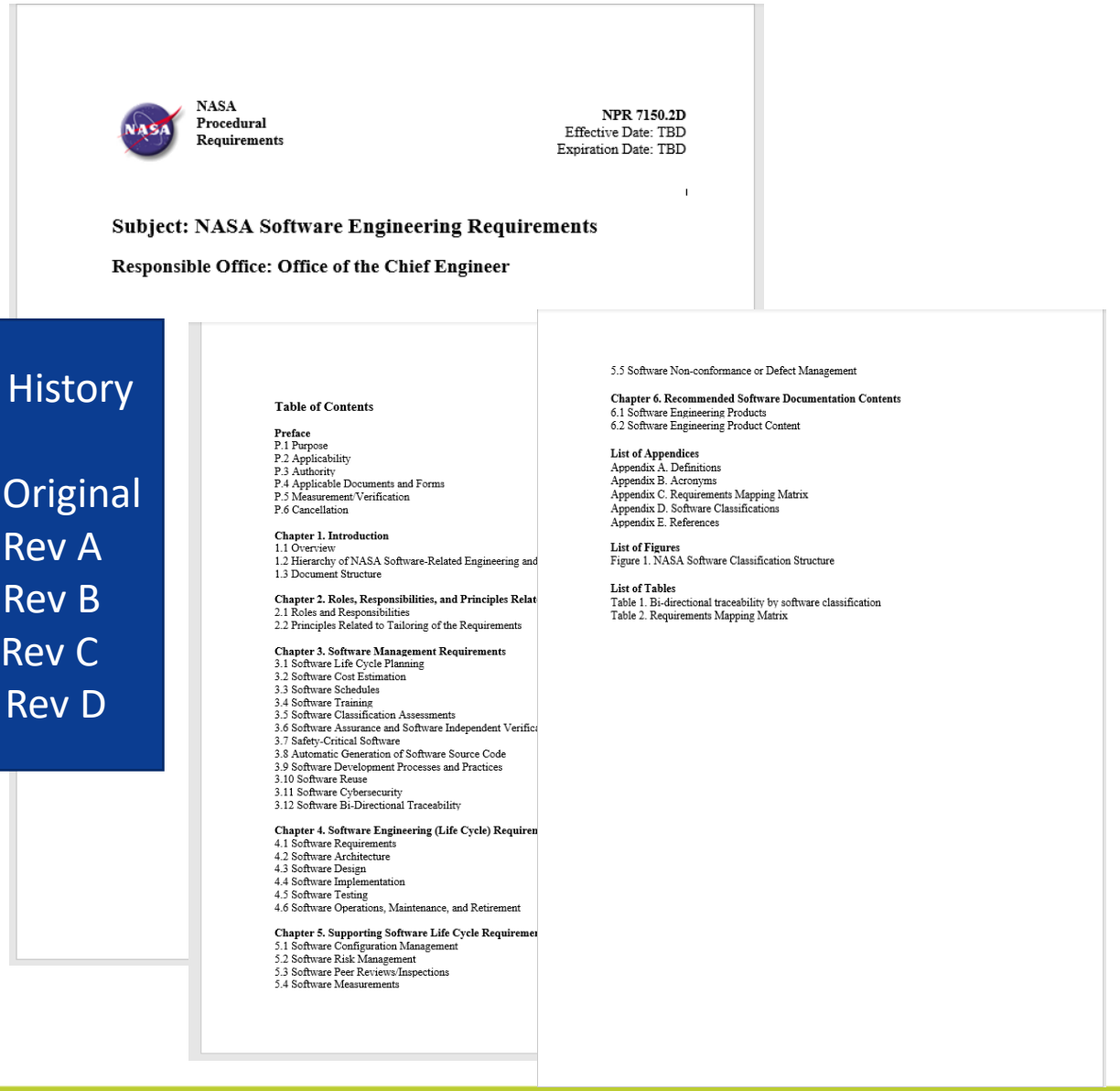


Purpose of the NASA Software Engineering Requirements, NPR 7150.2

- Software engineering is a core capability for NASA's missions and supporting infrastructure.
- Support the implementation of NASA's policies
- Provide a minimal set of requirements
- Support NASA programs and projects in accomplishing their planned goals

NPR 7150.2 History

Nov 2004 – Original
Nov 2009 – Rev A
Nov 2014 – Rev B
Aug 2019 – Rev C
Mar 2022 – Rev D



NASA Procedural Requirements

NPR 7150.2D
Effective Date: TBD
Expiration Date: TBD

Subject: NASA Software Engineering Requirements

Responsible Office: Office of the Chief Engineer

Table of Contents

Preface
P.1 Purpose
P.2 Applicability
P.3 Authority
P.4 Applicable Documents and Forms
P.5 Measurement/Verification
P.6 Cancellation

Chapter 1. Introduction
1.1 Overview
1.2 Hierarchy of NASA Software-Related Engineering and
1.3 Document Structure

Chapter 2. Roles, Responsibilities, and Principles Relat
2.1 Roles and Responsibilities
2.2 Principles Related to Tailoring of the Requirements

Chapter 3. Software Management Requirements
3.1 Software Life Cycle Planning
3.2 Software Cost Estimation
3.3 Software Schedules
3.4 Software Training
3.5 Software Classification Assessments
3.6 Software Assurance and Software Independent Verific
3.7 Safety-Critical Software
3.8 Automatic Generation of Software Source Code
3.9 Software Development Processes and Practices
3.10 Software Reuse
3.11 Software Cybersecurity
3.12 Software Bi-Directional Traceability

Chapter 4. Software Engineering (Life Cycle) Requiren
4.1 Software Requirements
4.2 Software Architecture
4.3 Software Design
4.4 Software Implementation
4.5 Software Testing
4.6 Software Operations, Maintenance, and Retirement

Chapter 5. Supporting Software Life Cycle Requiremer
5.1 Software Configuration Management
5.2 Software Risk Management
5.3 Software Peer Reviews/Inspections
5.4 Software Measurements
5.5 Software Non-conformance or Defect Management

Chapter 6. Recommended Software Documentation Contents
6.1 Software Engineering Products
6.2 Software Engineering Product Content

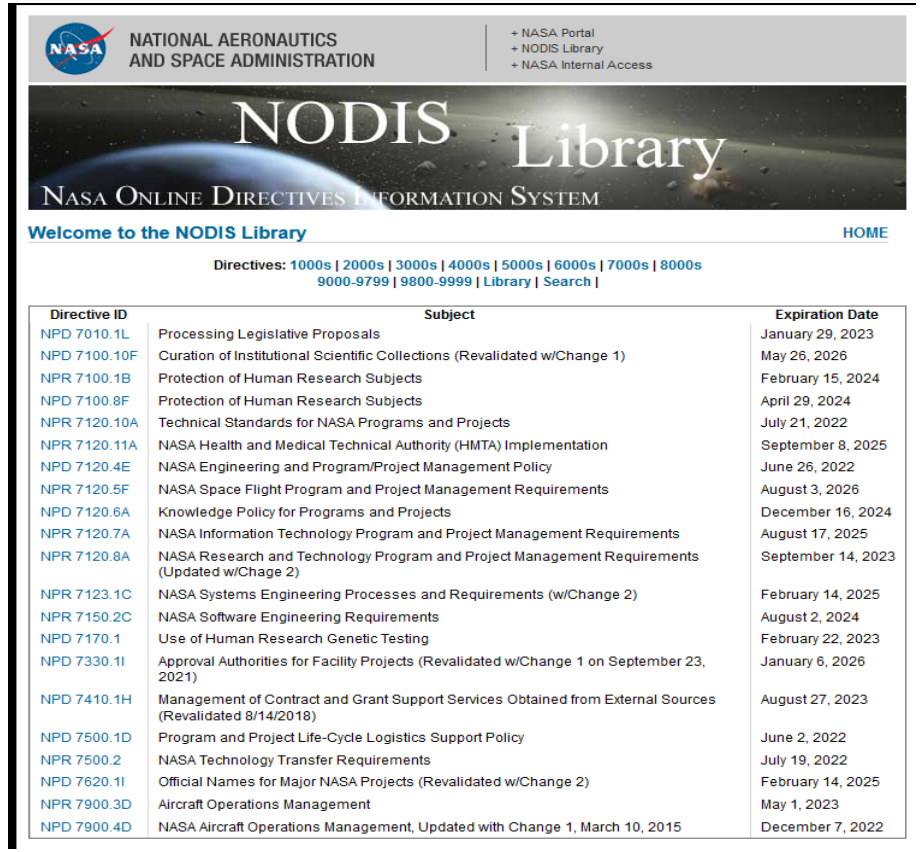
List of Appendices
Appendix A. Definitions
Appendix B. Acronyms
Appendix C. Requirements Mapping Matrix
Appendix D. Software Classifications
Appendix E. References

List of Figures
Figure 1. NASA Software Classification Structure

List of Tables
Table 1. Bi-directional traceability by software classification
Table 2. Requirements Mapping Matrix

About NASA's Software Engineering Requirements (NPR 7150.2)

- The NASA Office of the Chief Engineer is responsible for the NPR
- The NPR shall be applied to all software development, maintenance, operations, management, acquisition, and assurance activities
- Includes engineering and assurance requirements
- Requirements are levied on Center organizations as well as projects
  - Applicability of requirements is determined through the use of a NASA-wide definition of software classes
- To find the document online go to NASA Online Directives Information System (NODIS)
  - http://nodis3.gsfc.nasa.gov/main_lib.html
  - Look for NPR 7150.2

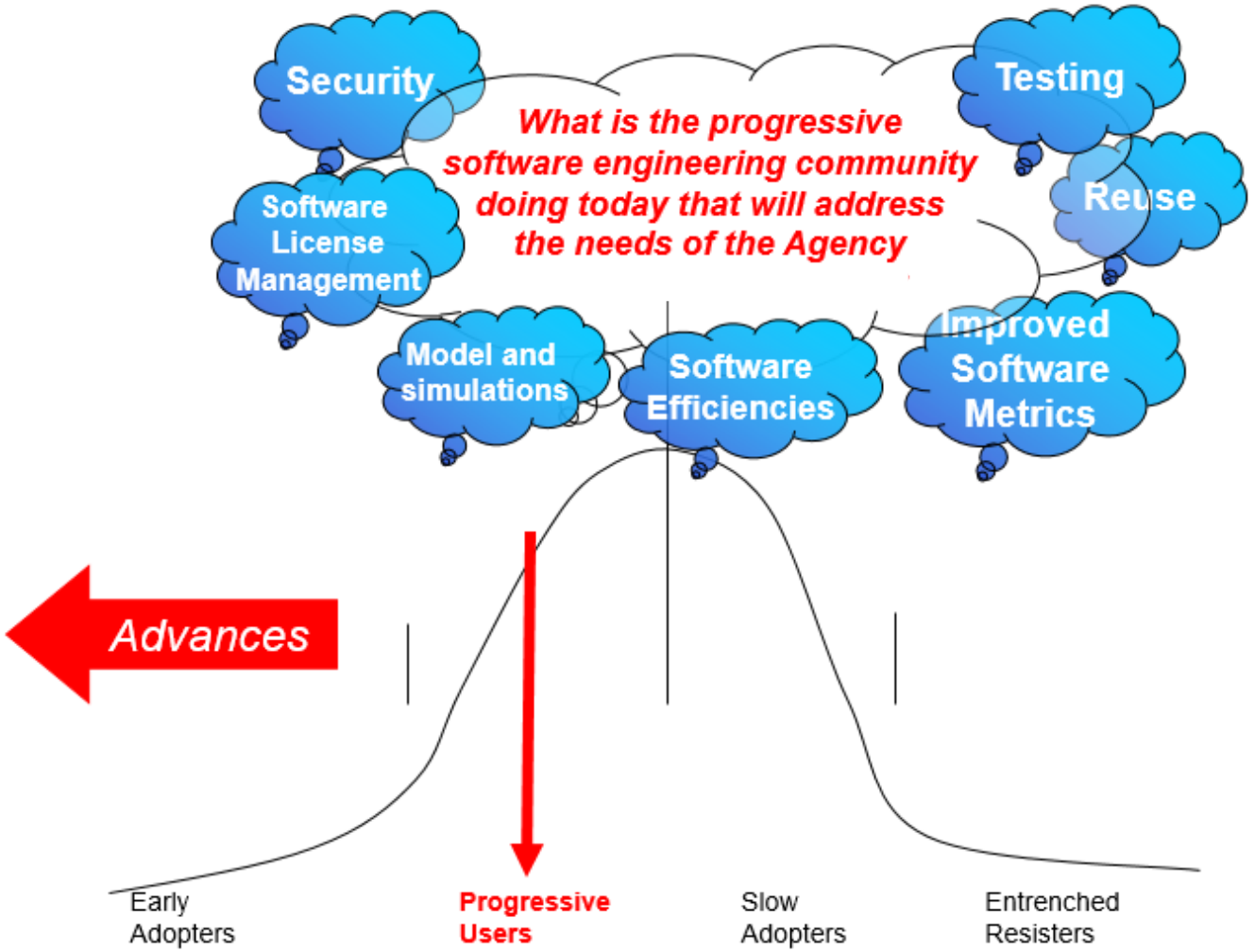


Directive ID	Subject	Expiration Date
NPD 7010.1L	Processing Legislative Proposals	January 29, 2023
NPD 7100.10F	Curation of Institutional Scientific Collections (Revalidated w/Change 1)	May 26, 2026
NPR 7100.1B	Protection of Human Research Subjects	February 15, 2024
NPD 7100.8F	Protection of Human Research Subjects	April 29, 2024
NPR 7120.10A	Technical Standards for NASA Programs and Projects	July 21, 2022
NPR 7120.11A	NASA Health and Medical Technical Authority (HMTA) Implementation	September 8, 2025
NPD 7120.4E	NASA Engineering and Program/Project Management Policy	June 26, 2022
NPR 7120.5F	NASA Space Flight Program and Project Management Requirements	August 3, 2026
NPD 7120.6A	Knowledge Policy for Programs and Projects	December 16, 2024
NPR 7120.7A	NASA Information Technology Program and Project Management Requirements	August 17, 2025
NPR 7120.8A	NASA Research and Technology Program and Project Management Requirements (Updated w/Chage 2)	September 14, 2023
NPR 7123.1C	NASA Systems Engineering Processes and Requirements (w/Change 2)	February 14, 2025
NPR 7150.2C	NASA Software Engineering Requirements	August 2, 2024
NPD 7170.1	Use of Human Research Genetic Testing	February 22, 2023
NPD 7330.1I	Approval Authorities for Facility Projects (Revalidated w/Change 1 on September 23, 2021)	January 6, 2026
NPD 7410.1H	Management of Contract and Grant Support Services Obtained from External Sources (Revalidated 8/14/2018)	August 27, 2023
NPD 7500.1D	Program and Project Life-Cycle Logistics Support Policy	June 2, 2022
NPR 7500.2	NASA Technology Transfer Requirements	July 19, 2022
NPD 7620.1I	Official Names for Major NASA Projects (Revalidated w/Change 2)	February 14, 2025
NPR 7900.3D	Aircraft Operations Management	May 1, 2023
NPD 7900.4D	NASA Aircraft Operations Management, Updated with Change 1, March 10, 2015	December 7, 2022

Recent update made to NPR 7150.2 for NPR 7150.2D

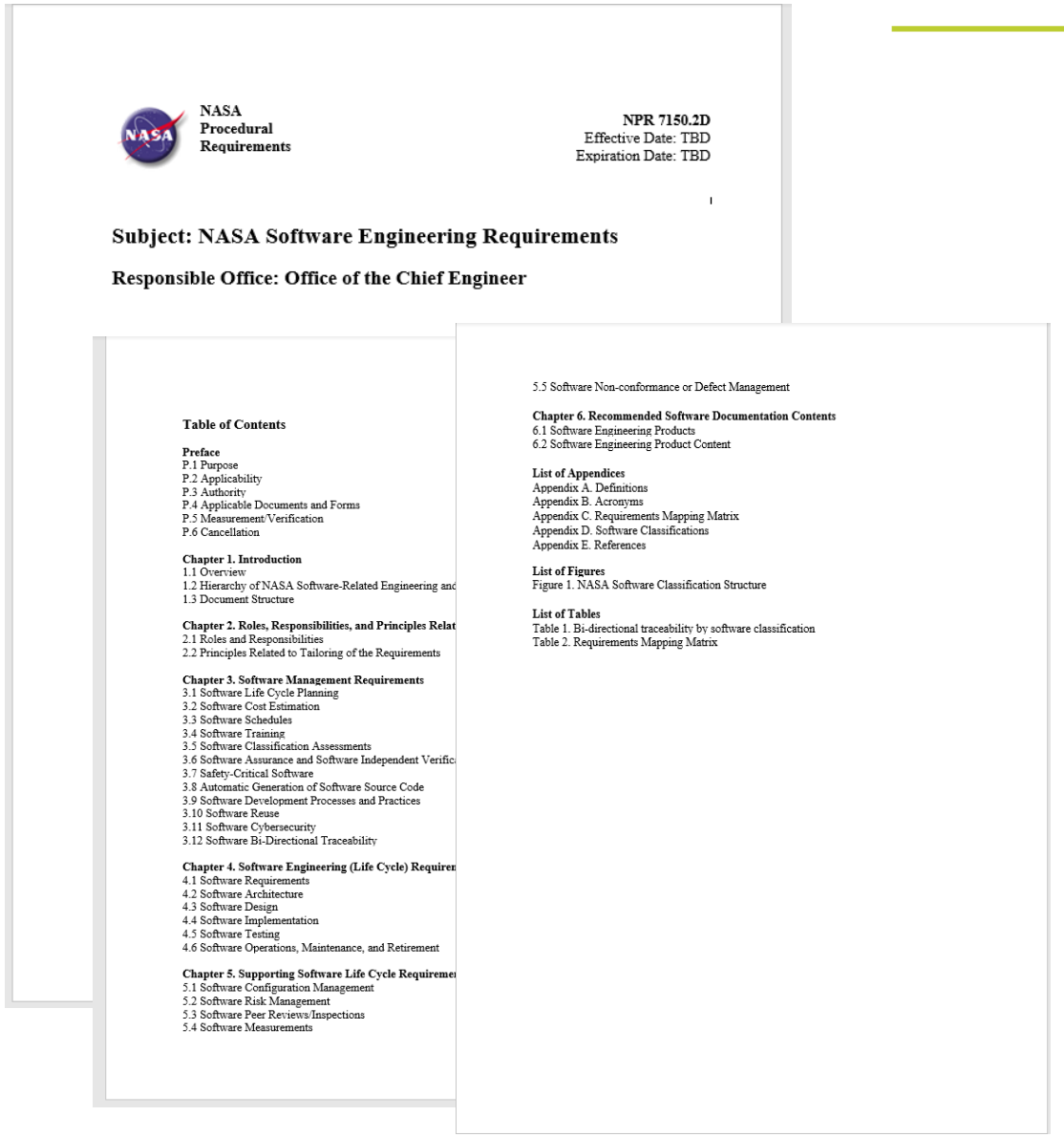
Update sources used:

- Inputs from across the Agency and NASA HQ
- Impacts on future missions
- OCE and OSMA surveys and audits
- Feedback from Projects
- Questions asked in the implementation of the NPR 7150 requirements
- Management Feedback
- Industry software standards
- Discussions with other engineering disciplines
- Program directions
- Studies of software



Themes and Targeted Change Areas for NASA Software Engineering Requirements

- Updated applicable documents and forms
- Added SWE requirements for SMA
  - Converted from “will” to “shall”
- Updated Tech authority wording
- Clarifications on Licensing and IP rights
- Addition of 100% code coverage for safety-critical software
- Addition of cyclomatic complexity for safety-critical software
- Adaptation of cybersecurity requirements
- Number of editorial fixes



NASA Procedural Requirements

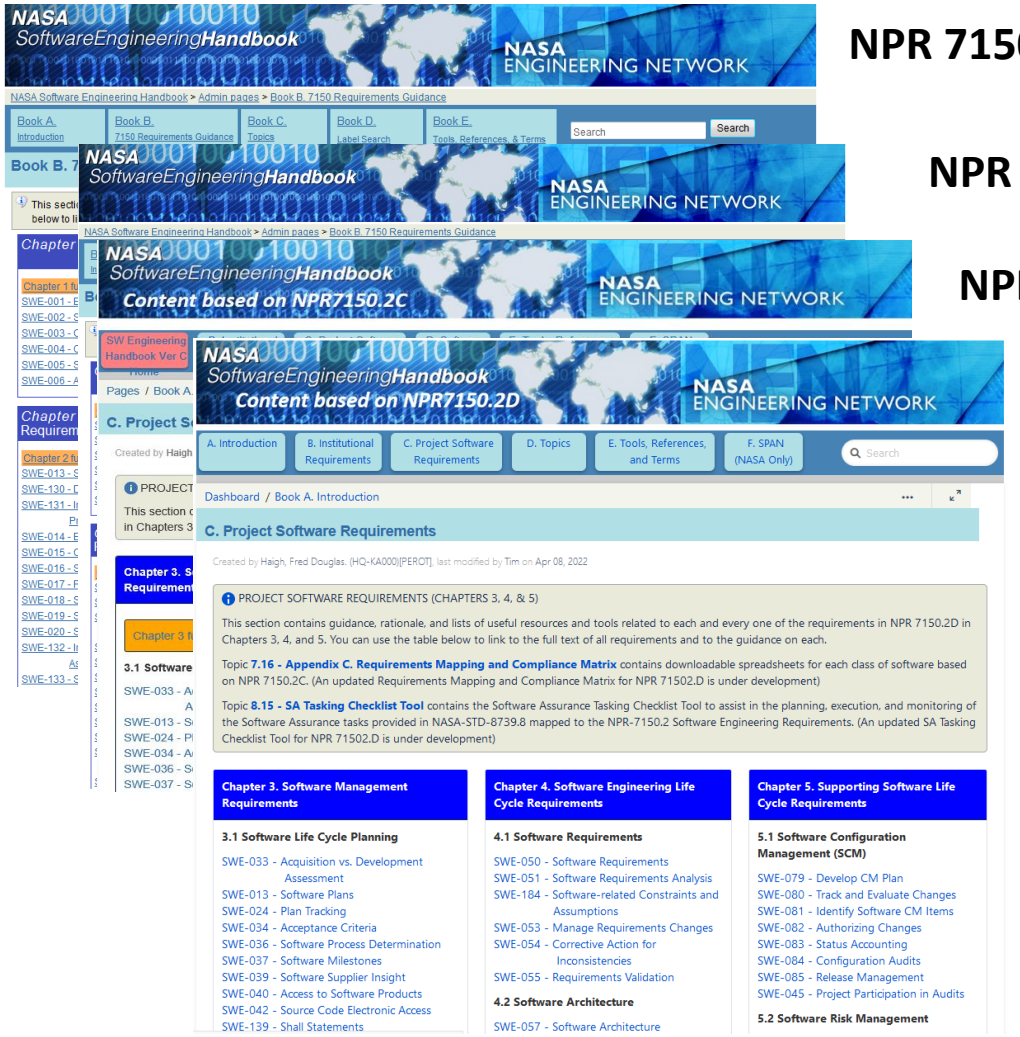
NPR 7150.2D
Effective Date: TBD
Expiration Date: TBD

Subject: NASA Software Engineering Requirements

Responsible Office: Office of the Chief Engineer

Table of Contents

Software Engineering Handbook



NPR 7150.2A

NPR 7150.2B

NPR 7150.2C

NPR 7150.2D

- Guidance material to help the NASA workforce implement the software engineering requirements in NPR 7150.2 and promote best practices across the Agency in software engineering.
- Addresses topics of interest identified by the Software Engineering community of practice
- Provides guidance for all of the software engineering requirements contained in NASA's NPR 7150.2, plus topics
- Guidance material includes requirement specific guidance, rationale, examples, best practices, lessons learned, references, tools and templates

https://swehb.nasa.gov/

Handbook Version Transition Page

Software Handbook –Project Requirements

ISWE

Remember…

- NPRs and Standards (including NPR 7150.2) are not intended to be "one size fits all documents"
  - They have built-in tailoring
    - Software Classification (Class A, B, C, D, E, or F)
    - Tailoring of the Software Classification requirements
  - There is a level of compliance and rigor specified that is associated with the class of the software to be built or acquired
  - *Part of your job as is to carefully consider what tailoring is necessary and build time into your schedule to complete it*
- There are tailoring procedures via Center and HQ Engineering Technical Authority (ETA)

Use good software engineering and software assurance judgement on which requirements should be implemented by your project

Summary

- The NPR provides a minimal set of requirements for software acquisition, development, maintenance, retirement, operations, and management
- The updated directive supports NASA programs and projects in accomplishing their planned goals (e.g., mission success, safety, schedule, and budget) while satisfying their specified requirements.
- The directive provides increased flexibility and tailoring options for software requirements for projects based on risk

Look at the software requirements and determine what you need to do for your project

Software Engineering Handbook Demo

https://swehb.nasa.gov/

Visual Overview of NPR 7150.2

30 "Institutional" Requirements (Chapter 2) Applicable to All Classifications

ISWE

OCE	SMA	Center Director/Delegate(s)		
Lead Software Engineering Initiave	Lead Software Assurance and Safety Initiative	Staff and advance software engineering capability	Measure for Improvement	Maintains contributor list
Benchmark Center's Capabilities against this NPR	Benchmark Center's SWA and SW Safety Capabilities	Establish and execute software processes	Establish and maintain software cost repo	Ensure Proper transfer of software
Benchmark Center Mapping Matrices	Review Center's Mapping Matrices	Comply with NPR per Classification in Appendix C	Contribute to Agency PAL (Process Asset Library)	Contract Officer: Ensure NPR is on contract
Authorize Compliance Appraisals	Authorize Appriasals against requirements	Report project status	Define content of SW documentation	Tech Authority: Assess against NPR
Provide Software Engineering Training	Provide Software Assurance Training	Maintain list of projects	Ensure Government rights to Software	OCE, SMA, OCIO: agree on tailoring
Maintain Process Asset Library (PAL)	Makes Decisions on Tailoring IVV Rqmt	Establish and maintain software Metrics	Ensure reuse software conforms to policies	Project Manager: Update plans per Classification

100 NPR Requirements* - *Applicable Based on Classification*

ISWE

Software Management (Chapter 3) | Lifecycle (Chapter 4) | Lifecycle Support-Ch5

Make/Buy	Tailor	Classify	Perform MC/DC	Verify Cyber Protection	Validate	Accredit Tools	Regression Test	Track Changes	Record Peer Review Results
Plan	Mapping to this NPR	Maintain Classification Records	Track Cyclomatic Complexity	Use Secure Coding	Architect	Plan, Report Tests	Test Safety Rqmts	Identify CM Items	Measure Software
Track Actual vs. Expected Plan	Establish and Acquire OTS	Plan SA & IVV	Plan Auto-Gen lifecycle	Use Cyber Static Analysis	Review Architecture	Test	Develop, Test Data Upload Procedures	Establish CM Procedures	Analyze Software Measurements
Determine Acceptance Criteria	Establish Cost	Ensure IVV	Receive Auto-Gen Supplier Inputs	Record Adversarial Actions	Design	Manage Configuration	Test Reuse/COTS Equally	Maintain CM Records	House Measurement Data
Determine Deliverables	Include Specific Cost Items	Ensure IVV Project Exec Plan (IPEP) if IVV	Perform and Certify as CMMI	Perform Bi-Directional Traceability	Implement, Code	Evaluate Test Results	Plan Ops, Maintenance, Retirement	Perform CM Audits	Compare Measured vs. Expected
Define Milestones	Store Cost in Repo	Provide IVV Artifacts	Identify Reuse Rqmts	Establish Rqmts	Adhere to Coding Standards	Use Accredited Tools	Deliver Products	Develop Release Procedures	Measure Software Volatility
Developer Report Status	Develop Schedule	Respond to IVV Findings	Evaluate Reusability	Map to System Rqmts	Perform Static Code Analysis	Update Plans	Complete Verification	Participate in Audits	Track Defects
Dev'er Provide Product & Metrics	Regularly Review with Stakeholders	Determine Safety Criticality	Assess Cyber	Include Safety Rqmts	Unit Test	Validate in High-fidelity	Maintain	Determine, Manage Risk	Determine Severity Levels
Developer to Provide Access to Source Code	Dev'ers Report Schedule	Adhere to 8739.8 SWA & SW Safety Std	Identify Cyber Risks	Track Rqmt Changes	Repeat Unit Test	Track Code Coverage Metrics	Archive	Peer Review Rqmts, Plans, Code, Test	Assess reuse, COTS defects
Comply with this NPR	Train	Do Safety-Crit items: SWE-134	Implement Cyber Protection	Track Corrective Actions	Develop VDD	Validate Metrics in Test	Plan CM	Follow Basic Peer Review Process	Assess Process Defects

*Note SWE-220 Cyclomatic Complexity has 2 shalls, counted as 1 here

Class A&B (All 100) C (92) D (64) E (12) Requirements

ISWE

Make/Buy	Tailor	Classify	Perform MC/DC	Verify Cyber Protection	Validate	Accredit Tools	Regression Test	Track Changes	Record Peer Review Results
Plan	Mapping to this NPR	Maintain Classification Records	Track Cyclomatic Complexity	Use Secure Coding	Architect	Plan, Report Tests	Test Safety Rqmts	Identify CM Items	Measure Software
Track Actuals vs. Expected Plan	Establish and Acquire OTS	Plan SA & IVV	Plan Auto-Gen lifecycle	Use Cyber Static Analysis	Review Architecture	Test	Develop, Test Data Upload Procedures	Establish CM Procedures	Analyze Software Measurements
Determine Acceptance Criteria	Establish Cost	Ensure IVV	Receive Auto-Gen Supplier Inputs	Record Adversarial Actions	Design	Manage Configuration	Test Reuse/COTS Equally	Maintain CM Records	House Measurement Data
Determine Deliverables	Include Specific Cost Items	Ensure IVV Project Exec Plan (IPEP) if IVV	Perform and Certify as CMMI	Perform Bi-Directional Traceability	Implement, Code	Evaluate Test Results	Plan Ops Maintenance, Retirement	Perform CM Audits	Compare Measured vs. Expected
Define Milestones	Store Cost in Repo	Provide IVV Artifacts	Identify Reuse Rqmts	Establish Rqmts	Adhere to Coding Standards	Use Accredited Tools	Deliver Product	Develop Release Procedures	Measure Software Volatility
Develop Report Status	Develop Schedule	Respond to IVV Findings	Evaluate Reusability	Map to System Rqmts	Perform Static Code Analysis	Update Plans	Complete Verification	Participate in Audits	Track Defects
Deliver Product & Metrics	Regularly Review with Stakeholders	Determine Safety Criticality	Assess Cyber	Include Safety Rqmts	Unit Test	Validate in High-fidelity	Maintain	Determine Manage Risk	Determine Severity Levels
Developer to Provide Access to Source Code	Delivers Report Schedule	Adhere to 8739.8 SWA & SW Safety Std	Identify Cyber Risks	Track Rqmt Changes	Repeat Unit Test	Track Code Coverage Metrics	Archive	Peer Review Rqmts, Plans, Code, Test	Assess reuse, COTS defects
Comply with this NPR	Train	Do Safety-Crit Items: SWE-134	Implement Cyber Protection	Track Corrective Actions	Develop VTD	Validate Metrics in Test	Plan CM	Follow Basic Peer Review Process	Assess Process Defects

Class F Requirement Applicability (OCIO Authority)

ISWE

Make/Buy	Tailor	Classify	Perform MC/DC	Verify Cyber Protection	Validate	Accredit Tools	Regression Test	Track Changes	Record Peer Review Results
Plan	Mapping to this NPR	Maintain Classification Records	Track Cyclomatic Complexity	Use Secure Coding	Architect	Plan, Report Tests	Test Safety Rqmts	Identify CM Items	Measure Software
Track Actual vs. Expected Plan	Establish and Acquire OTS	Plan SA & IV&V	Plan Auto-Gen lifecycle	Use Cyber Static Analysis	Review Architecture	Test	Develop, Test Data Upload Procedures	Establish CM Procedures	Analyze Software Measurements
Determine Acceptance Criteria	Establish Cost	Ensure IV&V	Receive Auto-Gen Supplier Inputs	Record Adversarial Actions	Design	Manage Configuration	Test Reuse/COTS Equality	Maintain CM Records	House Measurement Data
Determine Deliverables	Include Specific Cost Items	Ensure IV&V Project Exec Plan (IPEP) if IVV	Perform and Certify as CMMI	Perform Bi-Directional Traceability	Implement, Code	Evaluate Test Results	Plan Ops, Maintenance, Retirement	Perform CM Audits	Compare Measured vs. Expected
Define Milestones	Store Cost in Repo	Provide IV&V Artifacts	Identify Reuse Rqmts	Establish Rqmts	Adhere to Coding Standards	Use Accredited Tools	Deliver Products	Develop Release Procedures	Measure Software Volatility
Developer Report Status	Develop Schedule	Respond to IV&V Findings	Evaluate Reusability	Map to System Rqmts	Perform Static Code Analysis	Update Plans	Complete Verification	Participate in Audits	Track Defects
Dev'er Provide Product & Metrics	Regularly Review with Stakeholders	Determine Safety Criticality	Assess Cyber	Include Safety Rqmts	Unit Test	Validate in High-fidelity	Maintain	Determine, Manage Risk	Determine Severity Levels
Developer to Provide Access to Source Code	Dev'ers Report Schedule	Adhere to 8739.8 SWA & SW Safety Std	Identify Cyber Risks	Track Rqmt Changes	Repeat Unit Test	Track Code Coverage Metrics	Archive	Peer Review Rqmts, Plans, Code, Test	Assess reuse, COTS defects
Comply with this NPR	Train	Do Safety-Crit items SWE-134	Implement Cyber Protection	Track Corrective Actions	Develop VDD	Validate Metrics in Test	Plan CM	Follow Basic Peer Review Process	Assess Process Defects

Class Plan

ISWE

Software's Role and Importance in NASA Missions

NASA Software Engineering & Assurance Policies, Requirements and Resources

Software Planning Requirements and Considerations

- Software Documentation
- Software Costing
- Software Processes
- Software Assurance
- Software Safety-Critical
- Software IV&V
- Software Classifications
- Software Reuse and Internal Sharing
- Software Cybersecurity
- Software Lifecycles and Reviews

Software Life-cycle Requirements

- Software Requirements
- Software Architecture
- Software Design
- Software Coding
- Software Testing
- Software Maintenance

Software Development Supporting Requirements

- Software Configuration Management
- Software Risks
- Software Peer Reviews
- Software Measurements
- Software Defect Management
- Software Bi-Directional Traceability
- Software License Management
- Software Acquisition

Why do we do these things? Software Failures

Software Engineering Documentation

Key NPR requirements for documentation

- The project manager shall develop, maintain, and execute software plans, including security plans, that cover the entire software life cycle and, as a minimum, address the requirements of this directive with approved tailoring. [SWE-013]
- The project manager shall establish and maintain the software processes, software documentation plans, list of developed electronic products, deliverables, and list of tasks for the software development that are required for the project's software developers, as well as the action required (e.g., approval, review) of the Government upon receipt of each of the deliverables. [SWE-036]
- Where approved, the project manager shall document and reflect the tailored requirement in the plans or procedures controlling the development, acquisition, and deployment of the affected software. [SWE-121]
- The project manager shall transform the requirements for the software into a recorded software architecture. [SWE-057]
- The project manager shall develop, record, and maintain a software design based on the software architectural design that describes the lower-level units so that they can be coded, compiled, and tested. [SWE-058]
- The project manager shall establish and maintain: [SWE-065]
  - a. Software test plan(s).
  - b. Software test procedure(s).
  - c. Software test(s), including any code specifically written to perform test procedures.
  - d. Software test report(s).

Software Documentation Considerations

- When deciding how to prepare any of these items, consider the users of the information first.
- Reviewing and understanding the requirements, needs, and background of users and stakeholders are essential to applying the recommendations for content of software records
- Specific content within these records may not be applicable for every project.
- Use of NASA Center and contractor formats in document deliverables is acceptable if necessary content (as defined by the project) is addressed.
- Product records should be reviewed and updated as necessary.

Chapter 6: Recommended Software Records Content

6.1 It is possible to prepare a plan, associated procedures, and reports, as well as numerous records, requests, descriptions, and specifications for each software development life-cycle process. When deciding how to prepare any of these items, consider the users of the information first. Reviewing and understanding the requirements, needs, and background of users and stakeholders are essential to applying the recommendations for the content of software records defined in NASA-HDBK-2203. Specific content within these records may not apply to every project. Use of NASA Center and contractor formats in document deliverables is acceptable if the required content (as defined by the project) is addressed. Product records should be reviewed and updated as necessary. Typical software engineering products or electronic data include:

a. Software Development Plan/Software Management Plan.

b. Software Schedule.

c. Software Cost Estimate.

d. Software Configuration Management Plan.

e. Software Change Reports.

f. Software Test Plans.

g. Software Test Procedures.

h. Software Test Reports.

i. Software Version Description Reports.

j. Software Maintenance Plan.

k. Software Assurance Plan(s).

l. Software Safety Plan.

m. Software Requirements Specification.

n. Software Data Dictionary.

o. Software and Interface Design Description (Architectural Design).

p. Software Design Description.

q. Software User's Manual.

r. Records of Continuous Risk Management for Software.

Software Documentation

Typical software engineering products or electronic data include:

Plans:

- Software Development Plan/Software Management Plan.
- Software Configuration Management Plan.
- Software Test Plans.
- Software Maintenance Plan.
- Software Assurance Plan.
- Software Safety Plan, if safety-critical software.

Products:

- Software Schedule.
- Software Cost Estimate.
- Software Requirements Specification.
- Software Data Dictionary.
- Software Design Description.
- Software and Interface Design Description (Architectural Design).
- Software Change Reports.
- Software Test Procedures.
- Software Test Reports.
- Software Version Description Reports.
- Software Acceptance Criteria and Conditions.
- Software User's Manual.
- Programmer's/Developer's Manual.

Analysis products:

- Records of Continuous Risk Management for Software.
- Software Measurement Analysis Results.
- Software product analysis results
- Record of Software Engineering Trade-off Criteria & Assessments (make/buy decision).
- Software Status Reports.
- Software Reuse Report.

The recommendations for content of software records are defined in **NASA-HDBK-2203**.

The Software Engineering handbook also provides guidance regarding when these records should be drafted, baselined, and updated.

Examples and templates for these records and/or data sets are on the Software Process Across NASA (SPAN) Web site, accessible at **https://span.nasa.gov/.**

Software Life Cycle Planning

ISWE

7.18 - Documentation Guidance

Created by Haigh, Fred Douglas. (HQ-KA000)[PEROT], last modified on May 16, 2019

1. Purpose | 2. Resources | 3. Lessons Learned

1. Purpose

This topic provides a set of minimum content guidance for software project plans, reports, a
Handbook associated with NPR 7150.2A, which contained requirements for software docum
incorporated into the guidance provided in this Handbook topic.

1.1 Introduction

Guidance for each document is provided via the linked pages below. Software documentatio
pages.

The tab labels are abbreviated as follows:

- CR-PR - Software Change Request - Problem Report
- CSIP - Center Software Improvement Plan
- IDD - Interface Design Description
- Inspect - Software Inspection, Peer Reviews, Inspections
- Maint - Software Maintenance Plan
- Metrics - Software Metrics Report
- Safety - Software Safety Plan
- SAP - Software Assurance Plan
- SCMP - Software Configuration Management Plan
- SDD - Software Data Dictionary
- SDP-SMP - Software Development - Management Plan
- SRS - Software Requirements Specification
- STP - Software Test Plan
- STR - Software Test Report
- SUM - Software User Manual
- SVD - Software Version Description
- SwDD - Software Design Description
- Test - Software Test Procedures
- Train - Software Training Plan

1. Minimum Recommended Content | 2. Rationale | 3. Guidance | 4. Small Projects | 5. Resources | 6. Lessons Learned

1. Minimum Recommended Content

a. Project organizational structure showing authority and responsibility of each organizational unit, including external organizations (e.g., Safety and Mission Assurance, Independent Verification and Validation (IV&V), Technical Authority, NASA Engineering and Safety Center, NASA Safety Center).
b. The safety criticality and classification of each of the systems and subsystems containing software.
c. Tailoring compliance matrix for approval by the designated Engineering Technical Authority, if the project has any waivers or deviations to this NPR.
d. Engineering environment (for development, operation, or maintenance, as applicable), including test environment, library, equipment, facilities, standards, procedures, and tools.
e. Work breakdown structure of the life-cycle processes and activities, including the software products, software services, non-deliverable items to be performed, budgets, staffing, acquisition approach, physical resources, software size, and schedules associated with the tasks.
f. Management of the quality characteristics of the software products or services.
g. Management of safety, security, privacy, and other critical requirements of the software products or services.
h. Subcontractor management, including subcontractor selection and involvement between the subcontractor and the acquirer, if any.
i. Verification:
    i. Identification of selected software verification methods and criteria across the life cycle (e.g., software peer review/inspections procedures, re-review/inspection criteria, testing procedures).
    ii. Identification of selected work products to be verified.
    iii. Description of software verification environments that are to be established for the project (e.g., software testing environment, system testing environment, regression testing environment).
    iv. Identification of where actual software verification records and analysis of the results will be documented (e.g., test records, software peer review/inspection records) and where software verification corrective action will be documented.
j. Validation
    i. Identification of selected software validation methods and criteria across the life cycle (e.g., prototyping, user groups, simulation, analysis, acceptance testing, operational demonstrations).
    ii. Identification of selected work products to be validated.
    iii. Description of software validation environments that are to be established for the project (e.g., simulators for operational environment).
    iv. Identification of where actual software validation records and analysis of the results will be documented (e.g., user group records, prototyping records, and acceptance testing records) and where software validation corrective action will be documented.

Software Cost Estimation

NPR 7150.2D Requirements on Software Cost Estimation

- 3.2.1 To better estimate the cost of development, the project manager shall establish, document, and maintain: [SWE-015]

  a. **Two cost estimate models and associated cost parameters for all Class A and B software projects** that have an estimated project cost of $2 million or more.

  b. **One software cost estimate model** and associated cost parameter(s) for all Class A and Class B software projects that have an estimated project cost of less than $2 million.

  c. One software cost estimate model and **associated cost parameter(s)** for all C and D software projects.

  d. One software cost estimate model and associated cost parameter(s) for all Class F software projects.

- 3.2.2 The project manager's **software cost estimate(s) shall satisfy the following conditions**: [SWE-151]

  a. **Covers the entire software life-cycle**.

  b. Is **based on selected project attributes** (e.g., programmatic assumptions/constraints, assessment of the size, functionality, complexity, criticality, reuse code, modified code, and risk of the software processes and products).

  c. Is based on the cost implications of the technology to be used and the required maturation of that technology.

  d. **Incorporates risk and uncertainty**, including end state risk and threat assessments for cybersecurity.

  e. **Includes the cost of the required software assurance support**.

  f. **Includes other direct costs**.

- 3.2.3 The project manager shall **submit software planning parameters**, including size and effort estimates, milestones, and characteristics, **to the Center measurement repository at the conclusion of major milestones.** [SWE-174]

Let's do a Cost Estimate!

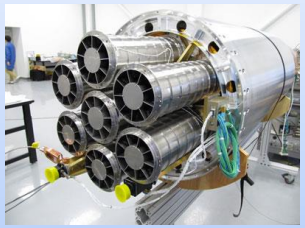
- You want to build your dream home!
  - 4000 ft2
  - Two Story Brick Veneer
  - 4 Bedrooms
  - Living Room, Dining Room, Den, Rec Room, Office, Laundry, etc.
  - 3 Baths (Master w/Separate Tub, Walk in Shower, and WC)
  - High-End Kitchen with Professional Appliances

How Much Will It Cost?

Assume that the land is provided

The Quality of an Estimate is Directly Affected by Experience, Time Available, the Detail/Maturity of the Technical Definition, and the Quantity and Relevance of Historical Data

The Cost Estimating Universe



Thousand$ Million$ Billion$

Payload Operations

Basic Research & Technology Development

Grants, Data Analysis, Space Act Agreements

Technology Development Missions

Small Optics and Components

Payloads & Large Components

Sounding Rocket & Balloon Experiments

Big Satellites

Large Observatories

Small Satellites

Human Spaceflight

Launch Vehicles

Focus of this Class*

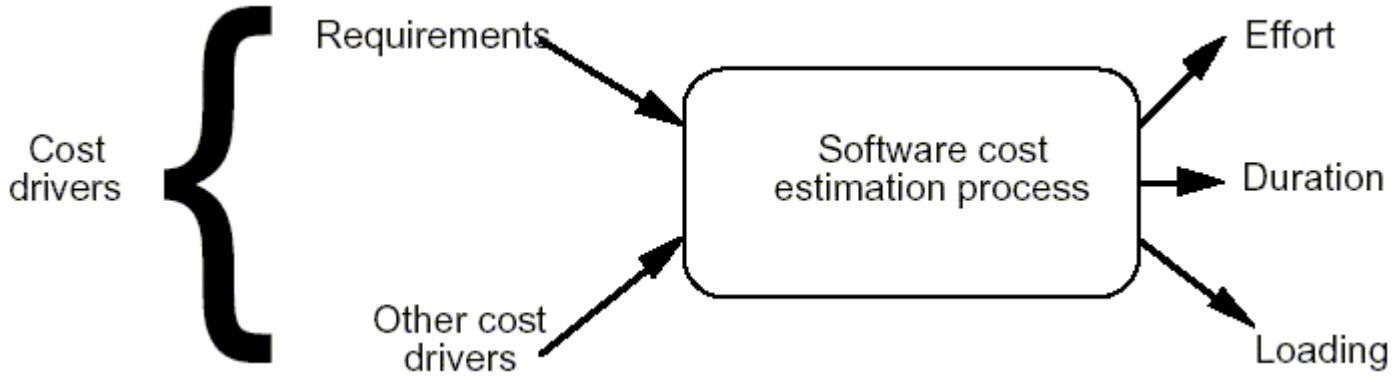
**However the Principles Apply to Any Estimate!*

Why Costing, Sizing, Progress Tracking?

- This section introduces you to some of the topics **necessary to manage your project well**
- In the current atmosphere of budget reductions, its critical to be able to **make good software cost estimates**
  - And to be able to track progress so projects finish on-time/within budget!
- NASA requires for software activities:
  - **doing at least one software cost estimate for your project, two are required for Class A and B projects $2M and over**
  - planning the project
  - tracking progress against the plan

Cost Estimating Methods

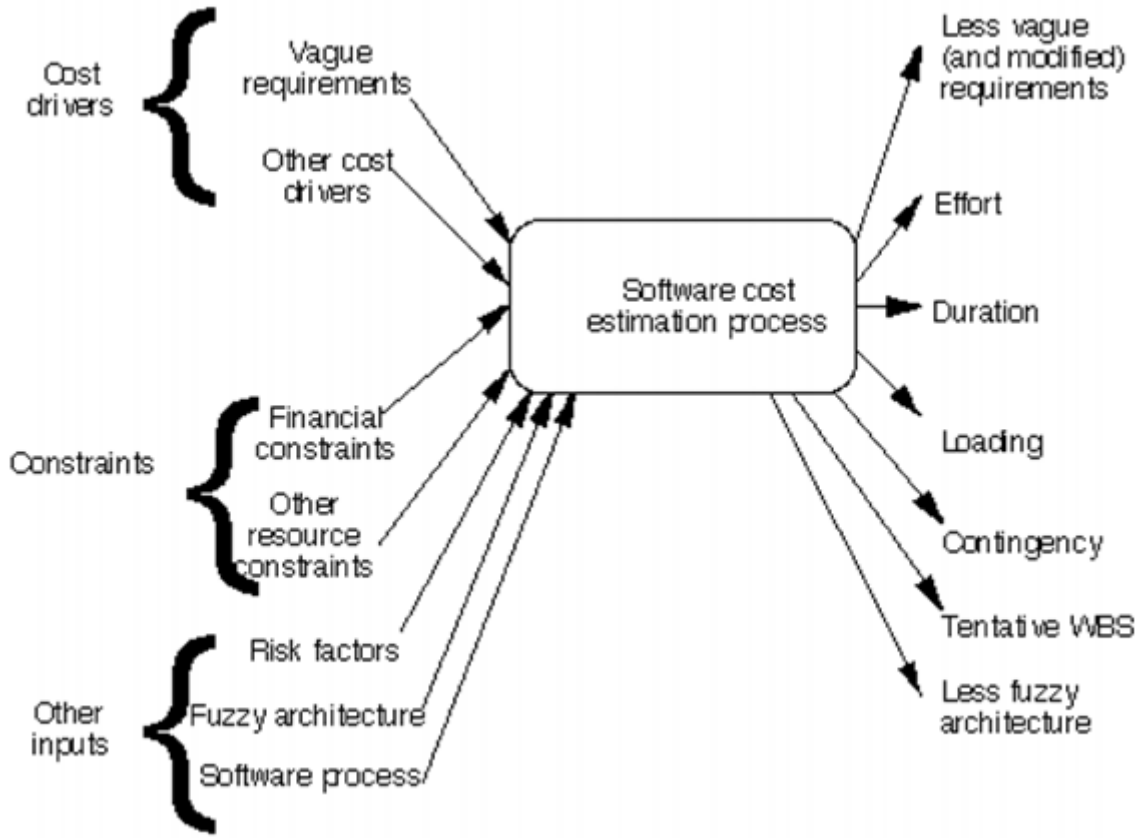
- Grass Roots/Bottoms-Up
- Analogy
- Expert Opinion/Delphi Approach
- Factors/Rules of Thumb
- Parametric



The Available Detail and Maturity of the Technical Content, Plus the Estimate Scope, Requirements, and Purpose will Determine the Best Estimating Method(s

Software Cost Estimation Issues

- Know the Purpose of a Cost Estimate
- Know How to Do a Cost Estimate
- Know if Your Cost Estimate is Any Good
- Budget 'bogies' get set very early in lifecycle. Sometimes based on casual conversations.
  - You will typically get held to this number!!
- Current proposal and planning process encourages/ demands under-estimating in early stages of lifecycle
- Software estimation is fundamentally an uncertain business under the best of conditions



Steps in Performing a Cost Estimate

1. Identify the Content of the Estimate (Spacecraft Bus, Subsystem, Component, Test, Analysis, Software components, etc.)
2. Determine the Work Required to Perform the Content
   - Design
   - Build
   - Integrate
   - Test
   - Etc.
3. Estimate the Resources Required to Perform the Work
4. Determine the Amount of Uncertainty and Risk in the Estimate
5. Validate and Document the Results

Estimating Software Size Using Source Lines of Code (SLOC)

- **Software 'size' is simply a measure of code 'bigness'**
- ***The most common way to estimate size is through Source Lines of Code (SLOC)***
- **Includes any code delivered as a software release**
- **Many definitions and standards:**
  - **Raw physical: SLOC are the total number of lines in a file**
  - **Physical: SLOC are the total number of non-blank, non-comment lines**
  - **Logical: SLOC captures size using language-specific rules.**
  - **……and many others**
- ***SLOC is easy to capture using common counting utilities***

Lines of Code estimation

- Source lines of code (SLOC) is a software metric used to measure the **size of a computer program** by counting the number of lines in the text of the program's source code.
- SLOC is typically used to **predict the amount of effort** that will be required to develop a program, as well as to **estimate programming productivity** or **maintainability** once the software is produced.

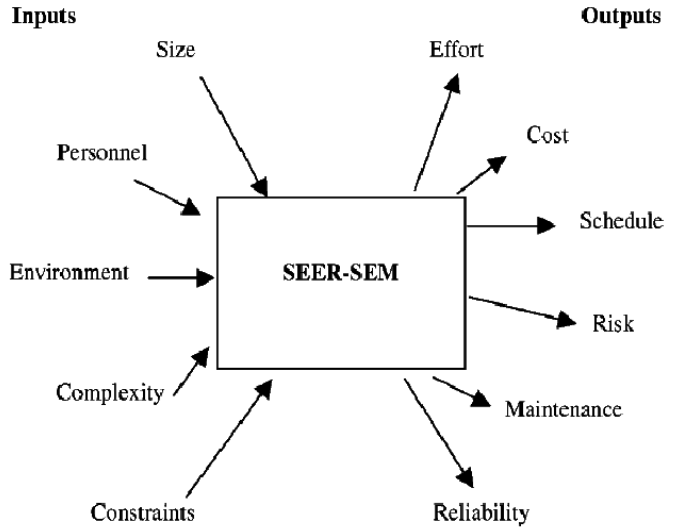
Parametric Software Cost Estimation

- Model-based estimates are estimates made using parametric cost models
- SEER-SEM and COCOMO are the two primary software cost models used with NASA
- Model-based estimates can be used
  - As a primary estimate early in life cycle
  - As a secondary backup estimate for validation
  - To help you "reason about the cost and schedule implications of software decisions you may need to make"
- Cost risk methodology using parametric models has been applied on many projects across NASA

COCOMO Conclusions

- COCOMO is the most popular software cost estimation method
- Easy to do, small estimates can be done by hand
- USC has a free graphical version available for download
- Many different commercial version based on COCOMO – they supply support and more data, but at a price

38



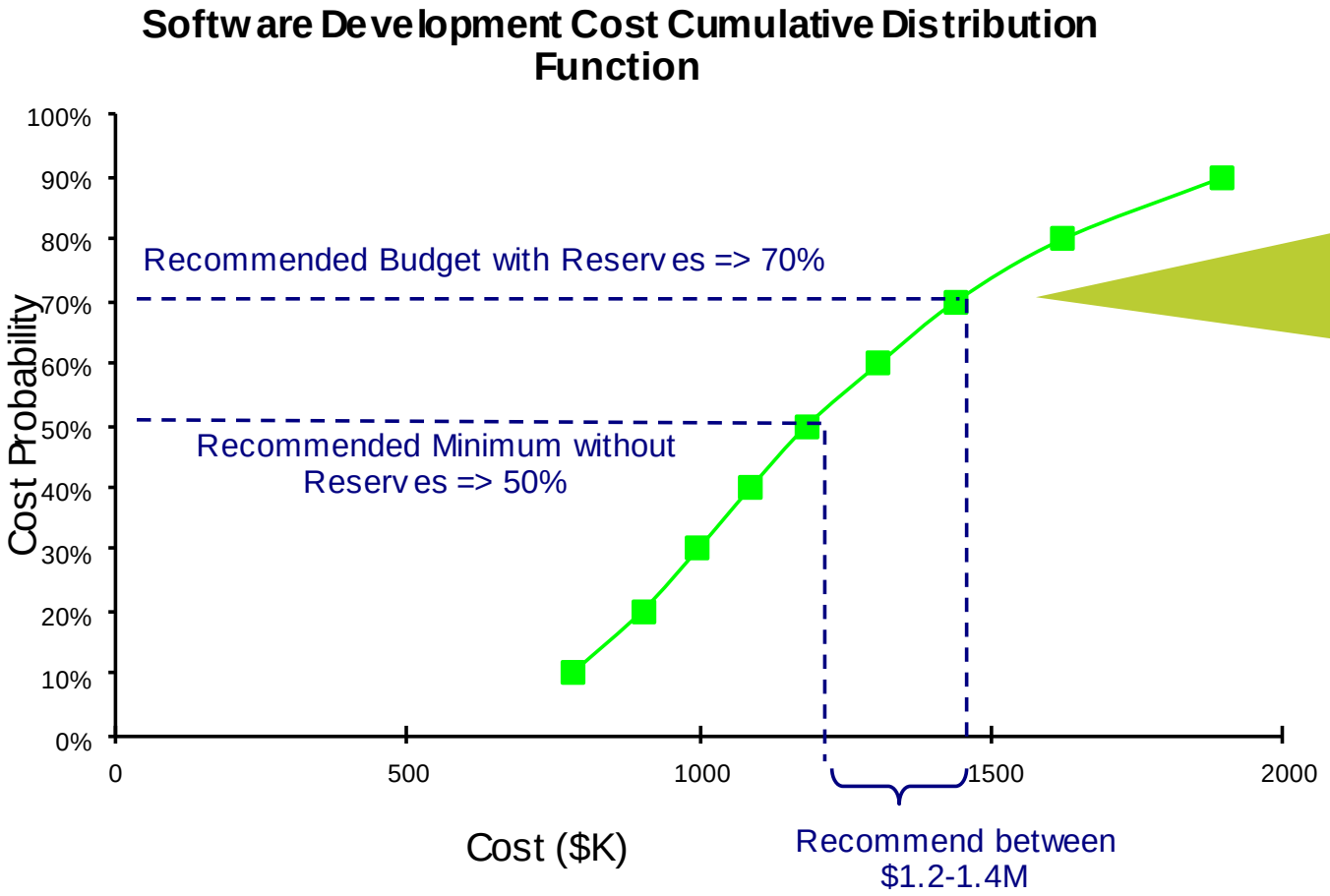
Software Cost Parameters

- Required Software Reliability:
- Database Size:
- Product Complexity
- Developed for Reusability:
- Documentation Match to Life-Cycle Needs:
- Execution Time Constraint:
- Analyst Capability:
- Programmer Capability:
- Personnel Continuity:
- Applications Experience:
- Platform Experience:
- Language and Tool Experience:
- Multisite Development:
- Required Development Schedule:
- Development Flexibility:
- Architecture / Risk Resolution
- Team Cohesion
- Process Maturity:
- Main Storage Constraint:
- Platform Volatility:
- Use of Software Tools:
- Precedentedness:

Table 1. COCOMO II Personnel Factors [23:47-49]

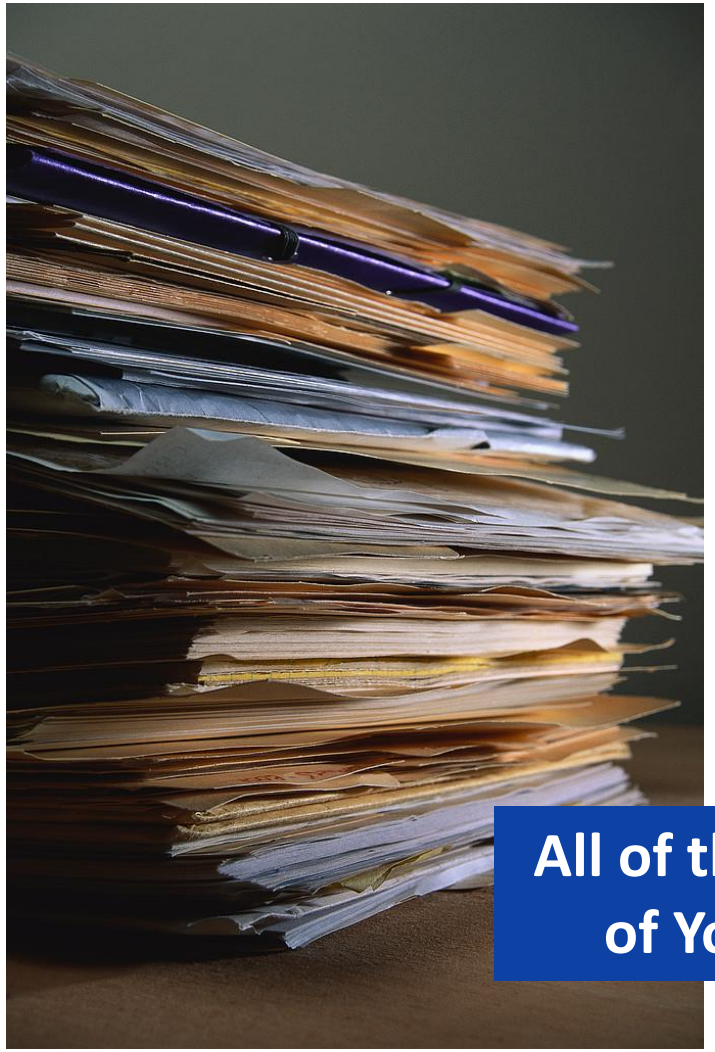
Personnel Factors		VL	L	N	H	VH	Productivity Range
Analyst Capability		1.42	1.19	1.00	0.85	0.71	2.00
Programmer Capability		1.34	1.15	1.00	0.88	0.76	1.76
Personnel Continuity		1.29	1.12	1.00	0.90	0.81	1.51
Applications Experience		1.22	1.10	1.00	0.88	0.81	1.51
Platform Experience		1.19	1.09	1.00	0.91	0.85	1.40
Language and Tool Experience		1.20	1.09	1.00	0.91	0.84	1.43

Example Model Output



- For tasks with 10% level of reserves or less recommend a range of 50% to 70% probability
- For tasks with 20% or greater reserves recommend 40-70%

Documenting the Estimate



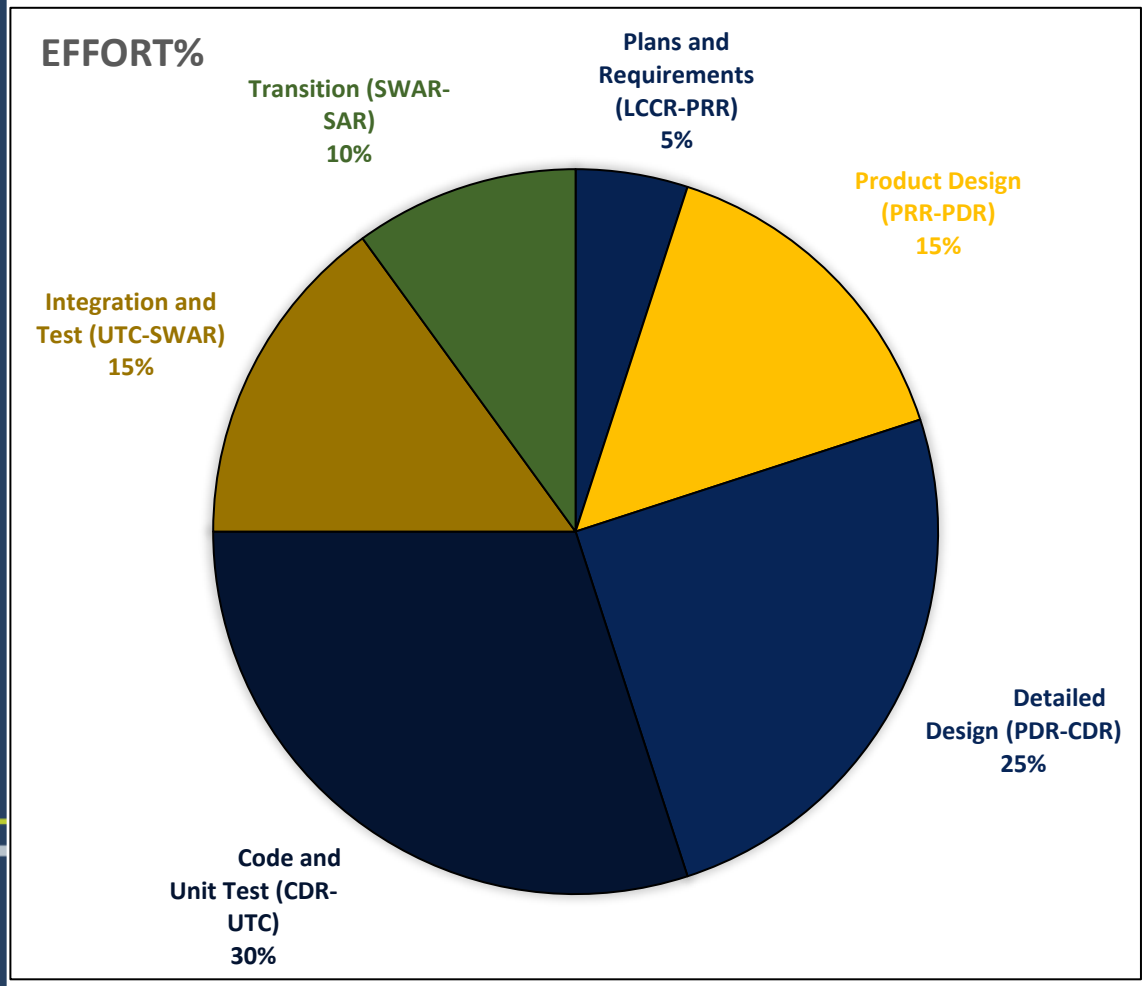
- ✓ What am I Estimating?
- ✓ Why am I doing the Estimate?
- ✓ What Information did I Use for the Estimate?
- ✓ How did I do the Estimate?
- ✓ How much Uncertainty and Risk is in the Estimate?
- ✓ How did I Validate the Results?

All of this Information becomes Part of Your Basis of Estimate (BOE)

Key Points

- Use at least two estimates
- Document the basis of estimate (BOE)
- Update estimate at significant milestones
- Keep your history
- Incorporate Uncertainty

Software Cost Data



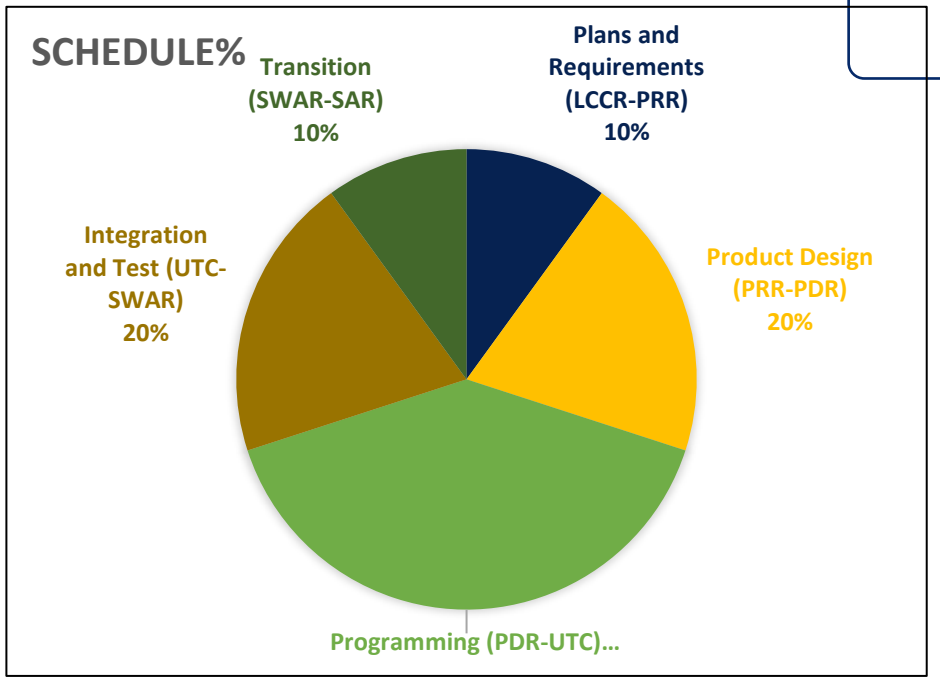


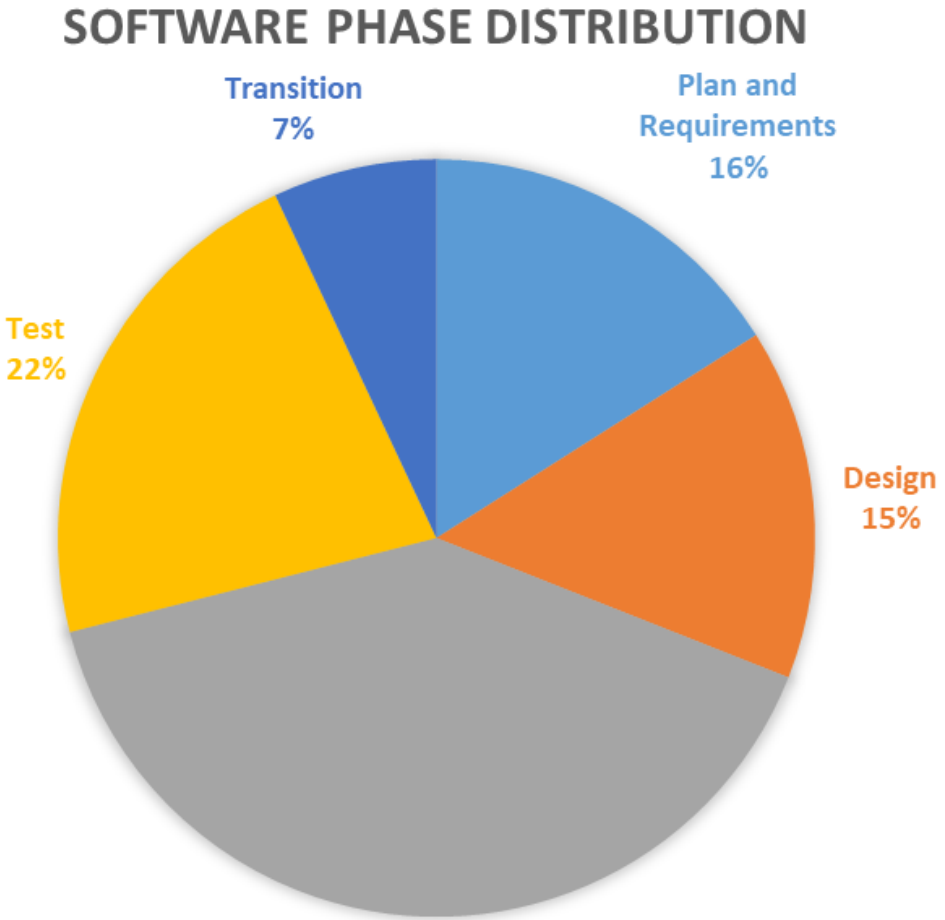
Table 4. Waterfall Phase Distribution Percentages

Phase (end points)	Effort%	Schedule%
Plans and Requirements (LCCR-PRR)	7 (2-15)	16-24 (2-30)
Product Design (PRR-PDR)	17	24-28
Programming (PDR-UTC)	64-52	56-40
Detailed Design (PDR-CDR)	27-23	
Code and Unit Test (CDR-UTC)	37-29	
Integration and Test (UTC-SWAR)	19-31	20-32
Transition (SWAR-SAR)	12 (0-20)	12.5 (0-20)

Software Cost Data

Table 4b Overall phase distribution profile

Phase	Plan&Req.	Design	Code	Test	Trans.
Min	1.82%	0.62%	6.99%	4.24%	0.06%
Max	35%	50.35%	92.84%	50.54%	36.45%
Median	15.94%	14.21%	36.36%	19.88%	4.51%
Mean	16.14%	14.88%	40.36%	21.57%	7.06%
Stdev	8.62%	8.91%	16.82%	11.04%	7.06%



Summary for Software Cost Estimation

- **Cost Estimation is *Indispensable* to Good Decision Making and Good Program/Project Management**
- **Expect a *Credible, Supportable, Defendable* Basis of Estimate**
- **Affordability Requires Awareness of the *Cost* to Perform the Work**
- **Beware of the Optimism Bias – *It will Cost More than You Think!***

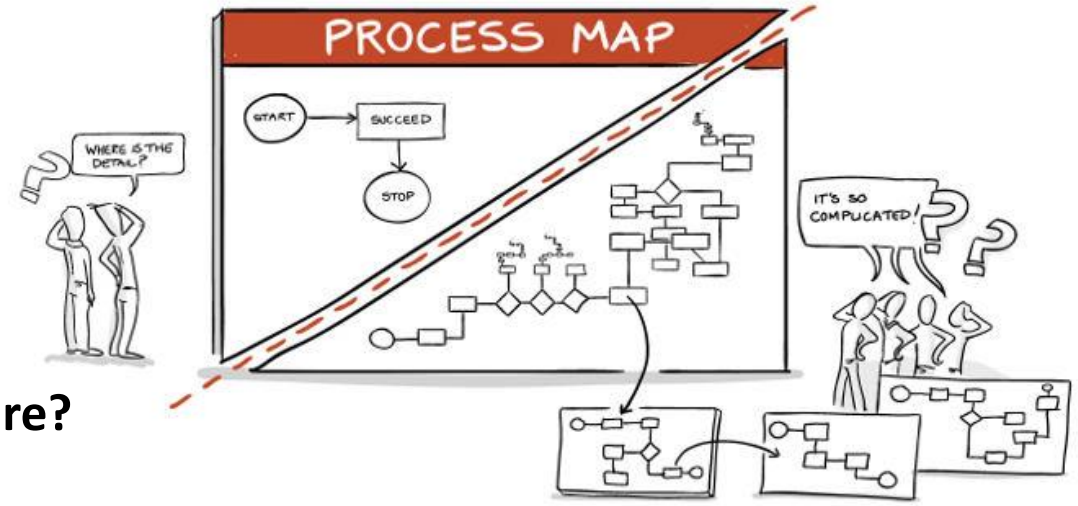
What do you need for ***a successful*** software development effort?

Software Processes

Process Questions

- **The challenge for leaders is to examine every area of their organization and identify the processes that are in place.**

- **Ask:**
  - **Does the right process or procedure exist?**
  - **Is the process effective? How do you know?**
  - **Do staff members know the outcome of the procedure?**
  - **Does everyone know the "why" of the process?**
  - **How and when is the process evaluated?**
  - **Does everyone know how they fit into the process and what to do?**
  - **Are staff members held accountable to the process?**
  - **What is the process to fix an ineffective process?**



Your process should not look like this

NASA's Software Engineering Capability as measured by CMMI Rating Level

- **What is CMMI?**
  - **The Capability Maturity Model Integration (CMMI)® is a proven set of global best practices that drives business performance through building and benchmarking key capabilities.**
  - **Is recognized worldwide as benchmark for software engineering capability**
  - **Consist of 5 well defined levels**

Originally created for the U.S. Department of Defense to assess the quality and capability of their software contractors.

https://cmmiinstitute.com/cmmi/intro

Why Are We Addressing CMMI® in This Course?

CMMI® has been shown by industry to have many benefits

When followed it can lead to better cost, schedule, and quality control, and…

It is required by NASA Directives for Class A and B software:

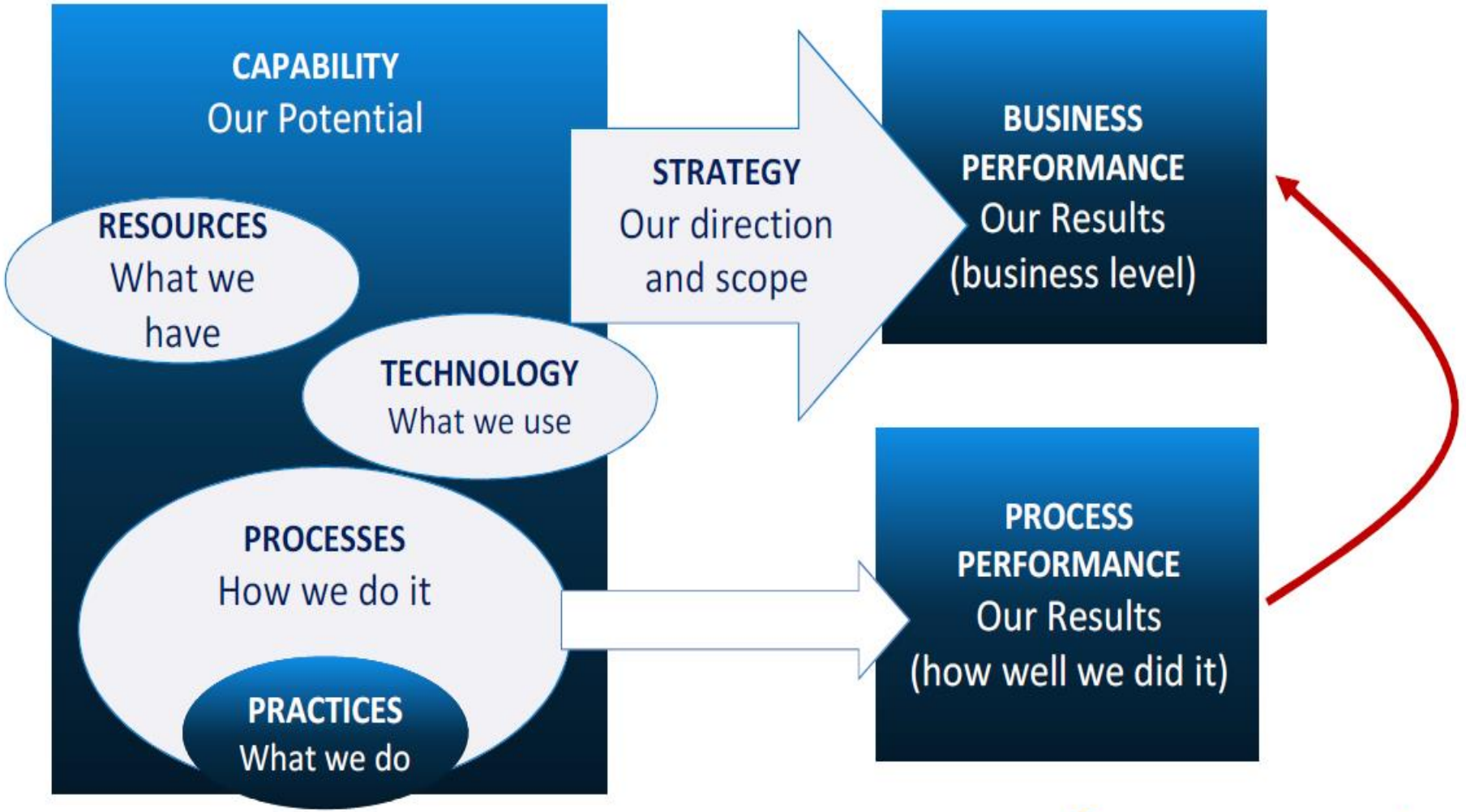
3.9.3 The project manager shall acquire, develop, and maintain software from an organization with a non-expired CMMI-DEV rating as measured by a CMMI Institute Certified Lead Appraiser as follows: [SWE-032]

- a. For Class A software: CMMI-DEV Maturity Level 3 Rating or higher for software.
- b. For Class B software (except Class B software on NASA Class D payloads, as defined in NPR 8705.4): CMMI-DEV Maturity Level 2 Rating or higher for software.

The CMMI model use at NASA

- The CMMI model is an industry-accepted model of software development practices.
- It is utilized to assess how well NASA projects are supported by software development organization(s) having the necessary skills, practices, and processes in place to produce reliable products within cost and schedule estimates. The CMMI model provides NASA with a methodology to:
  - Measure software development organizations against an industry-wide set of best practices that address software development and maintenance activities applied to products and services.
  - Measure and compare the maturity of an organization's product development and acquisition processes with the industry state of the practice.
  - Measure and ensure compliance with the intent of the directive's process related requirements using an industry standard approach.
  - Assess internal and external software development organization's processes and practices.
  - Identify potential risk areas within a given organization's software development processes and practices.

WHY USE THE CMMI MODEL?



CMMI Institute

CMMI CAPABILITY AREAS

CMMI Institute

CMMI V2.0 MODEL CORE PRACTICE AREA PRACTICE GROUP LEVELS

Why has NASA Management directed the use of CMMI® standards?

- ***The CMMI requirement is a qualifying requirement.*** **The requirement is included to *make sure NASA projects are supported by software development organization(s) having the necessary skills and processes in place to produce reliable products within cost and schedule estimates.***
- **It is a *benchmarking tool* widely used by industry and government, both in the US and abroad**
- **It acts as a *roadmap* for process improvement**
- **It *provides criteria* for reviews and appraisals**
- **It provides a *reference point to establish present state of processes***
- **It can help the government compare the maturity of one offerer (or supplier) to another**
- **It addresses practices that are the framework for process improvement**
- **It *is not prescriptive*; it does not tell an organization how to improve**

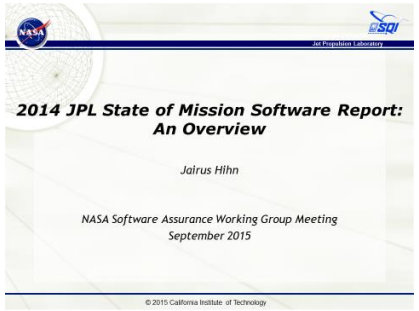
NASA SOFTWARE

Rigorous Software Processes Are Producing Superior Results

ISWE

Flight Software Key Process and Product Metrics			
Process Performance	**Effort Growth from PDR**	**Productivity (Lines of Code/ Work Month)**	**Defect Density (Defects/ Thousand Lines of Code)**
Robust Process	39%	150	4.3
Low to Moderate Process Performance	116%	106	5.9

- **In 2011 our data clearly showed the impact of the use of rigorous development processes (PPI >80%), when compared to JPL tasks with less rigorous processes (PPI <70%)**
  - ***lower cost growth***
  - ***higher productivity***
  - ***lower defect rates***

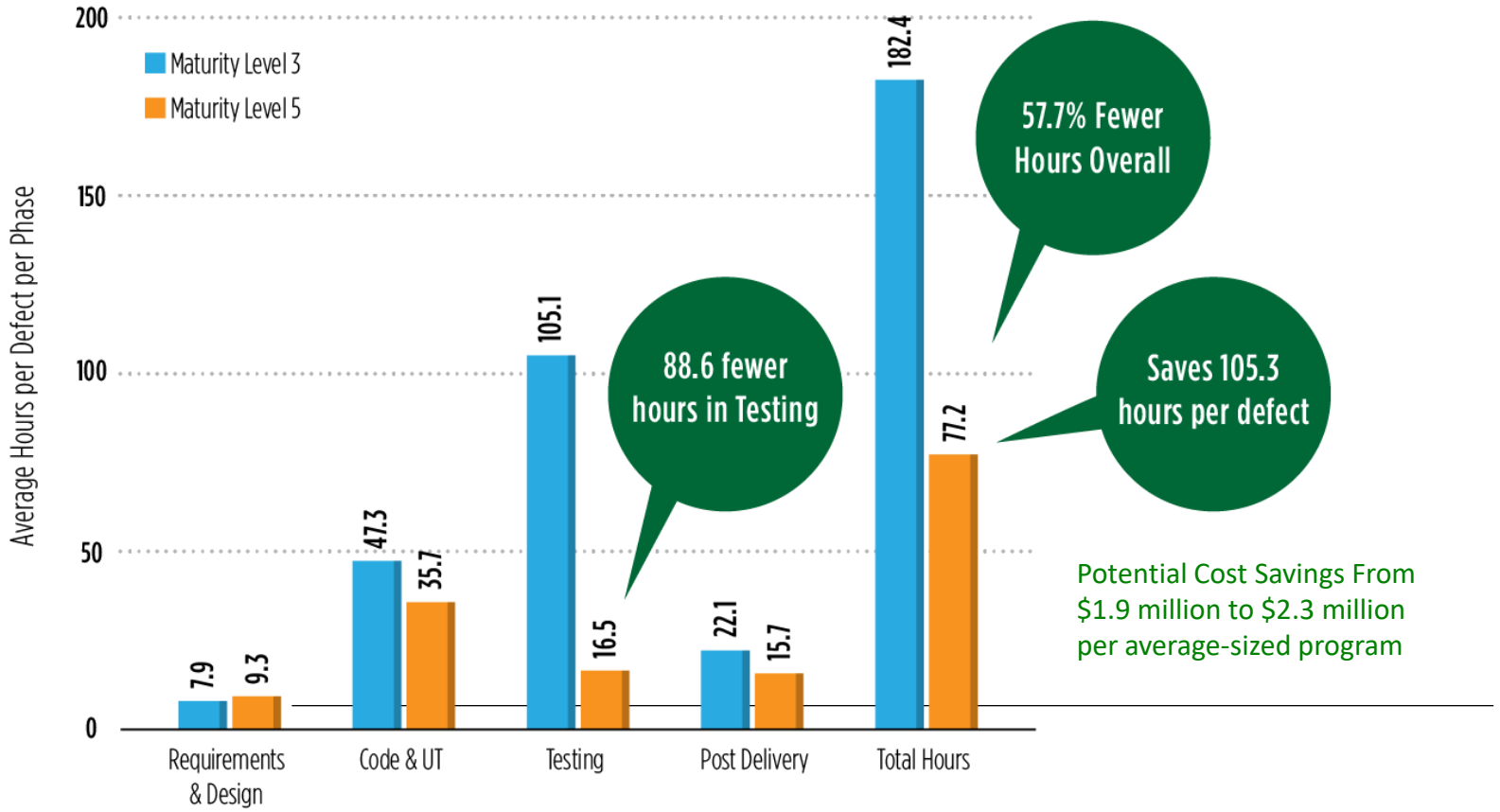


PPI – Process Performance Indicators

Quantitative Results

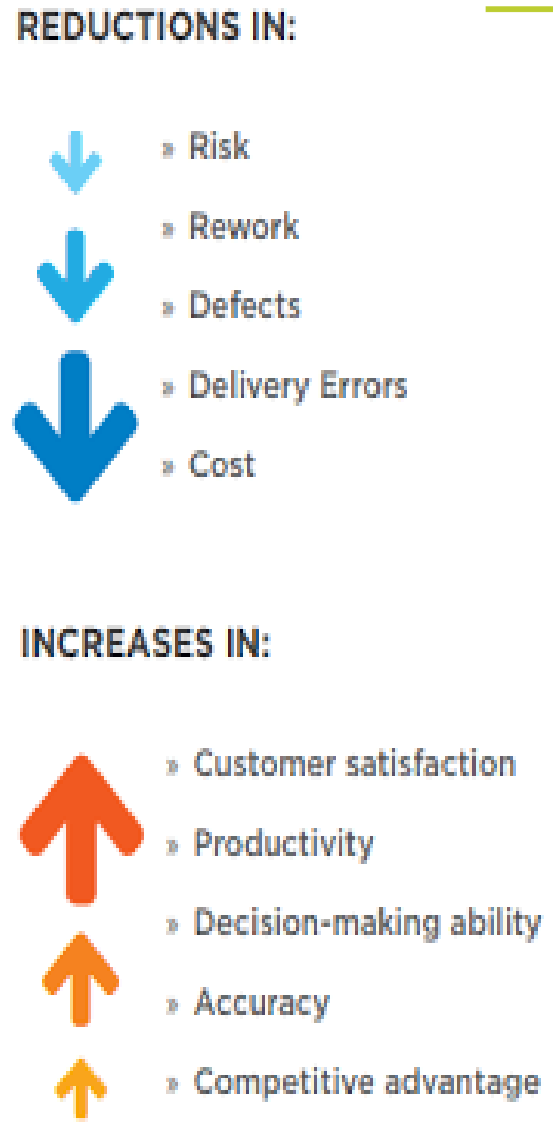
ISWE

Case Study: Defense Industry
High CMMI Maturity Reduces Costs for Repair



Benefits of CMMI

- Reducing risk of software failure - Increasing mission safety,
- Improving the accuracy of schedule and cost estimates by requiring the use of historical data and repeatable methods
- Helping NASA become a smarter buyer of contracted out software,
- Increasing quality by finding and removing more defects earlier,
- Improving the potential for reuse of tools and products across multiple projects,
- Increasing ability to meet the challenges of evolving software technology,
- Improving Software development planning across the Agency,
- Improving NASA contractor community with respect to software engineering,
- Lowering the software development cost, improves productivity
- Improving employee morale,
- Improving customer satisfaction,
- Improving NASA and Contractor community knowledge and skills,
- Providing NASA a solid foundation and structure for developing software in a disciplined manner.



Summary

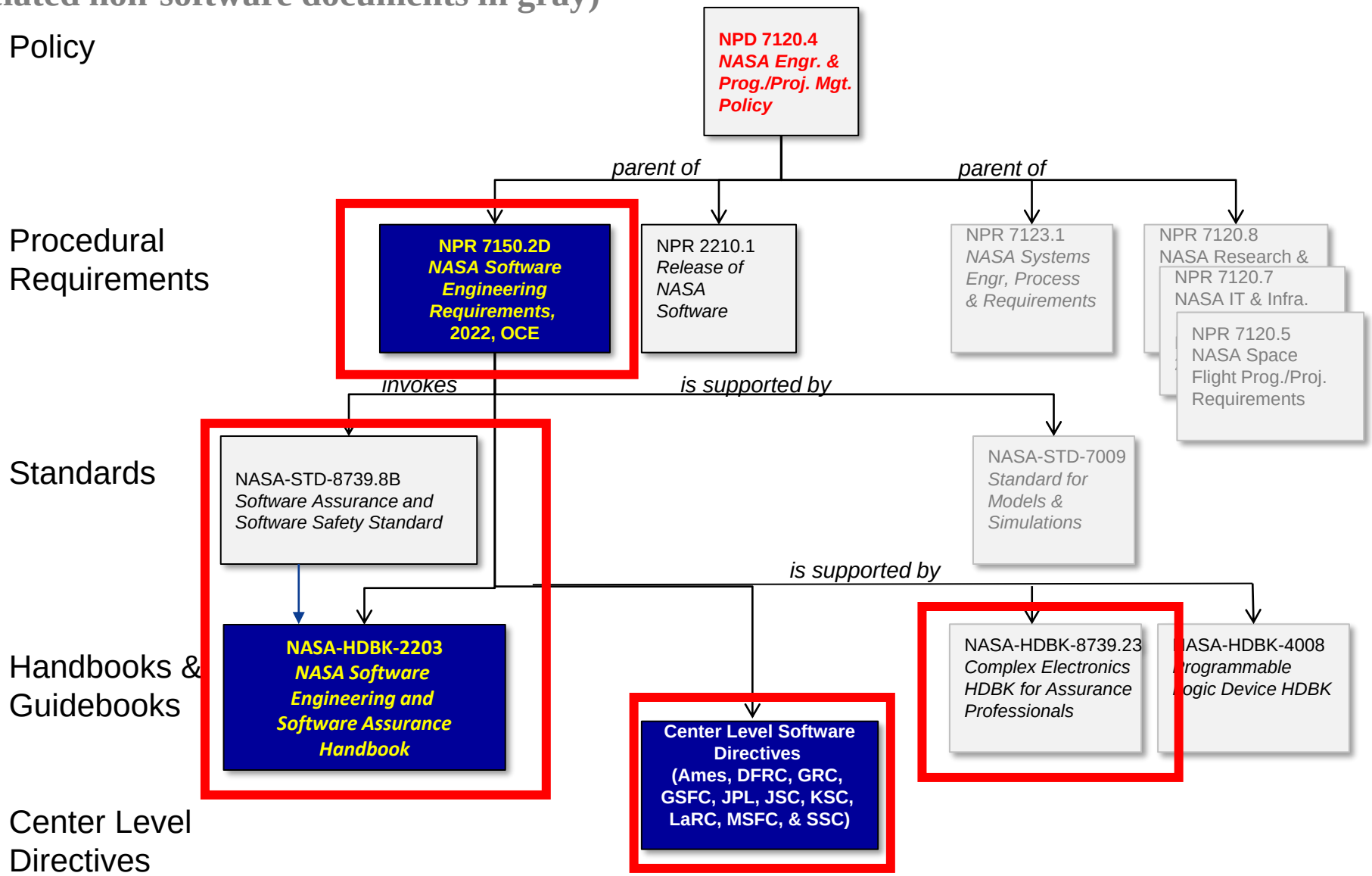
- CMMI® is an integrated framework for maturity models and associated products
- CMMI® combines
  - A set of best practices
  - A model for organizational improvement
  - A community developed guide
  - A common-sense application of process management and quality improvement concepts
- Successful projects require
  - Focus on customer satisfaction
  - Dynamic project planning
  - Compliance with NASA project requirements and plans
  - Use of appropriate methodologies and tools
  - Control of project financial and business issues

Software Assurance

Current NASA Software Documentation Tree

ISWE

(with a few related non-software documents in gray)



Documents:

- Links to the current releases of our software assurance documents are provided below. When new documents are created, or existing documents are updated, the list of links will be revised accordingly.

  NASA Software Assurance Standard (NASA-STD-8739.8)

  Complex Electronics Handbook for Assurance Professionals (NASA-HDBK-8739.23)

  https://swehb.nasa.gov/

  https://sma.nasa.gov/sma-disciplines/software-assurance-and-software-safety

  https://standards.nasa.gov/safety-quality-reliability-maintainability

The objectives or value of the Software Assurance and Software Safety include the following:

a. **Ensuring that the processes, procedures, and products used to produce and sustain the software conform to all specified requirements and standards that govern those processes, procedures, and products.**

   (a) **A set of activities that assess adherence to, and the adequacy of the software processes used to develop and modify software products.**

   (b) **A set of activities that define and assess the adequacy of software processes to provide evidence that establishes confidence that the software processes are appropriate for and produce software products of suitable quality for their intended purposes.**

b. **Determining the degree of software quality obtained by the software products.**

c. **Ensuring that the software systems are safe and that the software safety-critical requirements are followed.**

d. **Ensuring that the software systems are secure.**

e. **Employing rigorous analysis and testing methodologies to identify objective evidence and conclusions to provide an independent assessment of critical products and processes throughout the life cycle.**

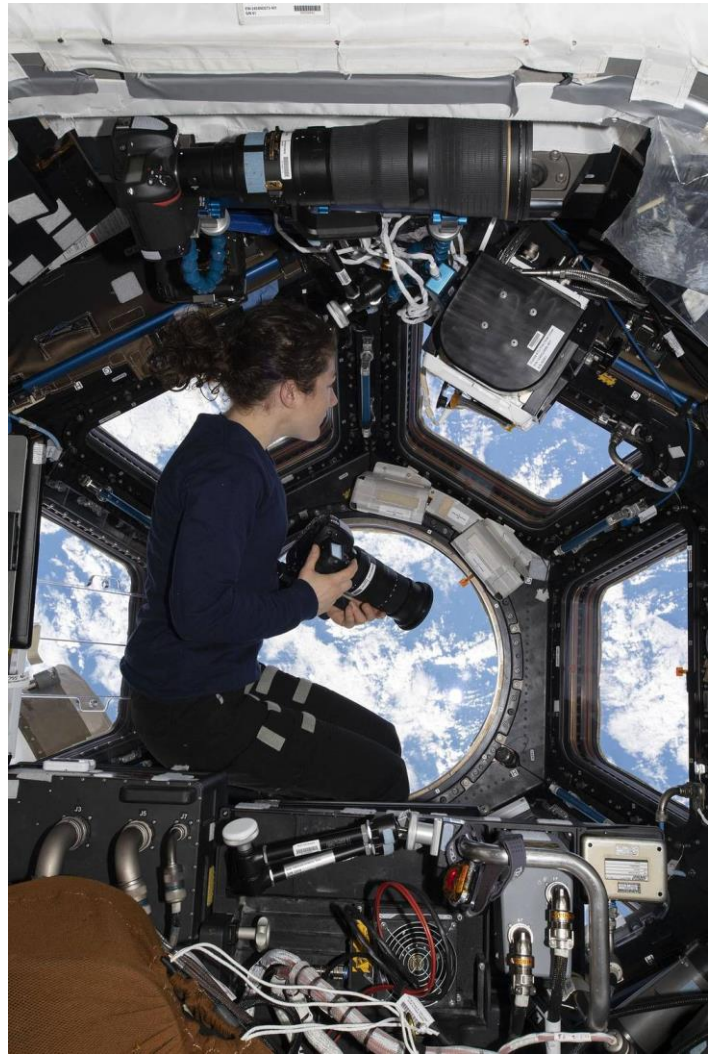
The Software Assurance activities provide a level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, that the software functions in an intended manner, and that the software does not function in an unintended manner.

Types of Software Defects Across NASA Projects

- **Requirements**
  - Missing Required Functionality
  - Poorly articulated requirements and traceability issues
  - Security controls assessment
  - High level use-case based requirements don't always fully encapsulate user expectations
- **Inadequate verification approach**
  - Depth and breadth of unit tests not adequate (based on requirements, not how code written)
  - Testing needed in development labs or simulated environments as well as hardware-in-the-loop environments
  - Limited ability to test in full up system integrated mode until System Integration Test
- **Coding Errors**
  - Data type differences; Memory Leaks; Race conditions; Timing/synchronization issues
- **Software Design**
  - Incorrect design to meet requirements
  - Interface definition not complete or missing

Examples of NASA software issues seen during operations

- Coding errors
  - Timing Discrepancies (of different varieties: between processes, between in-house built and COTS code);
  - Misunderstood requirements;
  - Changes during maintenance or updates that negate other software, have unintended consequences, or leave dead code behind
- Incomplete/Incorrect Requirements
- Incomplete ICD/Undocumented interface features
- Testing error
  - Incomplete Regression Testing
  - Incomplete set of test cases during development
  - Inadequate hardware in the loop testing
- Use of software in an unknown/unplanned configuration or scenario
- General areas that introduced errors
  - Not enough insight into contractor activities
  - Inadequate risk management
  - Inadequate peer reviews



NASA-STD-8739.8A Standard Approach

NASA
Procedural
Requirements

NPR 7150.2C
Effective Date: TBD
Expiration Date: TBD

Subject: NASA Software Engineering Requirements

Responsible Office: Office of the Chief Engineer

Chapter 3: Software Management Requirements

3.1 Software Life-Cycle Planning

3.1.1 Software life cycle planning covers the software aspects of a project from inception through retirement. The software life cycle planning cycle is an organizing process that considers the software as a whole and provides the planning activities required to ensure a coordinated, well-engineered process for defining and implementing project activities. These processes, plans, and activities are coordinated within the project. At project conception, software needs for the project are analyzed, including acquisition, supply, development, operation, maintenance, retirement, decommissioning, and supporting activities and processes. The software effort is scoped, the development processes defined, measurements defined, and activities are documented in software planning documents.

3.1.2 The project manager shall assess options for software acquisition versus development. [SWE-033]

Note: The assessment can include risk, cost, and benefits criteria for each of the options listed below:
a. Acquire an off-the-shelf software product that satisfies the requirement.
b. Develop a software product or obtain the software service internally.
c. Develop the software product or obtain the software service through contract.
d. Enhance an existing software product or service.
e. Reuse an existing software product or service.
f. Source code available external to NASA.
See the NASA Software Engineering Handbook for additional detail.

3.1.3 The project manager shall develop, maintain, and execute software plans that cover the entire software life cycle and, as a minimum, address the requirements of this directive with approved tailoring. [SWE-013]

Note: The recommended practices and guidelines for the content of different types of software planning activities (whether stand-alone or condensed into one or more project level or software documents or electronic files) are defined in NASA-HDBK-2203. The project should include or reference in the software development plans procedures for coordinating the software development and the design and the system or project development life cycle.

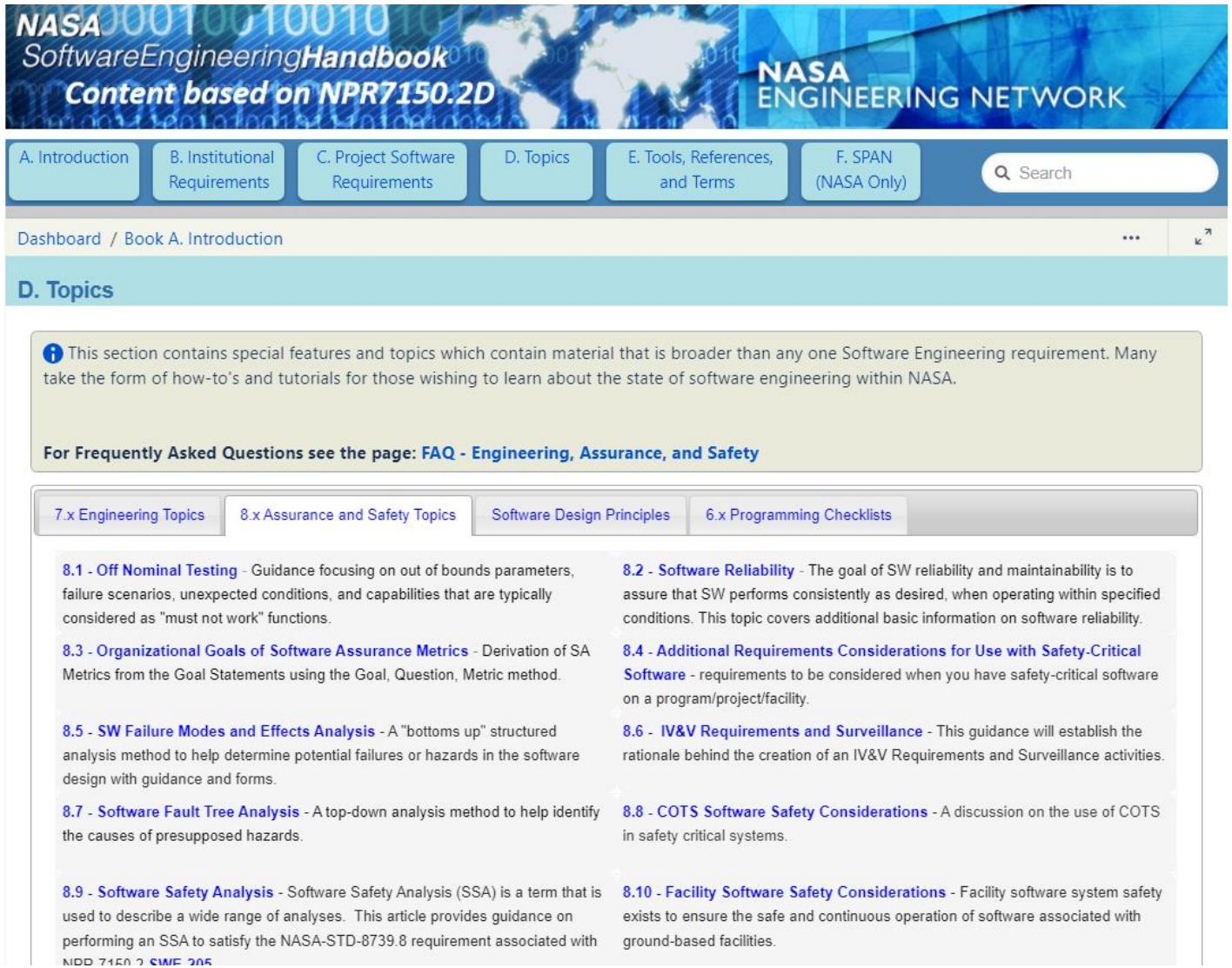
Table 1. Software Assurance and Software Safety Requirements Mapping Matrix

NPR 7150.2 Section	SWE #	NPR 7150.2 Requirement	Software Assurance and Software Safety Tasks
3		Software Management Requirements	
3.1		Software Life-Cycle Planning	
3.1.2	033	The project manager shall assess options for software acquisition versus development.	1. Confirm that the options for software acquisition versus development have been evaluated. 2. Confirm the flow down of applicable software engineering, software assurance, and software safety requirements on all acquisition activities. (NPR 7150.2 and NASA-STD-8739.8). 3. Assess any risks with acquisition versus development decision(s).
3.1.3	013	The project manager shall develop, maintain, and execute software plans that cover the entire software life-cycle and, as a minimum, address the requirements of this directive with approved tailoring.	1. Confirm that all plans are in place, and have expected content for the life-cycle events, with proper tailoring for the classification of the software. 2. Develop a Software Assurance Plan following the content defined in NASA-HDBK-2203 for a software assurance plan, including software safety.
3.1.4	024	The project manager shall track the actual results and performance of software activities against the software plans. a. Corrective actions are taken, recorded, and managed to closure. b. Including changes to commitments (e.g., software plans) that have been agreed to by the affected groups and individuals.	1. Assess plans for compliance with NPR 7150.2 requirements, NASA-STD-8739.8, including changes to commitments. 2. Confirm that closure of corrective actions associated with the performance of software activities against the software plans, including closure rationale.
3.1.5	034	The project manager shall define and document the accepta	1. Confirm software acceptance criteria are defined

- Performance-based requirements
- Closely tied to Engineering Requirements in NPR 7150.2C

Software Handbook – Requirements Example

Software Engineering and Software Assurance Handbook Topics



Key Software Assurance and Software Safety Activities

	Software Assurance Planning
1	Implementation of the NASA-STD-8739.8 requirements
2	Software assurance\safety requirements mapping matrix, review any tailored requirements
3	Software assurance\safety approach, plan and resource allocations
4	Software assurance\safety requirements flow down into contracts
	Software Assurance Analysis
5	Software requirements analysis
6	Software safety analysis
7	Software test analysis
8	Software hazard analysis
9	Software source code quality analysis
10	Peer reviews
11	**Static Analysis Tools Assessments**
	Audits
12	Software engineering requirements flow down and implementation
13	Software process audits
14	Software test witnessing
	Communication
15	Software assurance and software safety planned activities
16	Metric and status reporting by software assurance\safety
17	IV&V plan and communication (if required)
18	Software risks, findings or known issues
	Product reviews
19	Major Milestone product reviews
20	Software development product reviews
21	Software metric data reviews
	Defect Tracking and Management
22	Root causes analysis

SA Tasking Checklist Tool

Software Assurance Planning activities

- Checklist tool that gives Software Assurance and Software Safety analysts the ability to tailor the software assurance and software safety tasks in NASA-STD-8739.8 and generate a tailored checklist for the tasks required on a project's software classification and safety criticality.

SA Tasking Checklist

Project: Project	Safety Critical: Yes	Export to Excel	
Project Manager: Project Manager	Software Classification: A	Export to CSV	
Project POC: Project POC	Milestone(s): SwRR, PDR, CDR, TRR, SAR	Export to XML	
SA Analyst: SA Analyst			
SA Tech Authority:	SA Tech Authority Approval Date:		

Checklist

Milestone	SWE #	NPR 7150.2 Requirement	NASA-STD-8739.8A SA Tasking -Tailored	SWE Handbook Link	Analysis Type	Status	Date Planned
SwRR	33	The project manager shall assess options for software acquisition versus development.	1. Confirm that the options for software acquisition versus development have been evaluated.	https://swehb.nasa.gov/display/SWEHBVC/SWE-033+-+Acquisition+vs.+Development+Assessment	Confirm	Not Started	
SwRR	33	The project manager shall assess options for software acquisition versus development.	2. Confirm the flow down of applicable software engineering, software assurance, and software safety requirements on all acquisition activities. (NPR 7150.2 and NASA-STD-8739.8).	https://swehb.nasa.gov/display/SWEHBVC/SWE-033+-+Acquisition+vs.+Development+Assessment	Confirm	Not Started	
SwRR	33	The project manager shall assess options for software acquisition versus development.	3. Assess any risks with acquisition versus development decision(s).	https://swehb.nasa.gov/display/SWEHBVC/SWE-033+-+Acquisition+vs.+Development+Assessment	Assess	Not Started	
SwRR	13	The project manager shall develop, maintain, and execute software plans that cover the entire software life-cycle and, as a minimum, address the requirements of this directive with approved tailoring.	1. Confirm that all plans are in place, and have expected content for the life-cycle events, with proper tailoring for the classification of the software.	https://swehb.nasa.gov/display/SWEHBVC/SWE-013+-+Software+Plans	Confirm	Not Started	
SwRR	13	The project manager shall develop, maintain, and execute software plans that cover the entire software life-cycle and, as a minimum, address the requirements of this directive with approved tailoring.	2. Develop a Software Assurance Plan following the content defined in NASA-HDBK-2203 for a software assurance plan, including software safety.	https://swehb.nasa.gov/display/SWEHBVC/SWE-013+-+Software+Plans	Develop	Not Started	
SwRR	24	The project manager shall track the actual results and performance of software activities against the software plans. a. Corrective actions are taken, recorded, and managed to closure. b. Including changes to commitments (e.g., software plans) that have been agreed to by the affected groups and individuals.	1. Assess plans for compliance with NPR 7150.2 requirements, NASA-STD-8739.8, including changes to commitments.	https://swehb.nasa.gov/display/SWEHBVC/SWE-024+-+Plan+Tracking	Assess	Not Started	
			actions associated with the gainst the software plans, including	https://swehb.nasa.gov/display/SWEHBVC/SWE-024+-+Plan+Tracking	Confirm	Not Started	
		the software.	ria are defined and assess the criteria based on guidance in the NASA Software Engineering Handbook, NASA-HDBK-2203.	https://swehb.nasa.gov/display/SWEHBVC/SWE-034+-+Acceptance+Criteria	Confirm	Not Started	
SwRR	36	The project manager shall establish and maintain the software processes, software documentation plans, list of developed electronic products, deliverables, and list of tasks for the software development that are required	1. Confirm the following are approved, implemented, and updated per requirements: a. Software processes, including software assurance, software safety, and	https://swehb.nasa.gov/display/SWEHBVC/SWE-036+-+Software+Process+Determination	Confirm	Not Started	

https://swehb.nasa.gov/display/SWEHBVD/8.15+-+SA+Tasking+Checklist+Tool

Primary Software Assurance and Software Safety work products

Software Assurance Analyses activities

- Software Assurance Plan - Describes Software Assurance Plan content as well as sub-plans for Safety and Security
- IV&V Program Execution Plan - This is produced by the IV&V team, if software IV&V is required on a project.
- Software Requirements Analysis - This section focuses on analysis techniques for assuring and improving requirements
- Software Safety and Hazard Analysis - (Only applicable for safety critical projects) - Under Construction –
- Software Design Analysis – Section focuses on analysis techniques for improving the design.
- Source Code Quality Analysis - Section focuses on analysis techniques for determining and improving source code quality.
- Testing Analysis - Discusses considerations for developing and evaluating test products (test plans, test procedures and test results)
- Software Assurance Status Reports - Contains recommended content for SA status reporting, including reporting details for analysis, assessments and audits.
- Audit Reports - Discusses required audits and provides information and resources for performing audits
- Objective Evidence - This topic provides a definition with some examples of "objective evidence" and contains a listing of all the tasks in NPR-8739.8 [278] where "objective evidence" may be the only product.
- Hazard inputs
- Findings, issues, defects, problem reports, and identified software risks

https://swehb.nasa.gov/display/SWEHBVD/8.16+-+SA+Products

Software source code quality analysis

Code Risk

- Drilling down a level – and particularly for mission- and safety-critical systems, the code itself entails risks. For example, consider the risk that a code base is:
  - Hard to test thoroughly
  - Prone to critical failures / crashes
  - Unmaintainable over its expected lifecycle
  - Tough to extend for new capabilities
  - Exploitable to cyber attacks
  - Difficult to harvest for reuse
  - Plagued with a multitude of latent defects
  - Hard to change without adding new defects

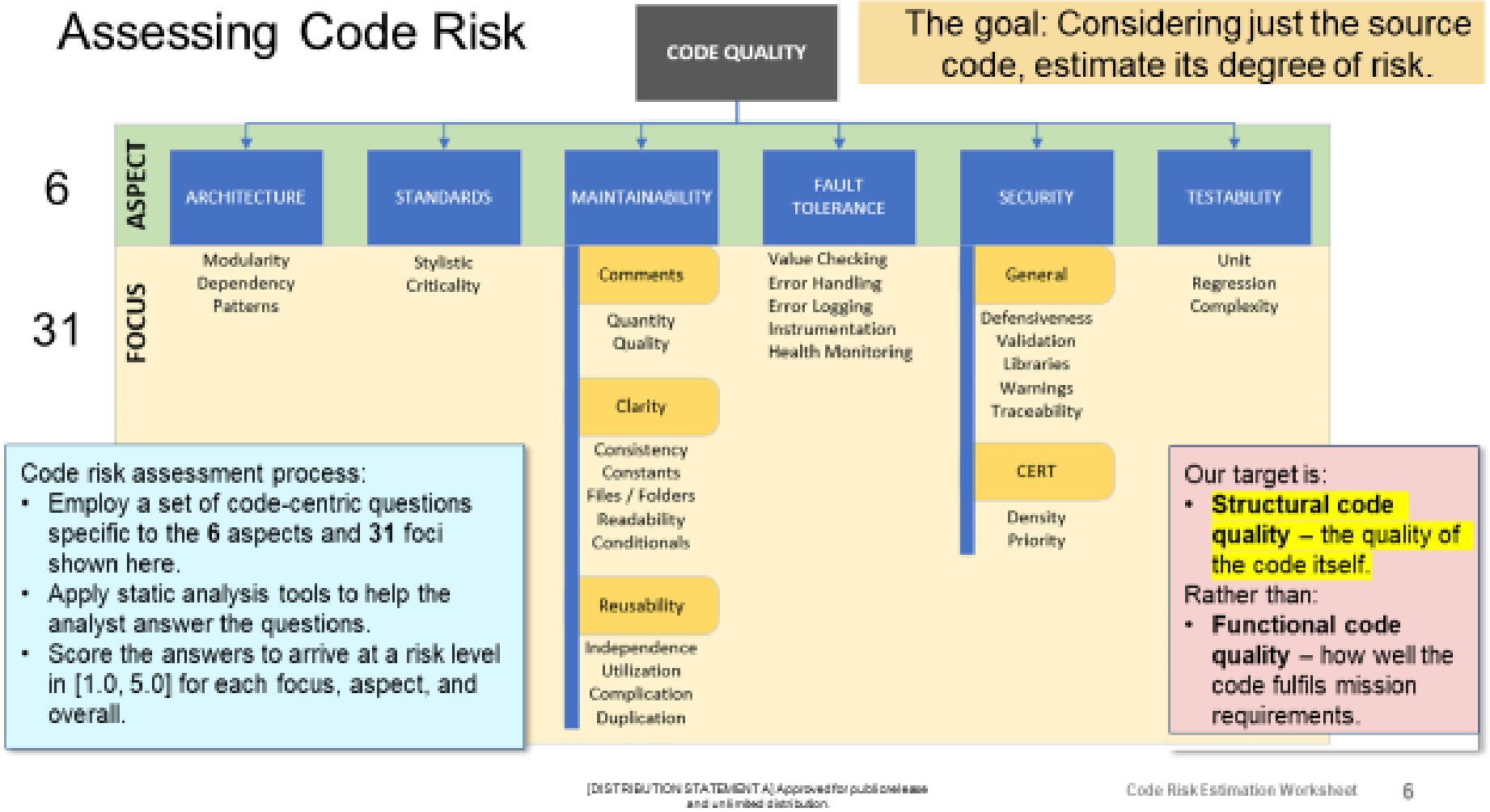
Poor Software Quality Costs the United States *How Much*?

January 27, 2021 at 2:00pm ET / 11:00am PT

This webinar will introduce The Cost of Poor Software Quality in the US: A 2020 Report published in January by CISQ. The report estimates that poor software quality cost the United States economy over $2 trillion in 2020 due to operational software failures, poor quality legacy systems, and unsuccessful projects. Compared to the country's projected Gross Domestic Product (GDP) of $20.66 trillion, or the $1.4 trillion spent on employing IT/software professionals in 2020, it represents a staggering amount of wasted resources.

Is there a way to characterize these types of risk for a given code base?

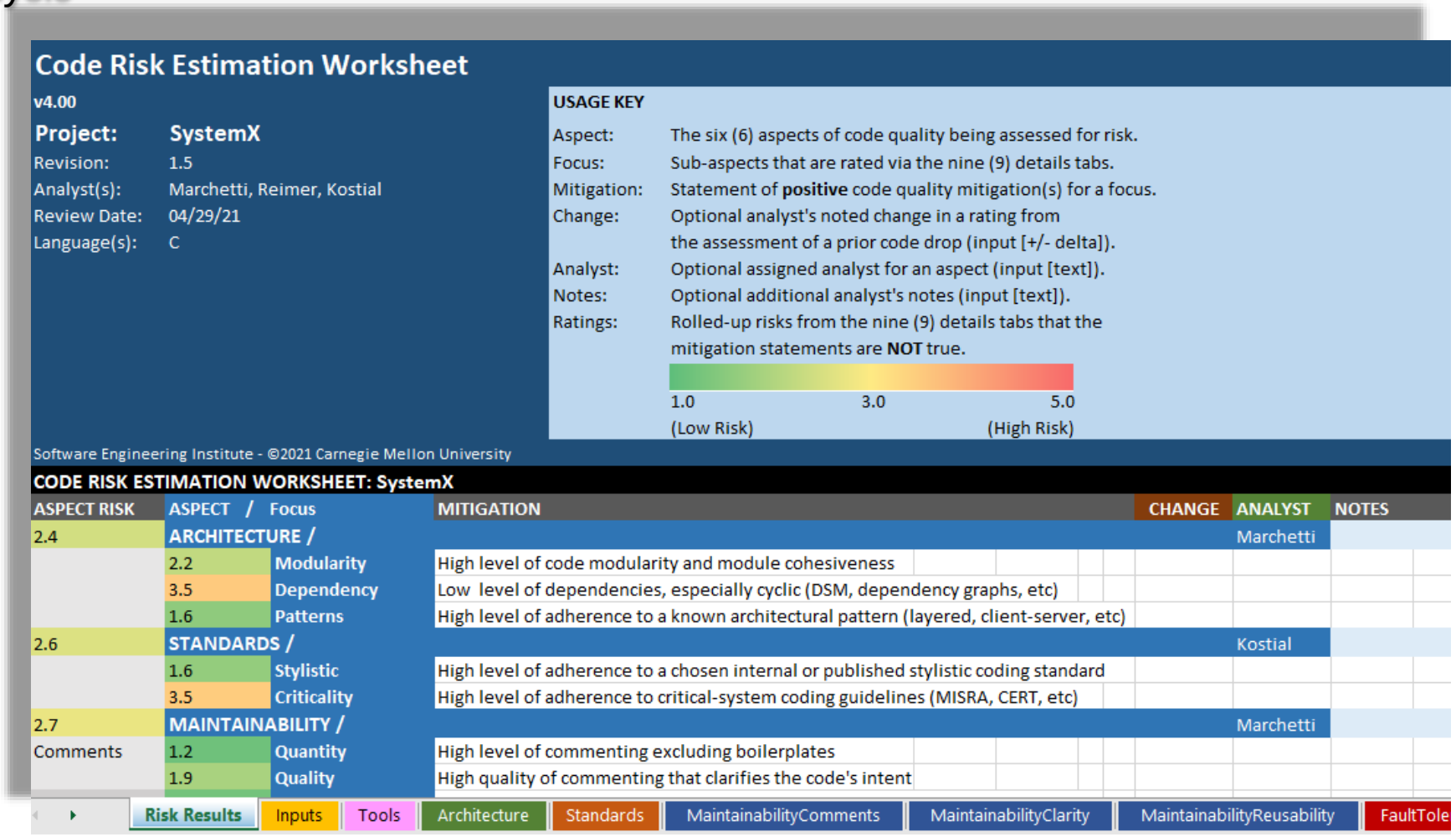
Software source code quality analysis



Software source code quality analysis

We can use the Code Risk Estimation Worksheet to improve code assessments, enhancing:

- Thoroughness
- Objectivity
- Consistency
- Traceability
- Standards adherence



Code Risk Estimation Worksheet

v4.00

Project: SystemX
Revision: 1.5
Analyst(s): Marchetti, Reimer, Kostial
Review Date: 04/29/21
Language(s): C

USAGE KEY

Aspect:	The six (6) aspects of code quality being assessed for risk.
Focus:	Sub-aspects that are rated via the nine (9) details tabs.
Mitigation:	Statement of **positive** code quality mitigation(s) for a focus.
Change:	Optional analyst's noted change in a rating from the assessment of a prior code drop (input [+/- delta]).
Analyst:	Optional assigned analyst for an aspect (input [text]).
Notes:	Optional additional analyst's notes (input [text]).
Ratings:	Rolled-up risks from the nine (9) details tabs that the mitigation statements are **NOT** true.

1.0 (Low Risk) 3.0 5.0 (High Risk)

Software Engineering Institute - ©2021 Carnegie Mellon University

CODE RISK ESTIMATION WORKSHEET: SystemX

ASPECT RISK	ASPECT / Focus		MITIGATION	CHANGE	ANALYST	NOTES
2.4	ARCHITECTURE /				Marchetti	
	2.2	Modularity	High level of code modularity and module cohesiveness			
	3.5	Dependency	Low level of dependencies, especially cyclic (DSM, dependency graphs, etc)			
	1.6	Patterns	High level of adherence to a known architectural pattern (layered, client-server, etc)			
2.6	STANDARDS /				Kostial	
	1.6	Stylistic	High level of adherence to a chosen internal or published stylistic coding standard			
	3.5	Criticality	High level of adherence to critical-system coding guidelines (MISRA, CERT, etc)			
2.7	MAINTAINABILITY /				Marchetti	
Comments	1.2	Quantity	High level of commenting excluding boilerplates			
	1.9	Quality	High quality of commenting that clarifies the code's intent			

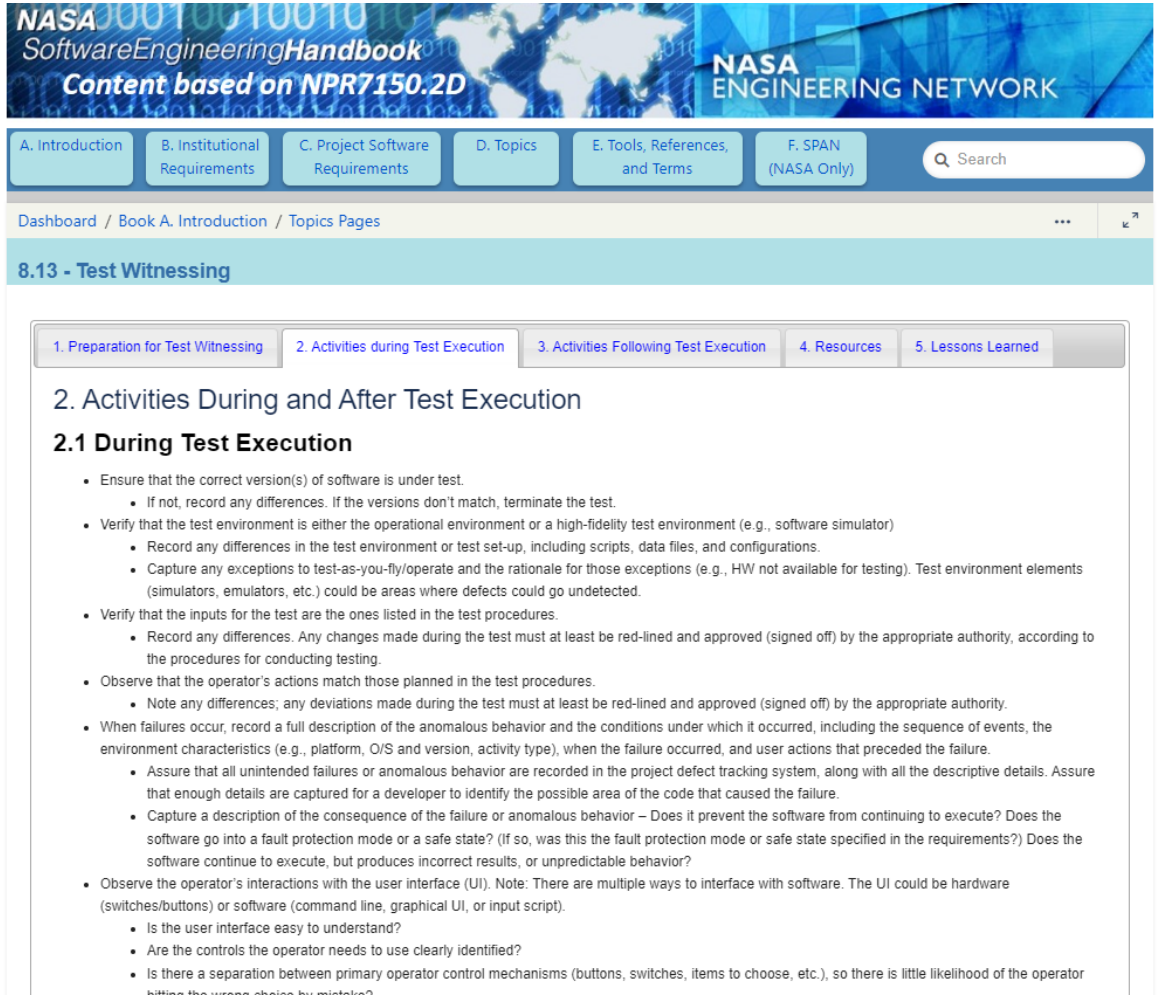
Risk Results | Inputs | Tools | Architecture | Standards | MaintainabilityComments | MaintainabilityClarity | MaintainabilityReusability | FaultTole

The resulting estimates provide customers with an easy to understand snapshot of the risk level inherent within their code base.

Software test witnessing

Guidance for software assurance personnel performing test witnessing.

- Software assurance will review the test procedures and either review test results or witness the tests being run to confirm the test coverage of the requirements.
- In projects with safety-critical code, software assurance will perform extra rigor to ensure that all safety-related features are thoroughly tested.
- Tests for safety features should include testing in operational scenarios, nominal scenarios, off-nominal conditions, stress conditions, and error conditions that require bringing the system to a safe mode.
- Projects should do regression for any changes made to the software during the test process, following the project's change management process.



https://swehb.nasa.gov/display/SWEHBVD/8.13+-+Test+Witnessing

Static Code Analysis

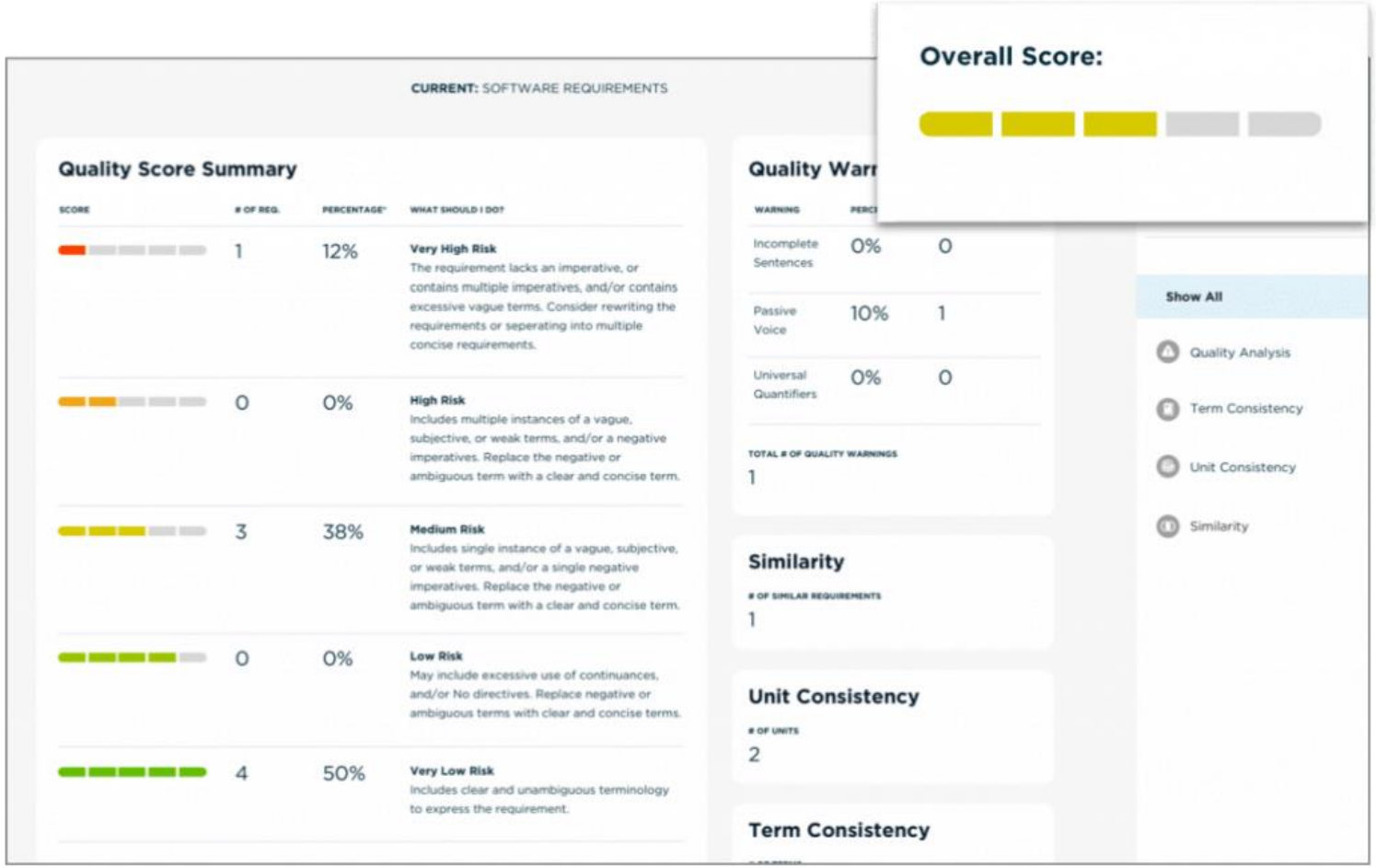
Static Analysis Tools Assessments

- SWE-135 in NPR 7150.2 requires the use of static analyzer tools during development and testing.
- Modern static code analysis tools can identify a variety of issues and problems, including but not limited to dead code, non-compliances with coding standards, security vulnerabilities, race conditions, memory leaks, and redundant code.
- Software peer reviews/inspections of code items can include reviewing the results from static code analysis tools.
- One issue with static code analyzers is they may generate a number of false positives that will need to be resolved and can be very time consuming.
- Static code analyzers are not available for all platforms or languages.
- For critical code, it is essential to use sound and complete static analyzers.

Examples of some SCA Tools used across NASA
CodeSonar
Cppcheck
HPFortify
Klocwork
SonarQube
Understand
coverity
FindBugs/SpotBugs
IKOS
JPL CAE SRUB
lgtm
OCLint
Parasoft C++
Polyspace
PRQA
RIPS
semmle
VI Analyzer (LabVIEW)

Requirements Analysis Software

- Manual requirements review is an unreliable process.
- There are simply too many elements to confirm against industry standards and best practices (as well as internal best practices) for these manual checks to be fully accurate.
- Poor requirements analysis can lead to costly corrections in later development phases that would otherwise be easier and less expensive to correct when requirements are first written.



qvscribe

Agency wide tool
Hosted by the NSC

Software Assurance Auditing Function

Audits

- Audits provide management with information about the project team, the project processes and help identify best practices and areas of improvement.
- Audits are useful to assess:
  - Adequacy of project plans, processes, systems
  - Compliance with those plans, processes, systems
  - Effectiveness of those plans, processes, systems, and internal project controls on those processes
  - Product fitness for use/compliance to specifications
  - Areas for improvement
- The results of audits allow project management to make adjustments and corrections to ensure high-quality products are being produced and delivered and that the team is functioning efficiently and effectively.
- Trending audit results over time allows management to identify systemic issues and areas of risk while monitoring the effect of process and product improvements.

Ensuring that the processes, procedures, and products used to produce and sustain the software conform to all specified requirements and standards that govern those processes, procedures, and products.

https://swehb.nasa.gov/display/SWEHBVD/8.12+-+Basics+of+Software+Auditing

Principle	Description
Auditors are qualified	Auditors need to have knowledge of or experience with audit processes and necessary backgrounds in the audit subject matter, such as software engineering or software assurance. Qualification can be through training, on-the-job experience, a mentor-mentee relationship, or simply by including a variety of these skills on the audit team.
An audit is against agreed-to requirements/criteria	To get the best objective results, define the audit criteria before the project starts (i.e., the process requirements, standards, development plans, etc. to be used for the audit). The team being audited knows they are expected to follow these criteria so the audit team simply looks for evidence of that compliance.
Conclusions are based on the evidence	Audit results are based on and backed up by the collected evidence only.
The audit focuses on the project records, not the personnel	An audit is designed to assess compliance, not personalities or behavior; therefore, the auditors focus on the records, the interviews, and observations to determine the results.

Maturity of software assurance and software safety products at milestone reviews

Communication

Software Assurance and Software Safety	MCR	SRR	MDR	SDR	PDR	CDR	SIR	TRR	SAR	ORR
Analysis showing software requirement and hazard control coverage			D	P	B	U	U	U	U	
Cost estimate for the project's SA support.	D	P	U	U	B	U				
IV&V Project Execution Plan (IPEP)		B	U	U	U	U	U	U	U	U
Preliminary Hazard Analysis and software controls and mitigations (FHA / Hazard Reports / Hazard Analysis Tracking Index)		P	U	U	U	B	U			
Requirements mapping table for the SA requirements.				P	B	U	U	U		
SA analysis showing uncovered software code percentage						P	U	U	U	U
SA audit and status reports		U	U	U	U	U	U	U	U	U
SA metric analysis procedures.		P	U	U	B	U				
SA Product Acceptance Criteria and Conditions					P	B				
SA schedule	D	P	U	U	B	U				
Software Assurance and Software Safety Plan(s)		P	P	P	B	U				
Software Process Root cause analysis results					U	U	U	U	U	U
Software Safety Analysis		P	U	U	B	U				
The defined SA processes for the SA activities on the project per the requirements in the Software Assurance and Safety standard.		P	U	U	B	U				
The list of all software safety-critical components that have been identified by the system hazard analysis.		P	U	U	U	B	U			
The results of SA independent static code analysis results for cybersecurity vulnerabilities and weaknesses.							P	B		
The results of SA independent static code analysis, on the source code, show that the source code follows the defined secure coding practices.							P	B		
The software training records for SA personnel on a project.		P	U	U	B	U				

Legends

Maturity Types

F = Final, D = Draft, P = Preliminary, B = Baseline, U = Updated/Updated as required, X = assume complete (final), not explicit in NPRs

Review Types

MCR = Mission Concept Review,	SRR = System Requirements Review	MDR = Mission Definition Review
SDR = System Definition Review	PDR = Preliminary Design Review	CDR = Critical Design Review
SIR = System Integration Review	TRR = Test Readiness Review	SAR = System Acceptance Review
ORR = Operational Readiness Review		

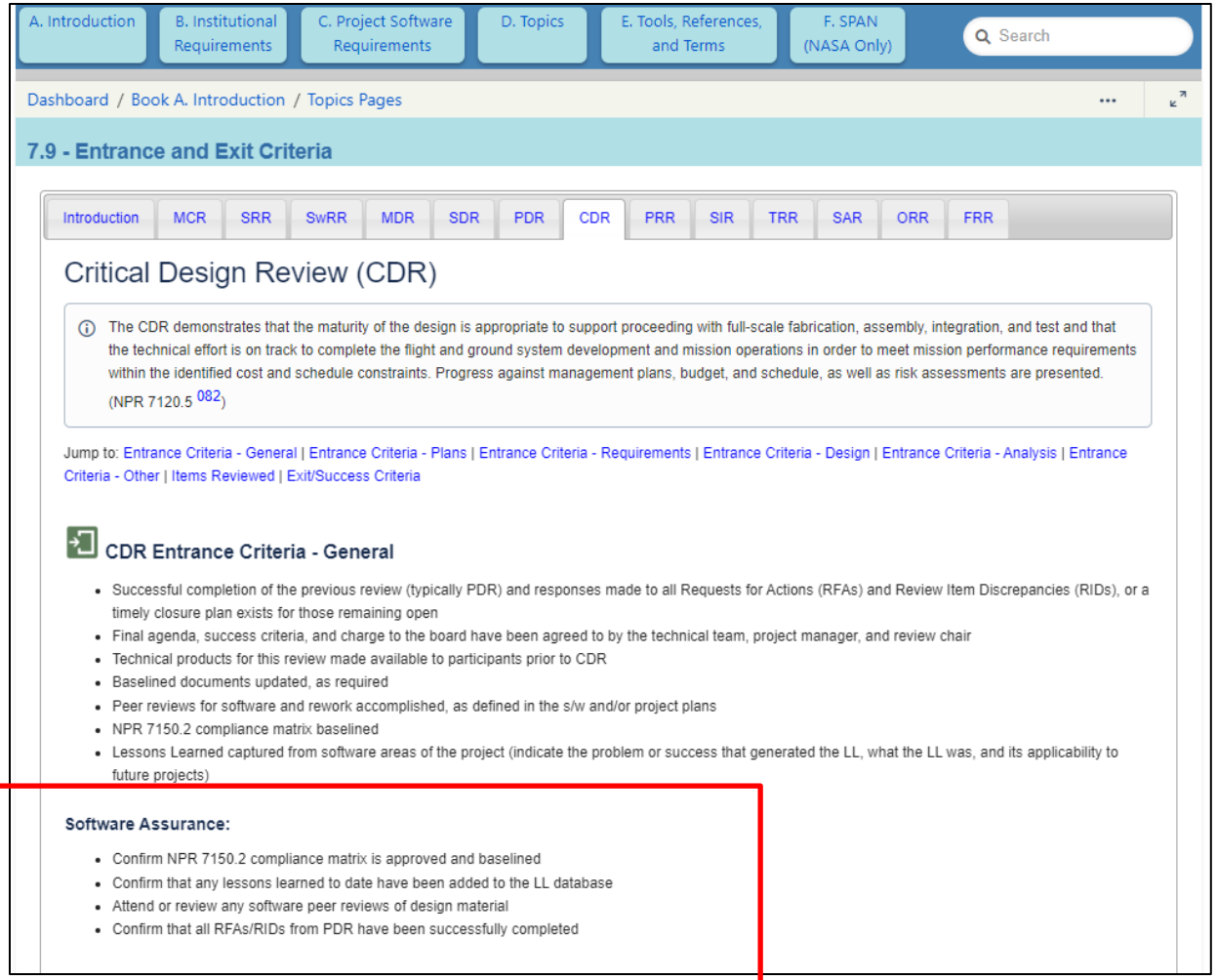
https://swehb.nasa.gov/display/SWEHBVD/7.8+-+Maturity+of+Life+Cycle+Products+at+Milestone+Reviews

https://swehb.nasa.gov/display/SWEHBVD/7.9+-+Entrance+and+Exit+Criteria

Entrance and Exit Criteria

Product reviews

- Defines the responsibilities of the software assurance community throughout the project life cycle reviews.
- Includes reviews and products which are the primary responsibility of the software assurance community as well as software engineering community contributions to system activities and products, such as the Project Plan.
- Note that different mission types (e.g., robotic vs. human) can have different life cycles and, therefore, different sets of life cycle reviews that apply.



Communication

Software Assurance Suggested Metrics

- There are multiple "**Metrics Types**", and each type includes optional "**Measurements**" by life-cycle phase for the "Associated SWE Requirements".
- Projects should choose a set of measurements to provide information on the project being implemented.
- The measurements do not have to be implemented as written.
- The metrics should be modified to best fit the characteristics of the project.

Metrics Type	Measurements	Plan	Reqt	Des	Imp	Test	Del	Associated SWE Reqt #
Peer Review Metrics	· # of peer reviews performed vs. # of peer reviews planned		X	X	X	X	X	SWE-016 SWE-087 SWE-089
	· # of Non-Conformances identified in each peer review		X	X	X	X	X	SWE-087 SWE-089
	· # of Non-Conformances identified by software assurance during each peer review		X	X	X	X	X	SWE-087 SWE-088 SWE-089
	· Total # of peer review Non-Conformances (Open, Closed)		X	X	X	X	X	SWE-087 SWE-088 SWE-089
	· Preparation time each review participant spent preparing for the review		X	X	X	X	X	SWE-088 SWE-089
	· Time required to close review Non-Conformances		X	X	X	X	X	SWE-087 SWE-088 SWE-089
	· # of peer review participants vs. total # invited		X	X	X	X	X	SWE-088 SWE-089
	· # of peer review Non-Conformances per work product vs. # of peer reviewers		X	X	X	X	X	SWE-088 SWE-089
Peer Review Audit Metrics	· # of audit Non-Conformances per peer review audit		X	X	X	X	X	SWE-088
	· # of Peer Review Audits planned vs. # of Peer Review Audits performed		X	X	X	X	X	SWE-016 SWE-087

https://swehb.nasa.gov/display/SWEHBVD/8.18+-+SA+Suggested+Metrics

Root Cause Analysis

- To reduce defects from occurring, we have to understand why the defect or software non-conformance occurred.
- Root Cause Analysis is a structured evaluation method that identifies the root causes of an undesired outcome and the actions adequate to prevent a recurrence.
- Software Assurance should use a method like, Root Cause Analysis as a technique to help the projects identifies the root causes of an undesired outcome
- Root cause analysis can be decomposed into four steps:
  - Identify and describe clearly the problem.
  - Establish a timeline from the normal situation up to the time the problem occurred.
  - Distinguish between the root cause and other causal factors (e.g., using event correlation).
  - Establish a causal graph between the root cause and the problem.

Defect Tracking and Management

Definitions About Root Cause Analysis Cause (Causal Factor)	An event or condition that results in an effect. Anything that shapes or influences the outcome.
Proximate Cause(s)	The event(s) that occurred, including any condition(s) that existed immediately before the undesired outcome, directly resulted in its occurrence and, if eliminated or modified, would have prevented the undesired outcome. Also known as the direct cause(s).
Root Cause(s)	One of the multiple factors (events, conditions, or organizational factors) that contributed to or created the proximate cause and subsequent undesired outcome, if eliminated or modified, would have prevented the undesired outcome. Typically multiple root causes contribute to an undesired outcome.
Root Cause Analysis (RCA)	A structured evaluation method that identifies the root causes of an undesired outcome and the actions adequate to prevent a recurrence. Root cause analysis should continue until organizational factors have been identified, or until data are exhausted.
Event	A real-time occurrence describes one discrete action, typically an error, failure, or malfunction. Examples: pipe broke, power lost, lightning struck, the person opened a valve, etc...
Condition	Any as-found state, whether or not resulting from an event, that may have safety, health, quality, security, operational, or environmental implications.
Organizational Factors	Any operational or management structural entity that exerts control over the system at any stage in its life cycle, including but not limited to the system's concept development, design, fabrication, test, maintenance, operation, and disposal. Examples: resource management (budget, staff, training); policy (content, implementation, verification); and management decisions.
Contributing Factor	An event or condition that may have contributed to the occurrence of an undesired outcome but, if eliminated or modified, would not by itself have prevented the occurrence.
Barrier	A physical device or administrative control is used to reduce the risk of the undesired outcome to an acceptable level. Barriers can provide physical intervention (e.g., a guardrail) or procedural separation in time and space (e.g., lock-out-tag-out procedure).

Software Hazard Causes

- When a device or system can lead to injury, death, the destruction or loss of vital equipment, or damage to the environment, system safety is paramount. The system safety discipline focuses on "hazards" and the prevention of hazardous situations.
- A hazard is the presence of a potential risk situation that can result in or contribute to a mishap. To ensure the system being developed is as safe as possible, it is important to begin identifying potential hazards as early as possible in the development. Thus, the software and system safety personnel generally look at the hazardous events that could happen and what could potentially cause them.
- Every hazard has at least one cause, which in turn can lead to several effects (e.g., damage, illness, failure).
- A hazard cause may be a defect in hardware or software, a human operator error, or an unexpected input or event which results in a hazard. The table below provides several potential software causes to consider in the project when developing the list of hazards and their potential causes.
- Hazard control is a method for preventing the hazard, reducing the likelihood of the hazard occurring, or the reduction of the impact of that hazard. Hazard controls use software (e.g. detection of the stuck valve and automatic response to open secondary valve), hardware (e.g. pressure relief valve), operator procedures, or a combination of methods to avert the hazard. For every hazard cause, there must be at least one control method, usually a design feature (hardware and/or software) or a procedural step.

Software hazard analysis

2. Table of Software Causes

Potential Software Causes to Consider When Identifying Software Causes in Hazard Analysis

Software Cause Areas to Consider	Potential Software Causes
Data errors	1. Asynchronous communications 2. Single or double event upset/bit flip or hardware-induced error 3. Communication to/from an unexpected system on the network 4. An out-of-range input value, a value above or below the range 5. Start-up or hardware initiation data errors 6. Data from an antenna gets corrupted 7. Failure of software interface to memory 8. Failure of flight software to suppress outputs from a failed component 9. Failure of software to monitor bus controller rates to ensure communication with all remote terminals on the bus schedule's avionics buses 10. Ground or onboard database error 14. Missing or failed integrity checks on inputs, failure to check the validity of input/output data 15. Excessive network traffic/babbling node - keeps the network so busy it inhibits communication from other nodes 16. Sensors or actuators stuck at some value 17. Wrong software state for the input
Commanding errors	1. Command buffer error or overflow 2. Corrupted software load

https://swehb.nasa.gov/display/SWEHBVD/8.21+-+Software+Hazard+Causes

Software Safety-Critical

Software Safety Analysis and Hazard Analysis

Software is classified as safety-critical if the software is determined by and traceable to a hazard analysis. Software is classified as safety-critical if it meets at least one of the following criteria:

a. Causes or contributes to a system hazardous condition/event,

b. Controls functions identified in a system hazard,

c. Provides mitigation for a system hazardous condition/event,

d. Mitigates damage if a hazardous condition/event occurs,

e. Detects, reports, and takes corrective action if the system reaches a potentially hazardous state.

The Cartwheel galaxy and its companion galaxies
NASA, ESA, CSA, STScI, Webb ERO Production Team

Primary Safety-Critical Software Requirements

If a project has safety-critical software, the project manager shall implement the safety-critical software requirements contained in NASA-STD-8739.8. [SWE-023]

Safety-critical software requirements contained in NASA-STD-8739.8.

1. **Confirm that the NPR 7150.2 requirement items "a" through "l" are documented in the detailed software requirements.**
2. **Assessment that the source code satisfies the conditions in the NPR 7150.2 requirement "a" through "l" for safety-critical software.**

If a project has safety-critical software or mission-critical software, the project manager shall implement the following items in the software:
a. The software is initialized, at first start and restarts, to a known safe state.
b. The software safely transitions between all predefined known states.
c. Termination performed by software of functions is performed to a known safe state.
d. Operator overrides of software functions require at least two independent actions by an operator.
e. Software rejects commands received out of sequence when execution of those commands out of sequence can cause a hazard.
f. The software detects inadvertent memory modification and recovers to a known safe state.
g. The software performs integrity checks on inputs and outputs to/from the software system.
h. The software performs prerequisite checks prior to the execution of safety-critical software commands.
i. No single software event or action is allowed to initiate an identified hazard.
j. The software responds to an off-nominal condition within the time needed to prevent a hazardous event.
k. The software provides error handling.
l. The software can place the system into a safe state.

Primary Safety-Critical Software Requirements

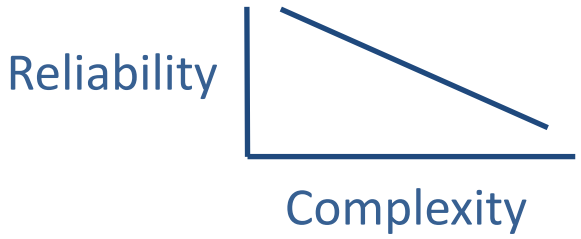
Confirm 100% code test coverage has been achieved or addressed for all identified software critical components

Test Coverage → Find Untested code

Test Coverage → Understand why any untested code exist

$$\text{Test Coverage} = \frac{\text{Number of Lines of Code Called by Test Suite}}{\text{Total Number of Relevant Lines of Code}} \times 100\%$$

Confirm that all identified software safety-critical components have a cyclomatic complexity value of 15 or lower.



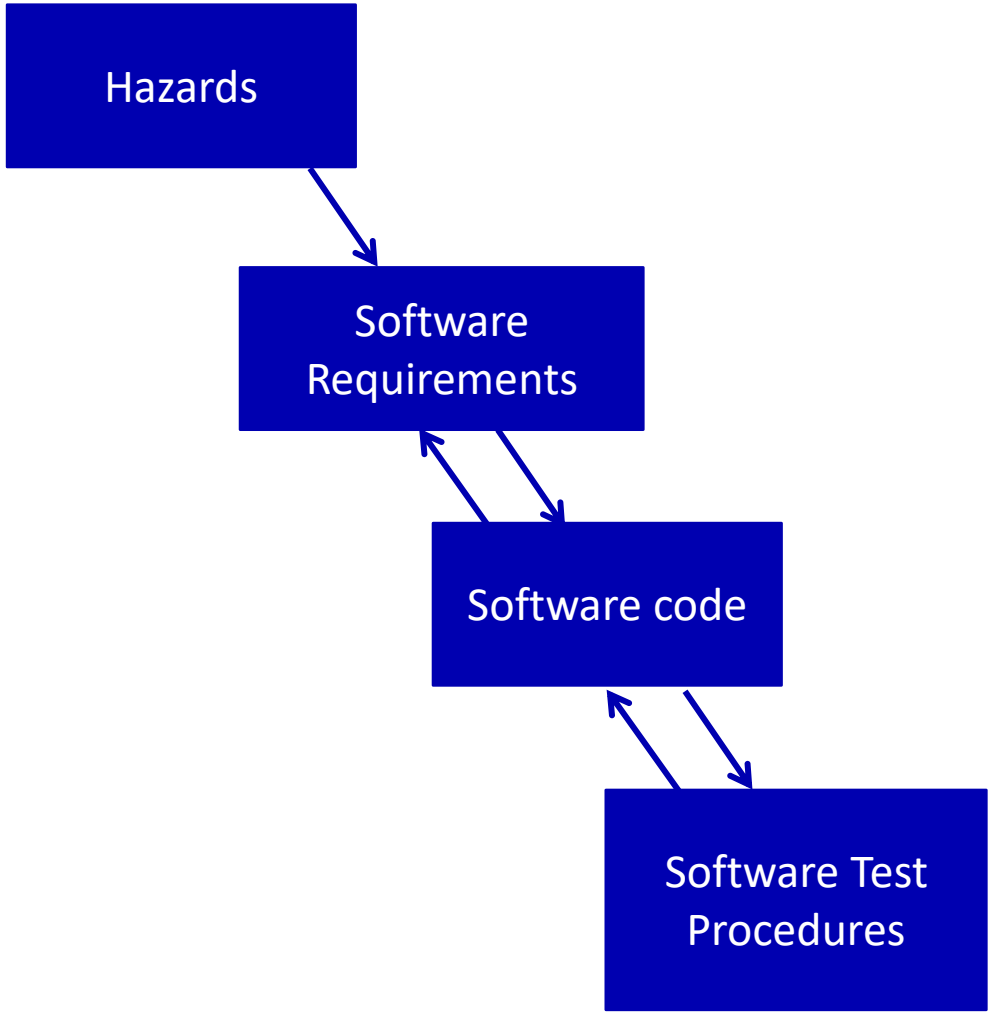
Safety-Critical Software Requirements

Include software related safety constraints, controls, mitigations and assumptions between the hardware, operator, and software in the software requirements documentation.

Verify through test the software requirements that trace to a hazardous event, cause, or mitigation technique.

The project manager shall perform, record, and maintain bi-directional traceability between the following software elements: [SWE-052]

Software requirements to the system hazards



NASA Software Independent Verification and Validation (IV&V) Activities

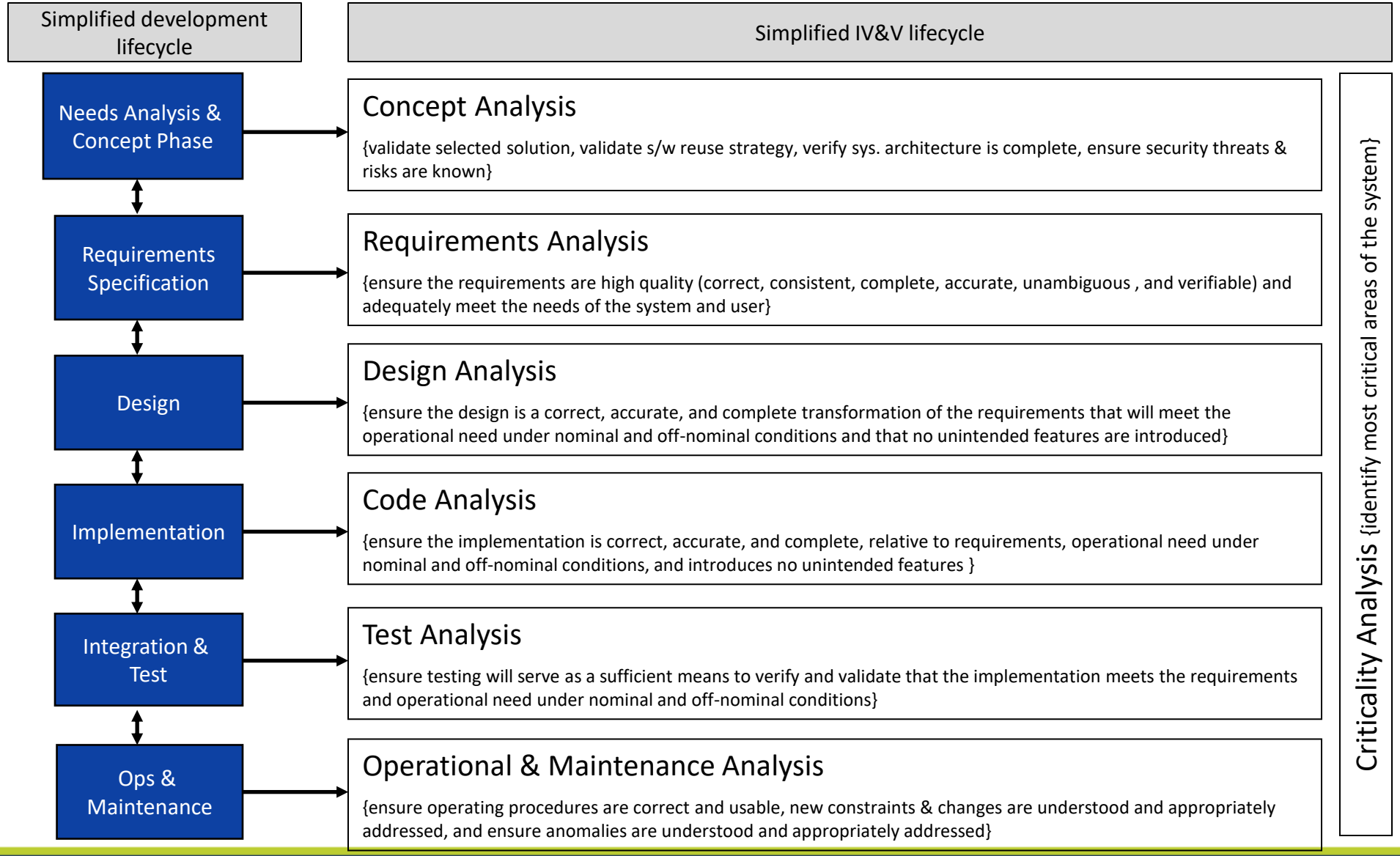
NASA's Independent Verification and Validation (IV&V) Program

- Fairmont, WV
- http://www.nasa.gov/centers/ivv/home/index.html

Introduction to IV&V

- Software Verification and Validation (V&V) is a systems engineering discipline.
  - V&V is more than testing, just like development is more than coding!
- The purpose is to help the development organization build quality into the software during the software life cycle.
  - Some objectives of performing V&V:
    - Facilitate early detection and correction of software errors
    - Enhance management insight into process and product risk
    - Support the software life cycle processes to ensure compliance with program performance, schedule, and budget requirements
- As part of Software Assurance at NASA, and utilizing IEEE standards, IV&V is differentiated from V&V because it is managerially, technically, and financially separated from developers.

Generic Look at IV&V



Determining the Amount of IV&V

- IV&V is conducted across the entire life cycle, BUT NOT on the entire system
  - IV&V can be focused or target just certain development phases, too
- The IV&V Program "scopes" the system to determine areas that warrant analysis
  - The process is called "Portfolio Based Risk Assessment" (PBRA)
  - Results in a risk score for each capability/subsystem for a particular project that enables informed decisions to be made:
    - What parts of the system should IV&V work on
    - How much analytical rigor should we apply (e.g., dynamic analysis should be conducted to thoroughly test the implementation of the protocol used for communications)
- Same approach utilized by organizations to determine which projects within their portfolio of projects warrant additional assurance

Products to Expect from IV&V

- Analyses that provides value added evidence into whether the requirements reflect/capture the user's needs, whether the implementation is reliable, safe, & secure and reflective of these user's needs and whether the testing of the system was adequate
- Confidence & Insight in terms of:
  - Confidence that the system will do what it is supposed to do
  - Confidence that the system will not do what it is not supposed to do
  - Confidence in terms of what/how the system will act/react to/under adverse conditions
- Independent Testing that provides exhaustive execution of hazard domain, failure scenarios, security breaches, duration testing, boundary testing, off nominal testing
- IV&V Project Execution Plans (IPEPs)
  - Documents/guides & communicates IV&V approach to our customers/stakeholders
- Software Risks Identification
  - Identified by IV&V; represent areas of concern/potential for negative consequence(s) for the development Project;
- Technical Issue Memorandums (TIMs)
  - Documents specific instances of problems resulting from analytical efforts
- Technical Analysis Reports
  - Formally documents results of IV&V analysis activities and results; typical reports include requirement validation report(s), test validation report(s), build analysis report(s), implementation analysis report(s) including design and code analysis reports
- Lifecycle Review Presentations/Safety and Mission Success Review (SMSR) Presentations
  - Provides necessary information for key decisions to be made regarding the technical maturity of system software (e.g. 3 questions including areas of risk)

Which Projects Receive IV&V?

[SWE-141] For projects reaching KDP A after the effective date of this directive's revision, the program manager shall ensure that software IV&V is performed on the following categories of projects:

a) Category 1 projects as defined in NPR 7120.5.

b) Category 2 projects as defined in NPR 7120.5 that have Class A or Class B payload risk classification per NPR 8705.4.

c) Projects selected explicitly by the Mission Directorate Associate Administrator (MDAA) to have software IV&V

IV&V Facility

Follow

More Social Media

Home
About NASA's IV&V Program
Education
Doing Business
IV&V Services
JSTAR

News & Events
Research
Dynamic Analysis
IV&V Annual Workshop
IV&V Management System
For IV&V Employees
SAS Procurement
SET Procurement

About Us

NASA's Independent Verification and Validation (IV&V) Facility, home of NASA's IV&V Program. NASA IV&V efforts have contributed to NASA's improved safety record since the program's inception.

Vision, Mission, and Values
Director's Bio

Tweets Follow

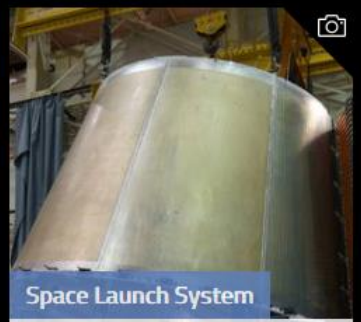
NASA's IV&V Program @NASAIVV — 8 Dec
The results from the #WVFLL State Tournament: facebook.com/wvfll/posts/75...
Expand

NASA's IV&V Program @NASAIVV — 5 Dec
Coming up at 1pm today -- official round matches in gym 1 of

Tweet to @NASAIVV

InSight Mars Lander

Mars Mission Team Addressing Vacuum Leak on Key Science Instrument



Space Launch System

Progress Continues on Test Version of SLS Connection Hardware

Education

NASA's education programs inspire interest in science, technology, engineering and mathematics (STEM) among America's youth and have a positive impact on the number of students who are proficient in STEM and choose to pursue careers in STEM fields. NASA increases the pool of future STEM workers, thus contributing to the workforce of the future by attracting and retaining students in STEM disciplines. With these efforts in STEM education, NASA helps the United States remain globally competitive and sustain a strong national economy.

Education
Educators
Student Opportunities

http://www.nasa.gov/centers/ivv/home/index.html

Software Classifications

NASA-wide software classification structure

These definitions are based on:

(1) usage of the software with or 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.

Note: It is not uncommon for a project to contain multiple separate systems and subsystems having different software classes.

NASA-Wide Software Classifications

Class A	Human-Rated Space Software Systems
Class B	Non-Human Space-Rated Software Systems or Large-Scale Aeronautics Vehicles
Class C	Mission Support Software or Aeronautic Vehicles, or Major Engineering/Research Facility Software
Class D	Basic Science/Engineering Design and Research and Technology Software
Class E	Design Concept, Research, Technology and General Purpose Software
Class F	General Purpose Computing, Business and IT Software

Notes: It is not uncommon for a project to contain multiple systems and subsystems having different software classes.

Software Classification vs. Tailoring

- **Software classification is the first level of tailoring!**
  - Classify software based on the definitions on the previous slide **NOT** the amount of project schedule, funding, manpower, or other resources available.
- Engineering and SMA provide dual Technical Authority chains for resolving classification issues. **The NASA Chief Engineer is the ultimate Technical Authority for software classification disputes concerning definitions in this NPR.**
  - Engineering evaluates the project characteristics and generates the initial software classification.
  - Software assurance can perform an independent software classification, or software assurance can concur with engineering's software classification decision. Software engineering and software assurance technical authorities must agree on the classification of each system and subsystem containing software.
- After classifying the software, software engineering tailors the applicable 7150.2D requirements based on project characteristics.

Tailoring Approach for NPR 7150.2D

Table 2. Requirements Mapping Matrix

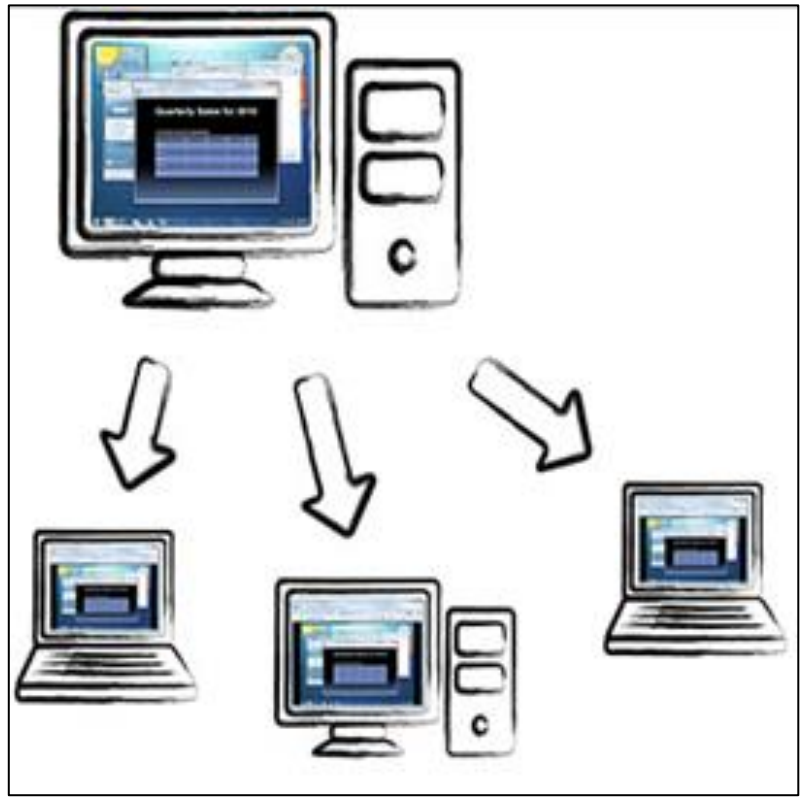
Section	SWE #	Requirement Text	Class A-E Authority	A	B	C	D	E		Class F Authority	F
3.0		**Software Management Requirements**									
3.1		**Software Life Cycle Planning**									
3.1.2	033	The project manager shall assess options for software acquisition versus development.	Center	X	X	X	X	X		CIO	X
3.1.3	013	The project manager shall develop, maintain, and execute software plans, including security plans, that cover the entire software life cycle and, as a minimum, address the requirements of this directive with approved tailoring.	Center	X	X	X	X	X		CIO	X
3.1.4	024	The project manager shall track the actual results and performance of software activities against the software plans. a. Corrective actions are taken, recorded, and managed to closure. b. Changes to commitments (e.g., software plans) that have been agreed to by the affected groups and individuals are taken, recorded, and managed.	Center	X	X	X	X			CIO	X
3.1.5	034	The project manager shall define and document the acceptance criteria for the software.	Center	X	X	X	X			CIO	X
3.1.6	036	The project manager shall establish and maintain the software processes, software documentation plans, list of developed electronic products, deliverables, and list of tasks for the software development that are required for the project's software developers, as well as the action required (e.g., approval, review) of the Government upon receipt of each of the deliverables.	Center	X	X	X	X			CIO	X

"the project manager shall..." means the roles and responsibilities of the project manager may be further delegated within the organization to the scope and scale of the system.

Software Reuse and Internal Sharing

Summary of New Requirements on Internal NASA Software Sharing or Reuse

- Clear rights in the software [SWE-215]
- Keep a list of all contributors to the software product. [SWE-217]
- Conforms to NASA software engineering policy and requirements. [SWE-216]
- Ensure that the software product contains appropriate disclaimer and indemnification provisions [SWE-217]
- Perform the following actions for each type of internal NASA software transfer or reuse: [SWE-214]
  - a. A NASA civil servant to a NASA civil servant
  - b. A NASA civil servant to a NASA contractor
  - c. A NASA civil servant to a foreign person or foreign entity



Sharing has many legal aspects

Software Cybersecurity

3.11 Software Cybersecurity

3.11.1 Software defects are a central and critical aspect of computer security vulnerabilities. Software defects with cybersecurity ramifications include implementation bugs such as buffer overflows and design flaws such as inconsistent error handling.

Note: Software security relies on high-quality code development and testing practices (clean code, modular structure, well-defined interfaces) – anything that reduces error rates and opportunities misinterpretation or error; considers both the development and deployment/operational context for the software; has the ability to rapidly assess, triage, correct, and deploy security-related updates while the software is in deployment/operations.

3.11.2 The project manager shall perform a software cybersecurity assessment on the software components per the Agency security policies and the project requirements, including risks posed by the use of COTS, GOTS, MOTS, OSS, or reused software components. [SWE-156]

3.11.3 The project manager shall identify cybersecurity risks, along with their mitigations, in flight and ground software systems and plan the mitigations for these systems. [SWE-154]

Note: Space Asset or Enterprise Protection Plans are a source of requirements to identify cybersecurity risks, along with their mitigations, in-flight and ground software systems. Space Asset or Enterprise Protection Plans describe the program's approach for planning and implementing the requirements for information, physical, personnel, industrial, and counterintelligence/counterterrorism security, and for security awareness/education requirements in accordance with NPR 1600.1, NPD 1600.2, NPD 2810.1, and NPR 2810.1.

3.11 Software Cybersecurity

3.11.5 The project manager shall test the software and record test results for the required software cybersecurity mitigation implementations identified from the security vulnerabilities and security weaknesses analysis. [SWE-159]

Note: Include assessments for security vulnerabilities during Peer Review/Inspections of software requirements and design. Utilize automated security static analysis as well as coding standard static analyses of software code to find potential security vulnerabilities.

3.11.6 The project manager shall identify, record, and implement secure coding practices. [SWE-207]

3.11.7 The project manager shall verify that the software code meets the project's secure coding standard by using the results from static analysis tool(s). [SWE-185]

3.11.8 The project manager shall identify software requirements for the collection, reporting, and storage of data relating to the detection of adversarial actions. [SWE-210]

3.11.4 The project manager shall implement protections for software systems with communications capabilities against unauthorized access per the requirements contained in the NASA-STD-1006, Space System Protection Standard. [SWE-157]

NASA-STD-1006, Space System Protection Standard Requirements

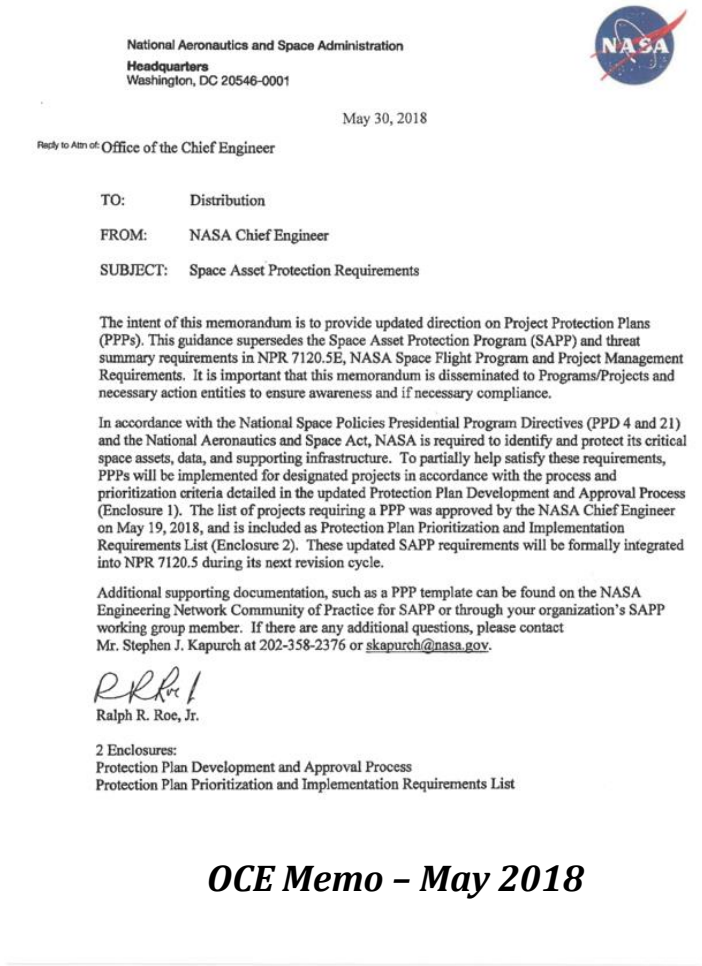
- Command Stack Protection
  - [SSPR 1] Programs/projects shall protect the command stack with encryption that meets or exceeds the Federal Information Processing Standard (FIPS) 140, Security Requirements for Cryptographic Modules, Level 1
- Backup Command Link Protection
  - [SSPR 2] If a project uses an encrypted primary command link, any backup command link shall, at a minimum, use authentication.
- Command Link Critical Program/Project Information (CPI)
  - [SSPR 3] The program/project shall protect the confidentiality of command link CPI as controlled unclassified information (CUI) to prevent inadvertent disclosure to unauthorized parties.
- Ensure Positioning, Navigation, and Timing (PNT) Resilience
  - [SSPR 4] If project-external PNT services are required, projects shall ensure that systems are resilient to the complete loss of, or temporary interference with, external PNT services.
- Interference Reporting
  - [SSPR 5] Projects/Spectrum Managers/Operations Centers shall report unexplained interference to MRPP or to other designated notifying organizations.
- Interference Reporting Training
  - [SSPR 6] Projects/Spectrum Managers/Operations Centers shall conduct proficiency training for reporting unexplained interference.

Project Protection Plan (PPP) Requirement

- NPR 7120.5E requires all flight programs/projects develop Threat Summaries and Project Protection Plans (PPP)
  - Develop program Threat Summary to address classified threat information (TS/SCI)
  - Develop PPP to recommend potential mitigations (SECRET)
  - Baseline by PDR

Current Space Asset Protection Policy

- Updated direction for PPPs established in Office of Chief Engineer (OCE) memo
  - Supersedes NPR 7120.5E requirement for Project Protection Plans (PPPs)
  - Memos expected annually until NPR 7120.5 is updated (schedule TBD)
- Memo and appendices define PPP
  - Establishes approval authority
  - Defines key elements of PPPs
  - Lists projects requiring PPPs; based on Agency Mission Program/Project List (AMPL)
- Candidate Protection Strategies (CPS) and PPP template
  - Posted on SAPP Community of Practice website
  - PPPs archived in classified web-portal
  - https://nen.nasa.gov/web/sap



National Aeronautics and Space Administration
Headquarters
Washington, DC 20546-0001

May 30, 2018

Reply to Attn of: Office of the Chief Engineer

TO: Distribution

FROM: NASA Chief Engineer

SUBJECT: Space Asset Protection Requirements

The intent of this memorandum is to provide updated direction on Project Protection Plans (PPPs). This guidance supersedes the Space Asset Protection Program (SAPP) and threat summary requirements in NPR 7120.5E, NASA Space Flight Program and Project Management Requirements. It is important that this memorandum is disseminated to Programs/Projects and necessary action entities to ensure awareness and if necessary compliance.

In accordance with the National Space Policies Presidential Program Directives (PPD 4 and 21) and the National Aeronautics and Space Act, NASA is required to identify and protect its critical space assets, data, and supporting infrastructure. To partially help satisfy these requirements, PPPs will be implemented for designated projects in accordance with the process and prioritization criteria detailed in the updated Protection Plan Development and Approval Process (Enclosure 1). The list of projects requiring a PPP was approved by the NASA Chief Engineer on May 19, 2018, and is included as Protection Plan Prioritization and Implementation Requirements List (Enclosure 2). These updated SAPP requirements will be formally integrated into NPR 7120.5 during its next revision cycle.

Additional supporting documentation, such as a PPP template can be found on the NASA Engineering Network Community of Practice for SAPP or through your organization's SAPP working group member. If there are any additional questions, please contact Mr. Stephen J. Kapurch at 202-358-2376 or skapurch@nasa.gov.

Ralph R. Roe, Jr.

2 Enclosures:
Protection Plan Development and Approval Process
Protection Plan Prioritization and Implementation Requirements List

OCE Memo – May 2018

Candidate Protection Strategies (CPS)

- Serve as a starting point for mission protection planning
- Linked to consistent high threat and risk issues
- Protection plans incorporate results of the CPS analysis, including any requisite requirement tailoring

Main CPS Categories

1. Engineering Focused Strategies – Space Segment (3)
2. Engineering Focused Strategies – Ground Segment (2)
3. Engineering Focused – All Segments (2)
4. ConOps Focused Strategies (6)
5. Cybersecurity Strategies

Software Engineering Lifecycles

Software Life Cycle Planning

- Software life cycle planning covers the software aspects of a project from inception through retirement.
- The software life cycle planning cycle is an organizing process that considers the software as a whole and provides the planning activities required to ensure a coordinated, well-engineered process for defining and implementing project activities.
- These processes, plans, and activities are coordinated within the project. At project conception, software needs for the project are analyzed, including acquisition, supply, development, operation, maintenance, retirement, and supporting activities and processes.
- The software effort is scoped and the processes, measurements, and activities are documented in software plan(s).
- NASA Software Engineering NPR makes no recommendation for a specific software life-cycle model (i.e., it allows agile, incremental, spiral, etc., life-cycle models). However, expectations from the system project life- cycle models need to be adequately addressed in the software plan(s).

3.1.3 The project manager shall develop, maintain, and execute software plans that cover the entire software life cycle and, as a minimum, address the requirements of this directive with approved tailoring. [SWE-013]

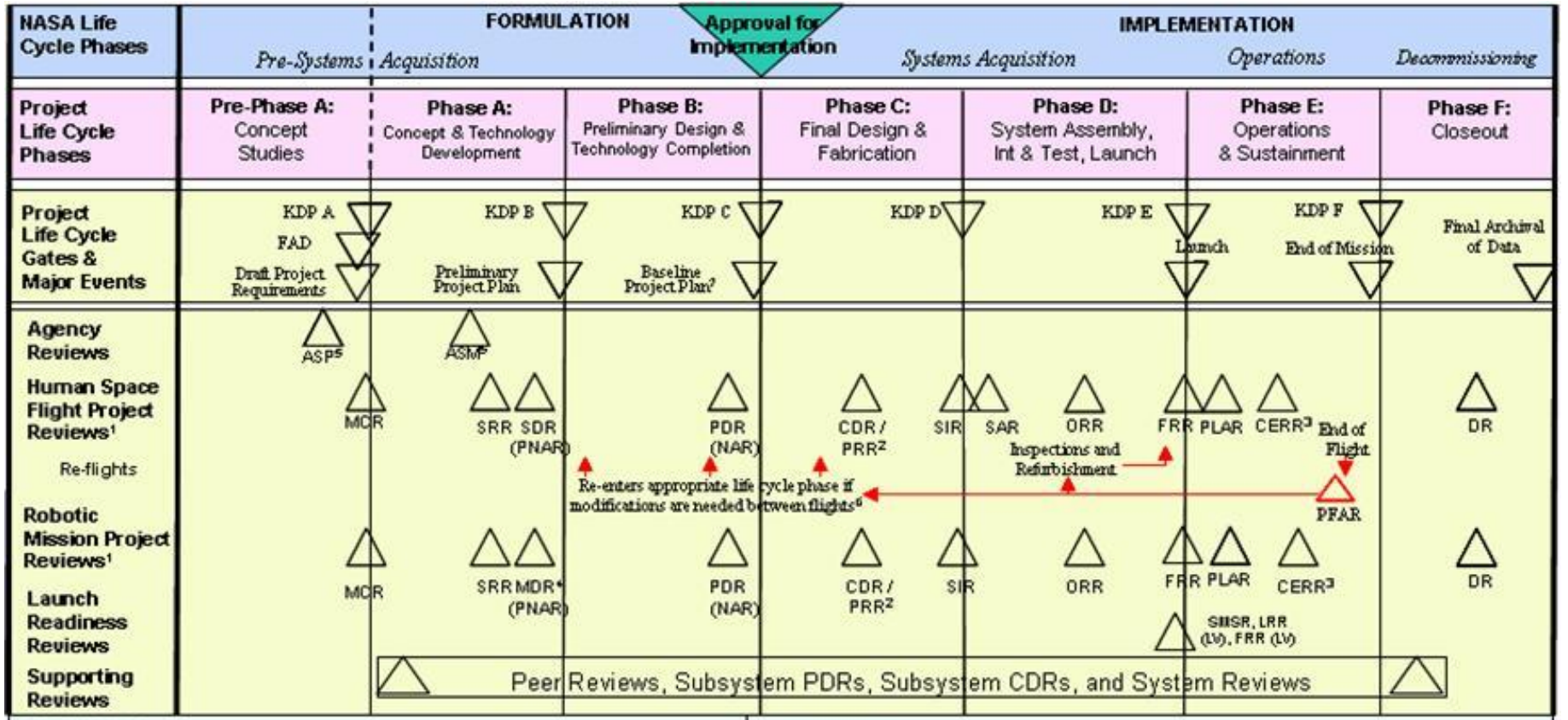
Note: The recommended practices and guidelines for the content of different types of software planning activities (whether stand-alone or condensed into one or more project level or software documents or electronic files) are defined in NASA-HDBK-2203. The project should include or reference in the software development plans procedures for coordinating the software development and the design and the system or project development life cycle.

3.1.4 The project manager shall track the actual results and performance of software activities against the software plans. [SWE-024]

a. Corrective actions are taken, recorded, and managed to closure.

b. Including changes to commitments (e.g., software plans) that have been agreed to by the affected groups and individuals.

Project Life Cycle



FOOTNOTES

1. Flexibility is allowed in the timing, number, and content of reviews as long as the equivalent information is provided at each KDP and the approach is fully documented in the Project Plan. These reviews are conducted by the project for the independent SRB. See Section 2.5 and Table 2-6.
2. PRR needed for multiple (≥4) system copies. Timing is notional.
3. CERRs are established at the discretion of Program Offices.
4. For robotic missions, the SRR and the MDR may be combined.
5. The ASP and ASM are Agency reviews, not life-cycle reviews.
6. Includes recertification, as required.
7. Project Plans are baselined at KDP C and are reviewed and updated as required, to ensure project content, cost, and budget remain consistent.

ACRONYMS

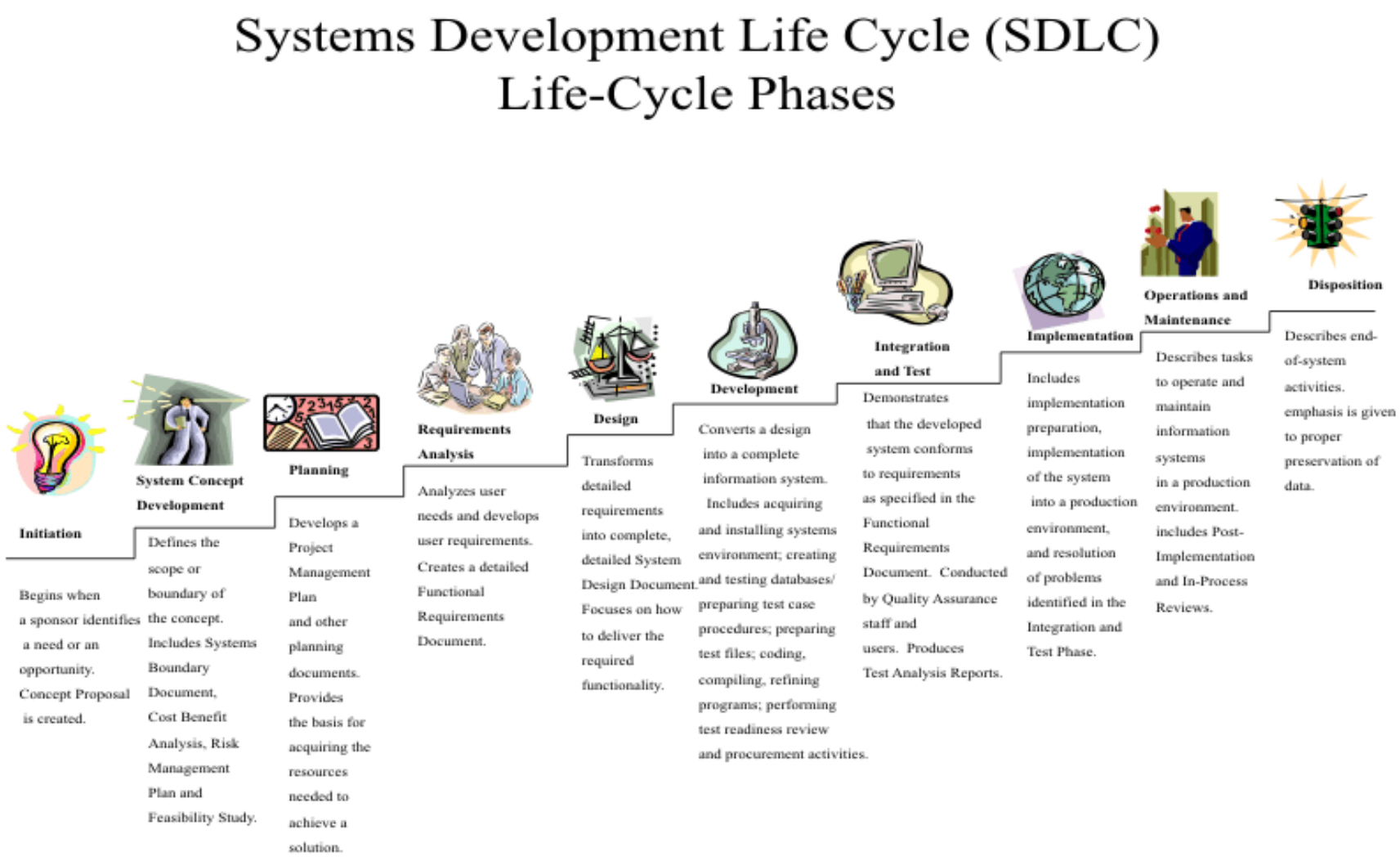
ASP—Acquisition Strategy Planning Meeting
ASM—Acquisition Strategy Meeting
CDR—Critical Design Review
CERR—Critical Events Readiness Review
DR—Decommissioning Review
FAD—Formulation Authorization Document
FRR—Flight Readiness Review
KDP—Key Decision Point
LRR—Launch Readiness Review
MCR—Mission Concept Review
MDR—Mission Definition Review
NAR—Non-Advocate Review
ORR—Operational Readiness Review
PDR—Preliminary Design Review
PFAR—Post-Flight Assessment Review
PLAR—Post-Launch Assessment Review
PNAR—Preliminary Non-Advocate Review
PRR—Production Readiness Review
SAR—System Acceptance Review
SDR—System Definition Review
SIR—System Integration Review
SMSR—Safety and Mission Success Review
SRR—System Requirements Review

From NPR 7150.2

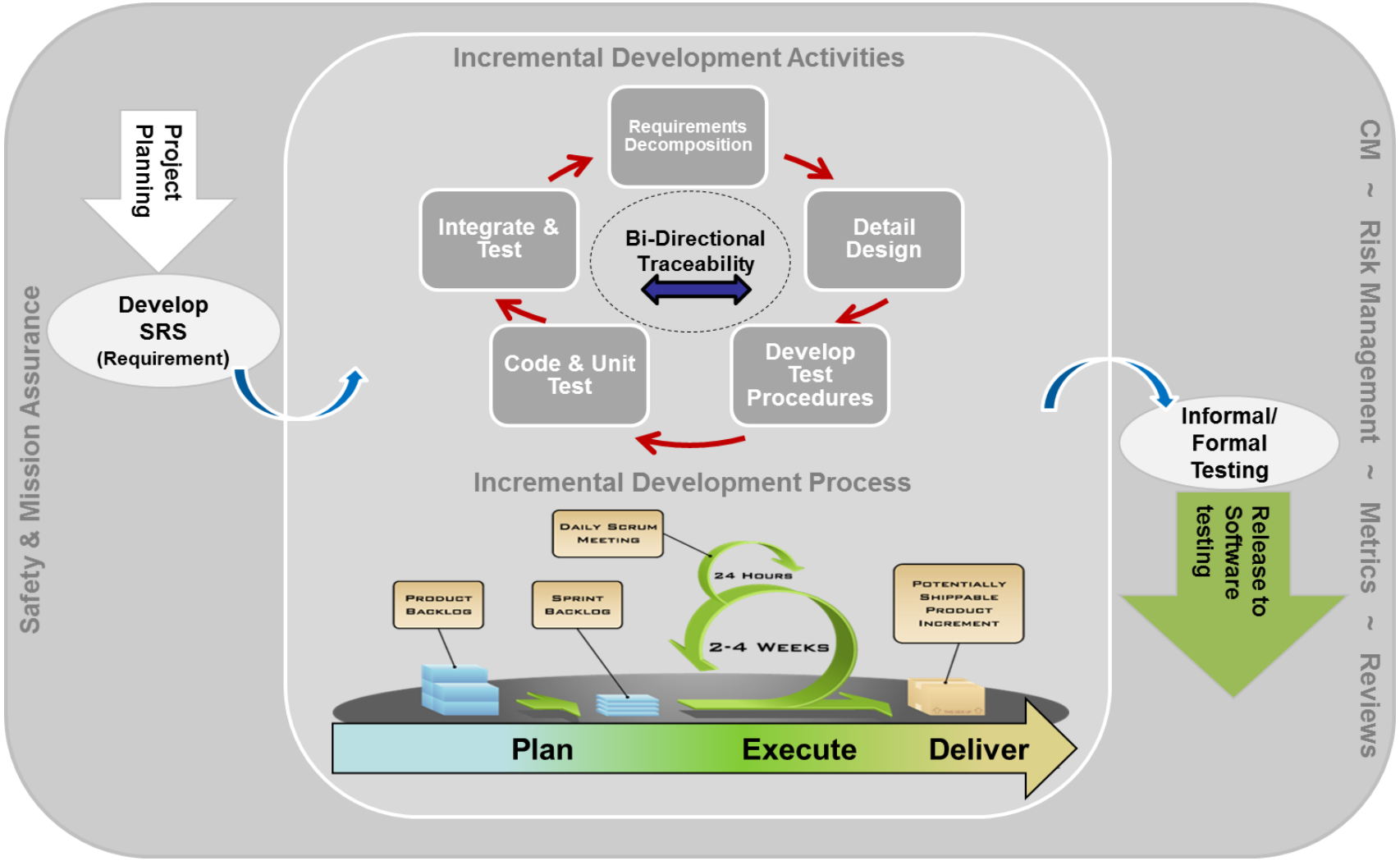
"This NPR makes no recommendation for a specific software life-cycle model. Each has its strengths and weaknesses, and no one model is best for every situation. Whether using the spiral model, the iterative model, waterfall, or any other development life-cycle model, each has steps of requirements, design, implementation, testing, release to operations, maintenance, and retirement…"

Frequently Discussed Lifecycles ….

- Waterfall
- Incremental Development
- Spiral Development
- Package-Based Development
- Agile Development
- Legacy System Maintenance

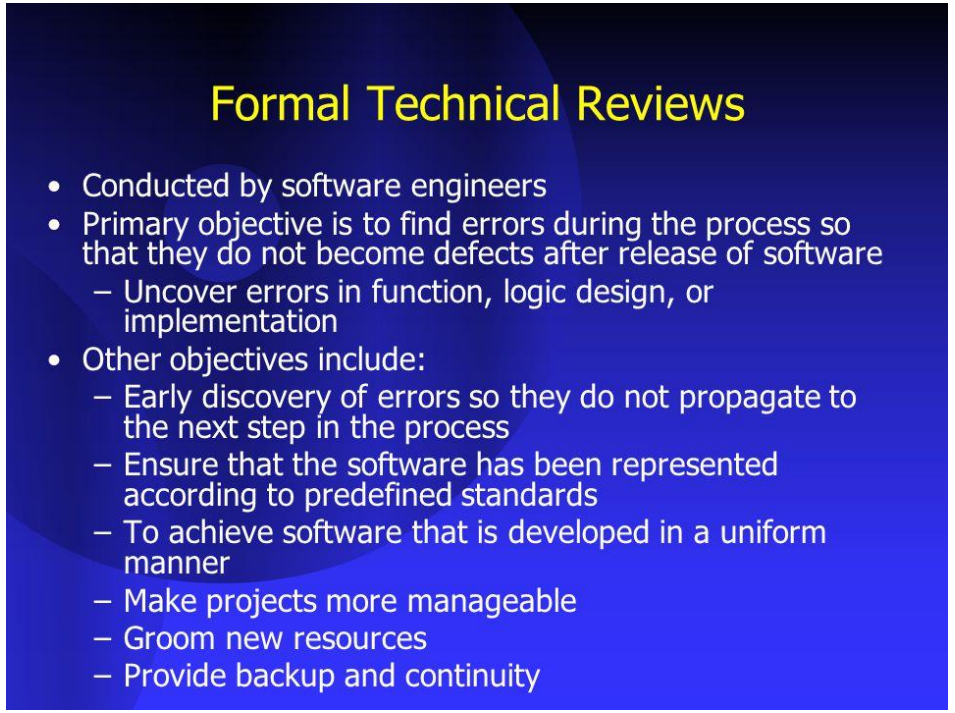


"Agile" Based Incremental Software Development Approach



All Project Lifecycles are "Punctuated" with Formal Technical Reviews

- **Evaluations of the project, or element thereof, by a knowledgeable group for the purposes of:**
  - **Assessing the status of and progress toward accomplishing the planned activities**
  - **Validating the technical tradeoffs explored and design solutions proposed**
  - **Identifying technical weaknesses or marginal design and potential problems (risks), and recommending improvements and corrective actions**
  - **Making judgments on the activities' readiness for the follow-on events to improve the likelihood of a successful outcome**
  - **Making assessments and recommendations to the project team, Center, and Agency management**
  - **Providing a historical record of decisions that were made during these formal reviews for future reference**
  - **Assessing the technical risk status and current risk profile**



Software Life cycle products and their maturity level at the various software project life cycle reviews (Part 1 of 2)

This chart summarizes current guidance approved by the NASA Office of the Chief Engineer (OCE) for software engineering life cycle products and their maturity level at the various software project life cycle reviews.

This chart serves as guidance only and NASA Center procedures should take precedence for projects at those Centers.

F = Final,
D = Draft,
P = Preliminary,
B = Baseline,
U = Updated/Updated as required,
X = assume complete (final), not explicit in NPRs

7150.2 Software Life-Cycle Products	MCR	SRR	MDR	SDR	PDR	CDR	SIR	TRR	SAR	ORR
Software Development Plan (SDP) / Software Management Plan (SMP)		P	P	B		U				
Software Schedule	D	P	U	U	B	U				
Software Cost Estimate	D	P	U	U	B	U				
Software Configuration Management Plan (SCMP)		P	P		B	U				
Software Test Plans					P	B	U	U		
Software Test Procedures						P		B		
Software Test Reports									F	
Software Maintenance Plan						D	P	P	B	U
Software Requirements Specification (SRS)		P			B	U		U		
Requirements on OTS s/w		P			B	U				
Software Data Dictionary					P	B				
Software Design Description (Architectual Design)					B	U		U		
Software Design Description (Detailed Design)					P	B		U		
Interface Design Description					P	B		U		
Software User's Manual (SUM)										B
Records of Continuous Risk Management	P	U	U	U	U	U		U	U	
Measurement Analysis Results					X	X				
Operational Concepts (part of "Mission Operations Concept" or separate)		P	U	U	B	U				
Record of trade-off criteria & assessment (make / buy decision)					X	X				
Acceptance Criteria and Conditions					P	B				

https://swehb.nasa.gov/display/7150/7.8+-+Maturity+of+Life+Cycle+Products+at+Milestone+Reviews

Software Life cycle products and their maturity level at the various software project life cycle reviews (Part 2 of 2)

This chart summarizes current guidance approved by the NASA Office of the Chief Engineer (OCE) for software engineering life cycle products and their maturity level at the various software project life cycle reviews.

This chart serves as guidance only and NASA Center procedures should take precedence for projects at those Centers.

F = Final,
D = Draft,
P = Preliminary,
B = Baseline,
U = Updated/Updated as required,
X = assume complete (final), not explicit in NPRs

Software Assurance and Software Safety	MCR	SRR	MDR	SDR	PDR	CDR	SIR	TRR	SAR	ORR
Software Assurance and Software Plan(s)		P	P	P	B	U				
Software Process Root cause analysis results					U	U	U	U	U	U
SA analysis showing uncovered software code percentage						P	U	U	U	U
SA audit and status reports		U	U	U	U	U	U	U	U	U
SA schedule	D	P	U	U	B	U				
Requirements mapping table for the SA requirements.				P	B	U	U	U		
Cost estimate for the project's SA support.	D	P	U	U	B	U				
Analysis showing software requirement and hazard control coverage			D	P	B	U	U	U	U	
SA Product Acceptance Criteria and Conditions					P	B				
The defined SA processes for the SA activities on the project per the requirements in the Software Assurance and Safety standard.		P	U	U	B	U				
The software training records for SA personnel on a project.		P	U	U	B	U				
The list of all software safety-critical components that have been identified by the system hazard analysis.		P	U	U	U	B	U			
The results of SA independent static code assessment results for cybersecurity vulnerabilities and weaknesses.							P	B		
The results of SA independent static code assessment, on the source code, showing that the source code follows the defined secure coding practices.							P	B		
SA metric analysis procedures.		P	U	U	B	U				

https://swehb.nasa.gov/display/7150/7.8+-+Maturity+of+Life+Cycle+Products+at+Milestone+Reviews

Benefits

- Help increase probability of mission success
- Help ensure that all tasks and deliverables are managed and achieved
- Issues presented or discovered during these activities are communicated to appropriate personnel
- The tracking of these issues to closure ensures that errors and shortcomings in the requirements, architecture, design and/or build of the software are corrected and prevented from reoccurring.
- Keep project stakeholders informed

Reviews

MCR
SRR
SwRR
MDR
SDR
PDR
CDR
PRR
SIR
TRR
SAR
ORR
FRR

NASA-HDBK-2203, Topic 7.9

Introduction | MCR | SRR | SwRR | MDR | SDR | PDR | CDR | PRR | SIR | TRR | SAR | ORR | FRR

Entrance and Exit Criteria

Background

This guidance provides the maximum set of life cycle review entrance and exit criteria for software projects and should be tailored for the project class.

This guidance is a summarized collection of material from the following core documents: NPR 7123.1, Appendix G [The license could not be verified: License Certificate has expired!" onclick="scrollTo(0,0)">082-T73]; NPR 7120.5 [The license could not be verified: License Certificate has expired!" onclick="scrollTo(0,0)">082]; and Center Procedures.

This guidance includes three types of information for each review:

1. Entrance criteria - Activities and products that are to be completed before the review can begin.
2. Materials for the Review - Items to be reviewed during review and used to confirm exit criteria; this information is typically available a couple of weeks prior to the review.
3. Exit criteria – Decisions and actions to be completed before the review is considered complete.

This guidance is focused on the responsibilities of the software engineering community throughout the project life cycle reviews. Therefore, the guidance includes reviews and products which are the primary responsibility of the software engineering community as well as software engineering community contributions to system activities and products, such as the Project Plan.

Note that different mission types (e.g., robotic vs. human) can have different life cycles and, therefore, different sets of life cycle reviews which apply.

This material considers a software project to be a system of systems as well as a single subsystem within the larger project. "System of systems" refers to a software project that includes software subsystems that perform functions allocated to them. Just as a project allocates requirements to hardware, software, external components, etc., software projects allocate software requirements to software subsystems.

This material has been reviewed by the Software Working Group and the Office of the Chief Engineer.

For each review point examples of:

1. **Entrance Criteria**
2. **Items Reviewed**
3. **Exit Criteria**

What does the Systems NPR 7123 state for Software

The PDR demonstrates that the preliminary design meets all system requirements with acceptable risk and within the cost and schedule constraints and establishes the basis for proceeding with detailed design.

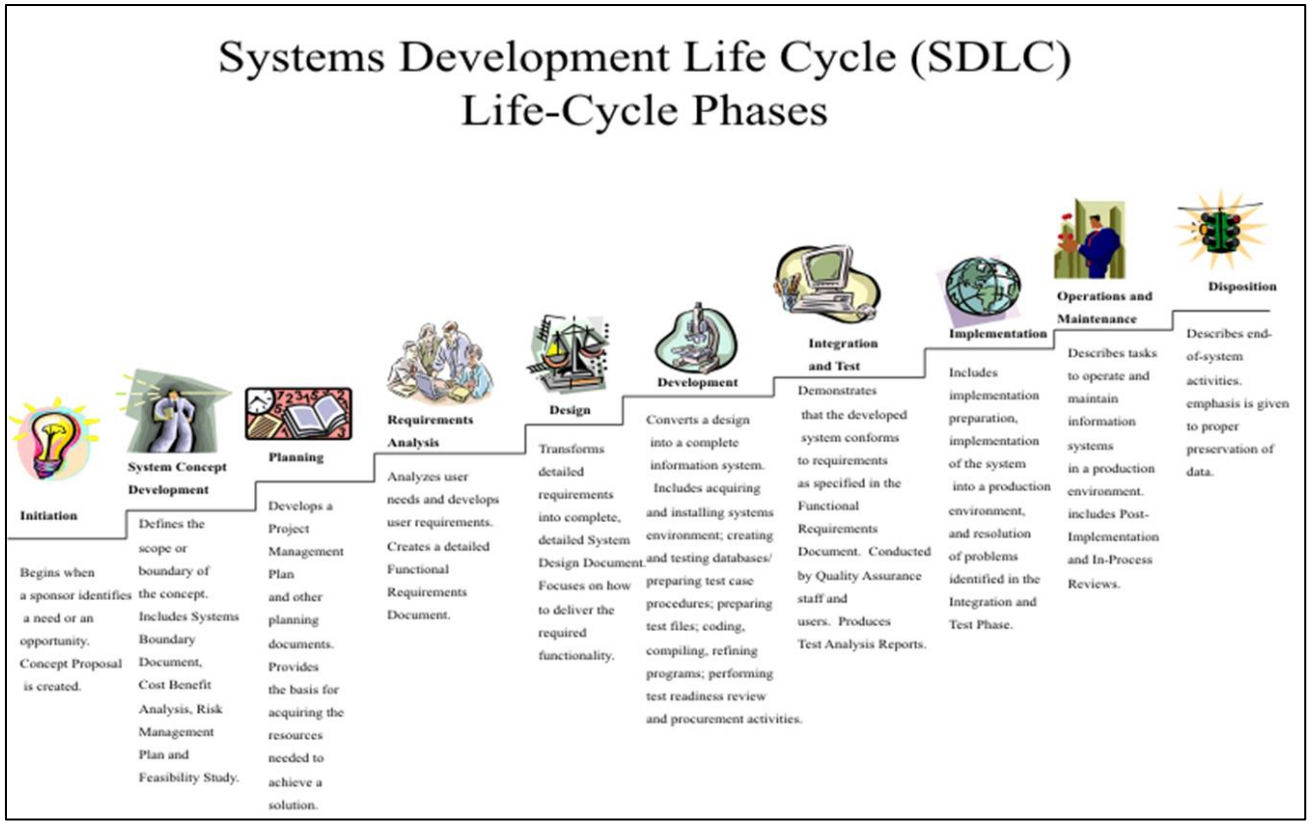
Table G-6 - PDR Entrance and Success Criteria

Preliminary Design Review	
Entrance Criteria	**Success Criteria**
1. The Project has successfully completed the previous planned milestone reviews, and responses have been made to all RFAs and RIDs, or a timely closure plan exists for those remaining open. 2. A preliminary PDR agenda, success criteria, and instructions to the review board have been agreed to by the technical team, project manager, and review chair prior to the PDR.	1. The top-level requirementsÃ¢?"including mission success criteria, TPMs, and any sponsor-imposed constraintsÃ¢?"are agreed upon, finalized, stated clearly, and consistent with the preliminary design. 2. The flow down of verifiable requirements is complete and proper or, if not, an
Package. s. Software criteria and products, per NASA-HDBK-2203, NASA Software Engineering Handbook.	included in design. 18. Software components meet the exit criteria defined in NASA-HDBK-2203, NASA Software Engineering Handbook.

* Product is required for programs/projects covered by NPR 7120.5. If there is disagreement

Summary for Lifecycles and Reviews

- Know the requirements of NPR 7150.2 and how they apply to your project
- Select a lifecycle that is appropriate to your schedule and the nature of the software system that you are building
- Make sure that you understand
  - What lifecycle you are using and the risks associated with the lifecycle selection
  - What should be done or reviewed during each stage of the lifecycle



- Waterfall
- Incremental Development
- Spiral Development
- Package-Based Development
- Agile Development
- Legacy System Maintenance

Class Plan

Software's Role and Importance in NASA Missions

ISWE

NASA Software Engineering & Assurance Policies, Requirements and Resources

Software Planning Requirements and Considerations

- Software Documentation
- Software Costing
- Software Processes
- Software Assurance
- Software Safety-Critical
- Software IV&V
- Software Classifications
- Software Reuse and Internal Sharing
- Software Cybersecurity
- Software Lifecycles and Reviews

Software Life-cycle Requirements

- Software Requirements
- Software Architecture
- Software Design
- Software Coding
- Software Testing
- Software Maintenance

Software Development Supporting Requirements

- Software Configuration Management
- Software Risks
- Software Peer Reviews
- Software Measurements
- Software Defect Management
- Software Bi-Directional Traceability
- Software License Management
- Software Acquisition

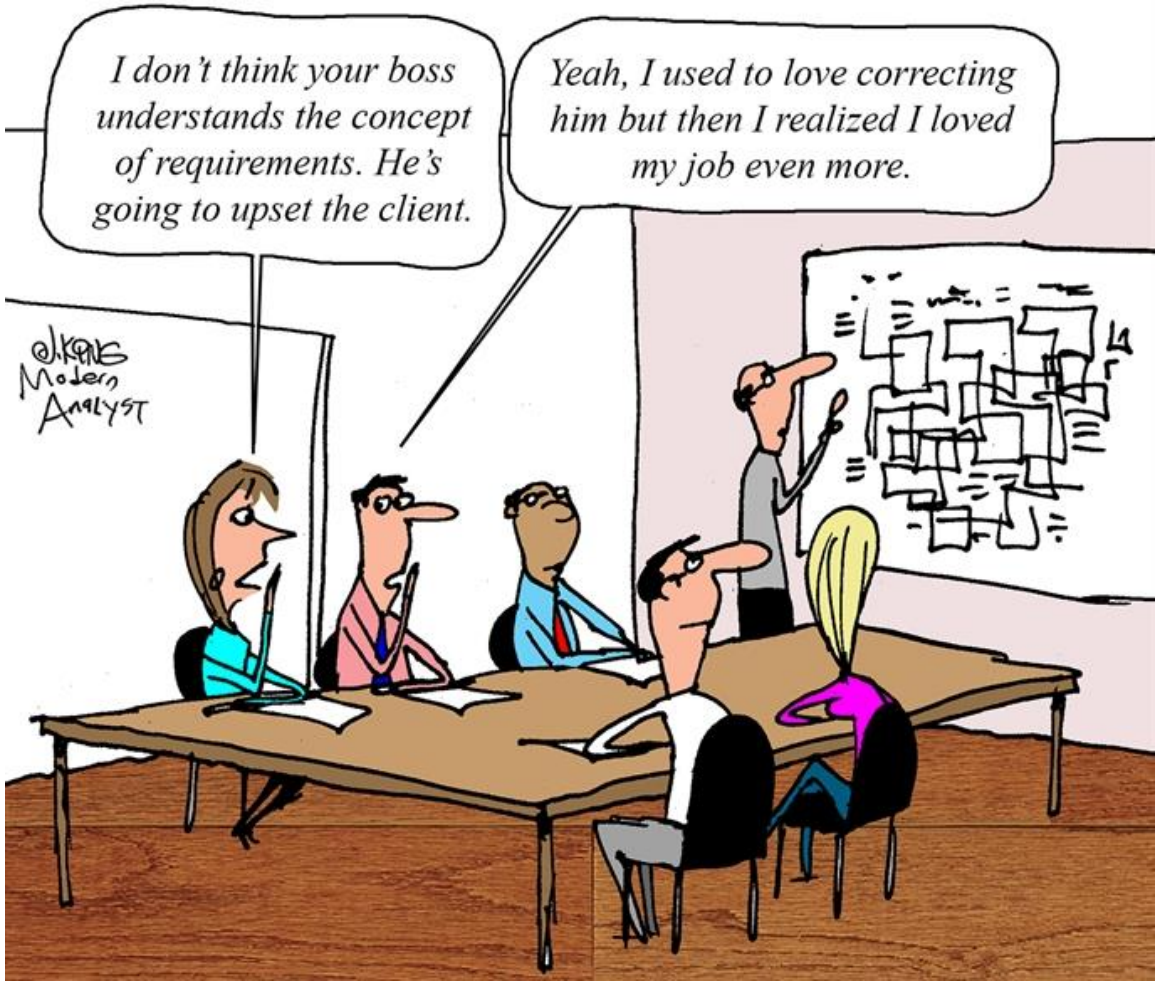
Why do we do these things? Software Failures

Software Requirements

"Walking on water and developing software from a specification are easy if both are frozen."

- Edward V Berard

Requirement Development



NPR 7150.2D Requirements on Software Requirements

- 4.1.2 The project manager shall **establish, capture, record, approve, and maintain software requirements**, including requirements for COTS, GOTS, MOTS, OSS or reused software components, as part of the technical specification. [SWE-050]

- 4.1.3 The project manager shall **perform software requirements analysis** based on flowed-down and derived requirements from the top-level systems engineering requirements, safety and reliability analyses, and the hardware specifications and design. [SWE-051]

- 4.1.4 The project manager shall **include software related safety constraints, controls, mitigations and assumptions** between the hardware, operator, and software in the software requirements documentation. [SWE-184]

NPR 7150.2D Requirements on Software Requirements

- 4.1.5 The project manager shall **track and manage changes** to the software requirements. [SWE-053]

- 4.1.6 The project manager shall identify, initiate corrective actions, and **track until closure inconsistencies among requirements, project plans, and software products.** [SWE-054]

- 4.1.7 The project manager shall **perform requirements validation** to ensure that the software will perform as intended in the customer environment. [SWE-055]

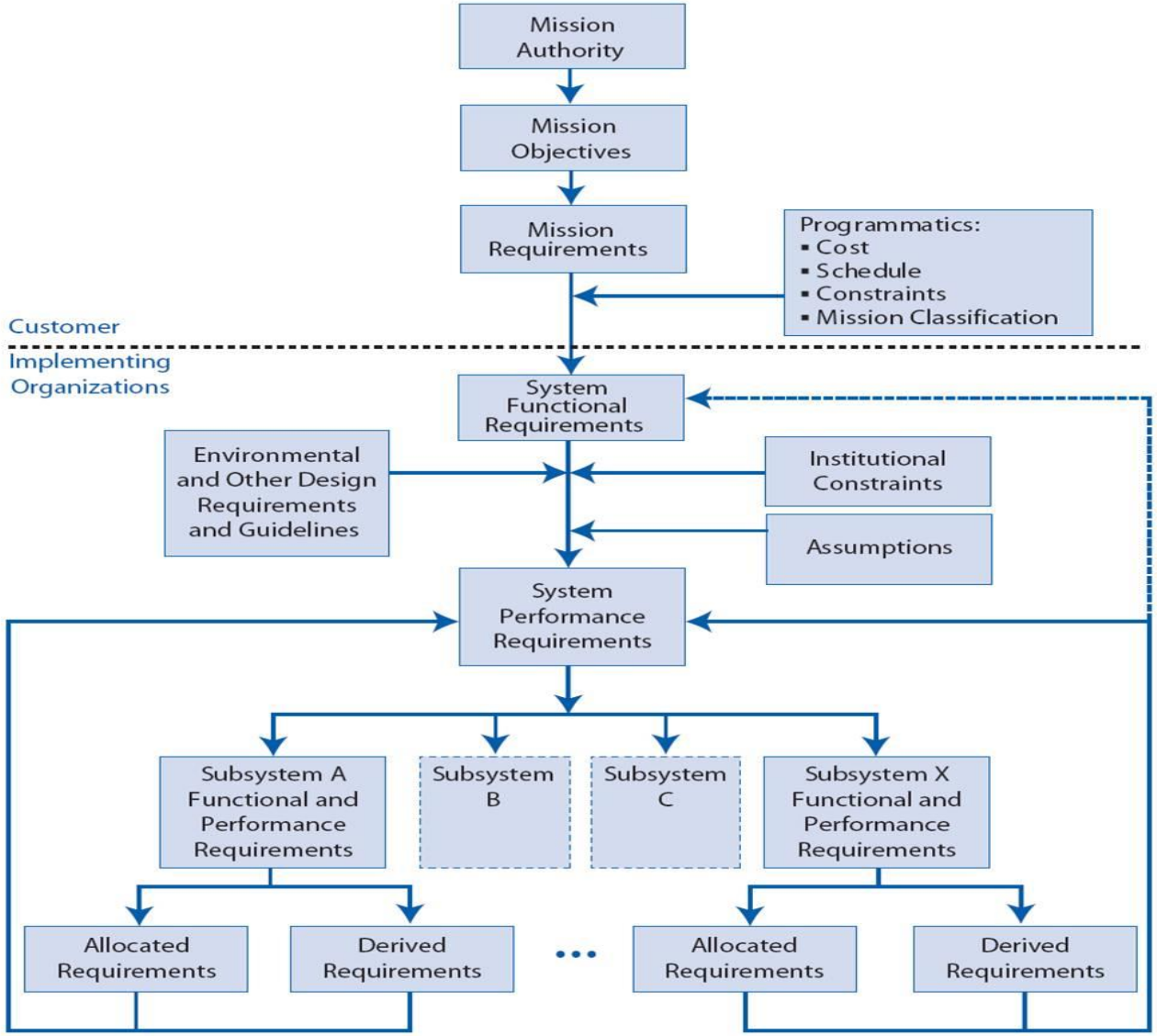
- 5.4.6 The project manager shall collect, track, and report **software requirements volatility metrics.** [SWE-200]

  Note: Software requirements volatility metrics are the total number of requirements compared to requirement changes over time. It may include additions, changes, and reduction of requirements.

Software Requirements

- Software Requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software.
- What requirements do you need to develop a component of software?
- What is the system requirement vs hardware requirement vs operational requirement vs software requirement split?

Flow Down of Requirements



Software Requirement Sources

Other Software Requirement Sources

Hardware specifications
Computer\Processor\Programmable Logic Device specifications
Hardware interfaces
Operating system requirements and board support packages
Data\File definitions and interfaces
Communication interfaces including bus communications **Software interfaces**
Derived from Domain Analysis
Fault Detection, Isolation and Recovery requirements
Models
Commercial Software interfaces and functional requirements
Software Security Requirements
User Interface Requirements
Algorithms
Legacy or Reuse software requirements
Derived from Operational Analysis
Prototyping activities
Interviews
Surveys
Questionnaires
Brainstorming
Observation
Software Test Requirements
Software Fault Management Requirements
Hazard Analysis

System Requirements

Software Requirements

Guidelines for the Software Requirements Specification Content

The Software Requirements Specification shall contain:

a) System overview.

b) CSCI requirements:
   - (1) Functional requirements.
   - (2) Required states and modes.
   - (3) External interface requirements.
   - (4) Internal interface requirements.
   - (5) Internal data requirements.
   - (6) Adaptation requirements (data used to adapt a program to a given installation site or to given conditions in its operational environment).
   - (7) Safety requirements.
   - (8) Performance and timing requirements.
   - (9) Security and privacy requirements.
   - (10) Environment requirements.
   - (11) Computer resource requirements:
     - (a) Computer hardware resource requirements, including utilization requirements.
     - (b) Computer software requirements.
     - (c) Computer communications requirements.
   - (12) Software quality characteristics.
   - (13) Design and implementation constraints.
   - (14) Personnel-related requirements.
   - (15) Training-related requirements.
   - (16) Logistics-related requirements.
   - (17) Packaging requirements.
   - (18) Precedence and criticality of requirements.

c) Qualification provisions (e.g., demonstration, test, analysis, inspection).

d) Bidirectional requirements traceability.

e) Requirements partitioning for phased delivery.

f) Testing requirements that drive software design decisions (e.g., special system level timing requirements/checkpoint restart).

g) Supporting requirements rationale.

Guidelines for the Software Data Dictionary Content

Software Data Dictionary shall include: [SWE-110]

a) Channelization data (e.g., bus mapping, vehicle wiring mapping, hardware channelization).

b) Input/Output (I/O) variables.

c) Rate group data.

d) Raw and calibrated sensor data.

e) Telemetry format/layout and data.

f) Data recorder format/layout and data.

g) Command definition (e.g., onboard, ground, test specific).

h) Effecter command information.

i) Operational limits (e.g., maximum/minimum values, launch commit criteria information).

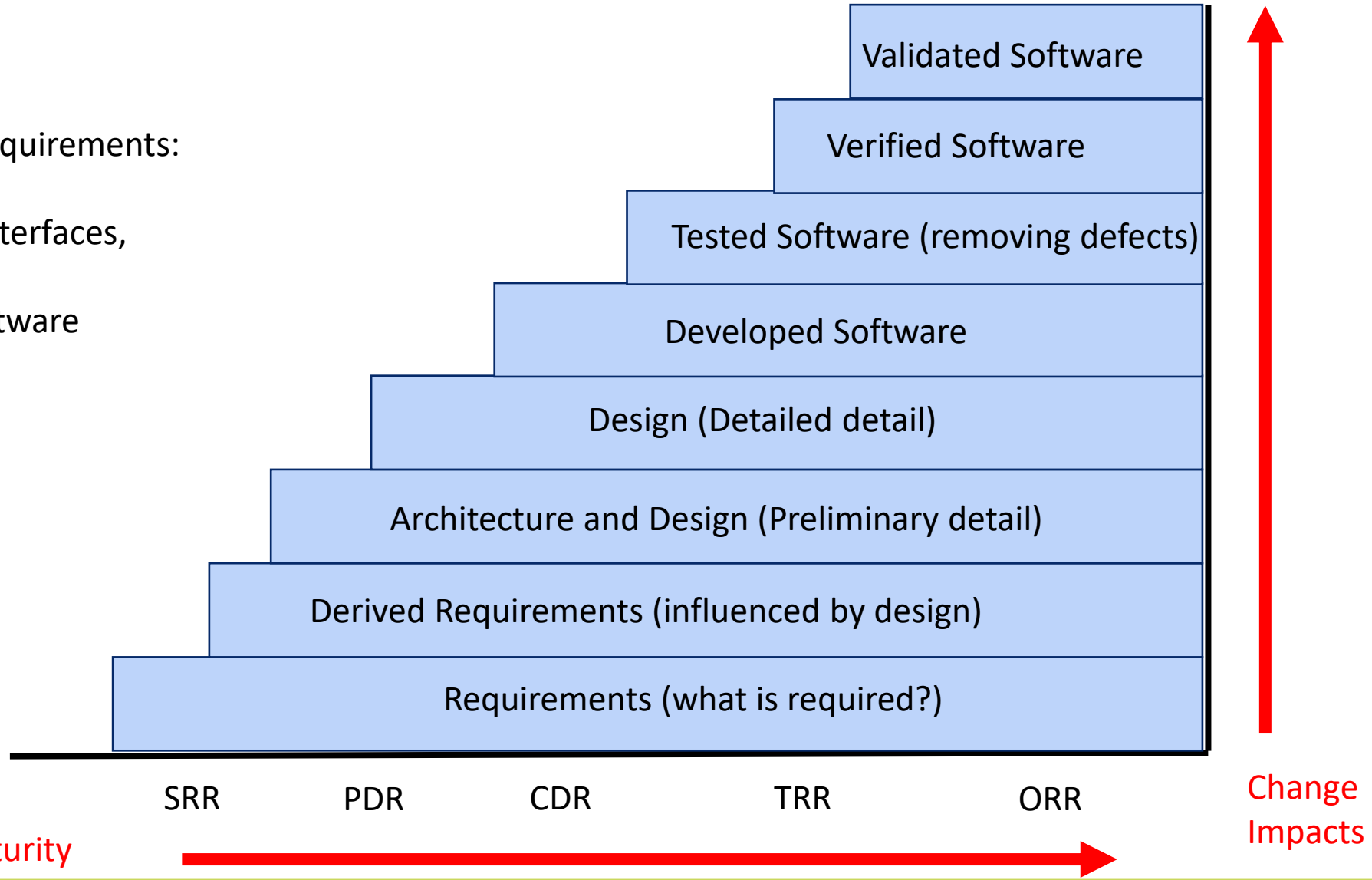
Example from Integrated Measurement And Command System

HardwareID	Radius
HardwareEngineeringName	ClockAngle
HardwareOpName	InternalExternal
HardwareDescription	HardwareComments
SignalType	HardwarePOC
HardwareType	HardwareControllingDocument
HardwareCategory	HardwareChangeAuthorization
InstrumentationType	SignalRouting
RefDes	Card
LowStateDefinition	Channel
HighStateDefinition	ExcitationConnector
PositiveAccuracy	ExcitationPinPositive
NegativeAccuracy	ExcitationPinNegative
AccuracyUnits	SignalConnector
Precision	SignalPinPositive
SampleRate	SignalPinNegative
LaunchCommitCriteria	HardwareConnectivityComments
FlightCritical	HardwareConnectivityPOC
Criticality	HardwareConnectivityControllingDocument
CriticalityRationale	HardwareConnectivityChangeAuthorization
AbortDetermination	PrimitiveCUI
CautionWarningDetection	HardwarePrimitiveIndexComments
CoordinateX	HardwarePrimitiveIndexPOC
CoordinateY	HardwarePrimitiveIndexControllingDocument
CoordinateZ	HardwarePrimitiveIndexChangeAuthorization
ApproxXStation	

Requirements Maturity

Examples of maturing requirements:

- Fault Management,
- Detailed Hardware Interfaces,
- Command Details,
- Hardware fixes in software



When Requirements Development Is Not Done Well…

- **Unstated requirements or poorly stated requirements lead to confusion among staff and customers.**
- **Design, implementation, and test work products inconsistently interpret the requirements.**
- **It takes an inordinately long time to get agreement on product design.**
- **There is an increased potential for higher costs to meet customer expectations.**

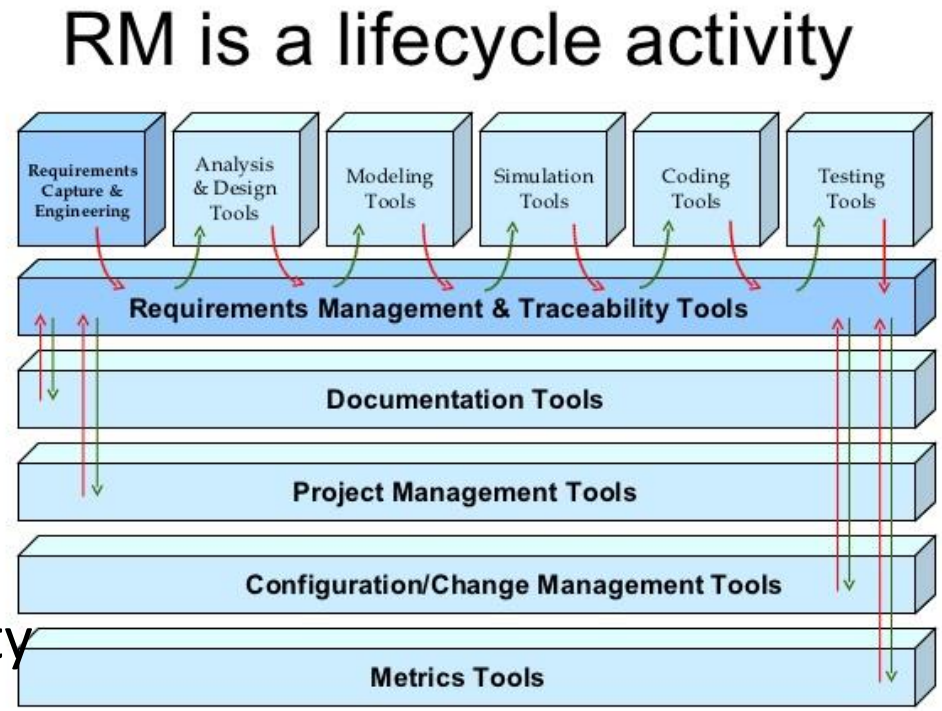
BA Fundamentals: Writing Good Requirements

- Clearly stated requirements are listed as one of the top reasons a project is successful.
- 50-70 % of projects fail due to poorly stated requirements
- Good Requirements are one of the key determinants of project "success".

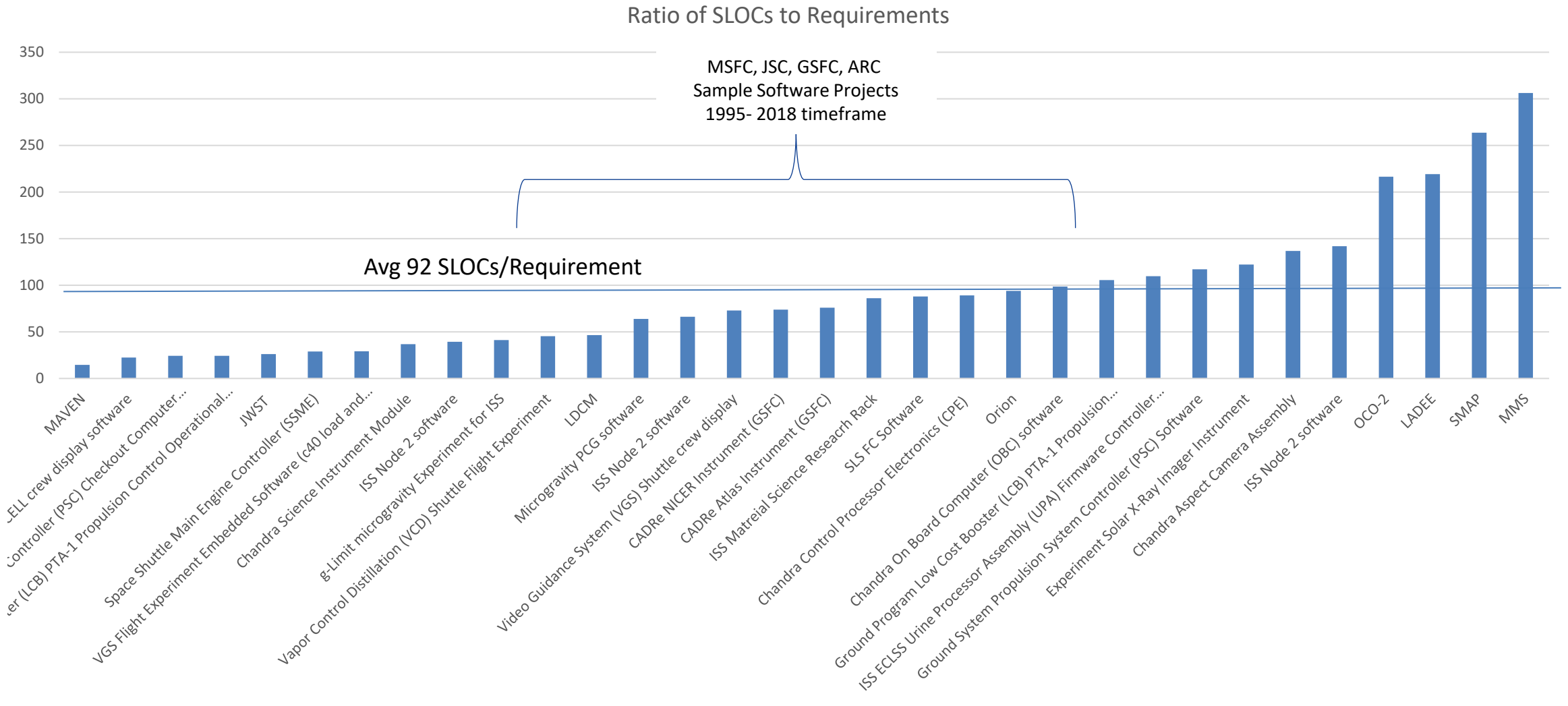
BA = Business Analysis
UBC Sauder, School of Business
The University of British Columbia

Requirements Management Metrics

- New/Added Requirements
- Modified Requirements
- Deleted Requirements
- Requirements Traceability Percentage
- Number of derived requirements
- Requirements Volatility
- HW and SW Interface requirement maturity
- Updated cost estimates



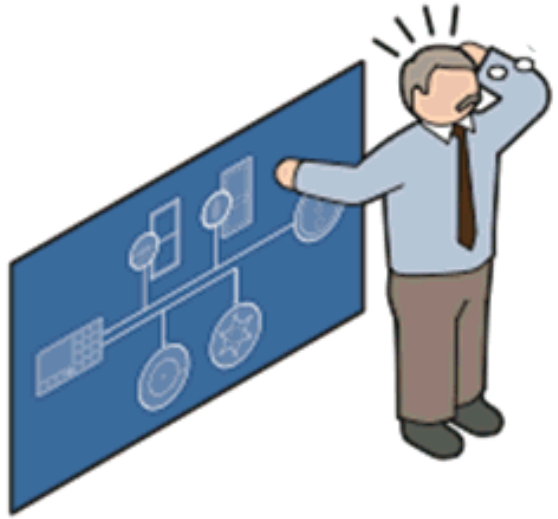
Comparison Chart for SLOC / Requirements Ratios



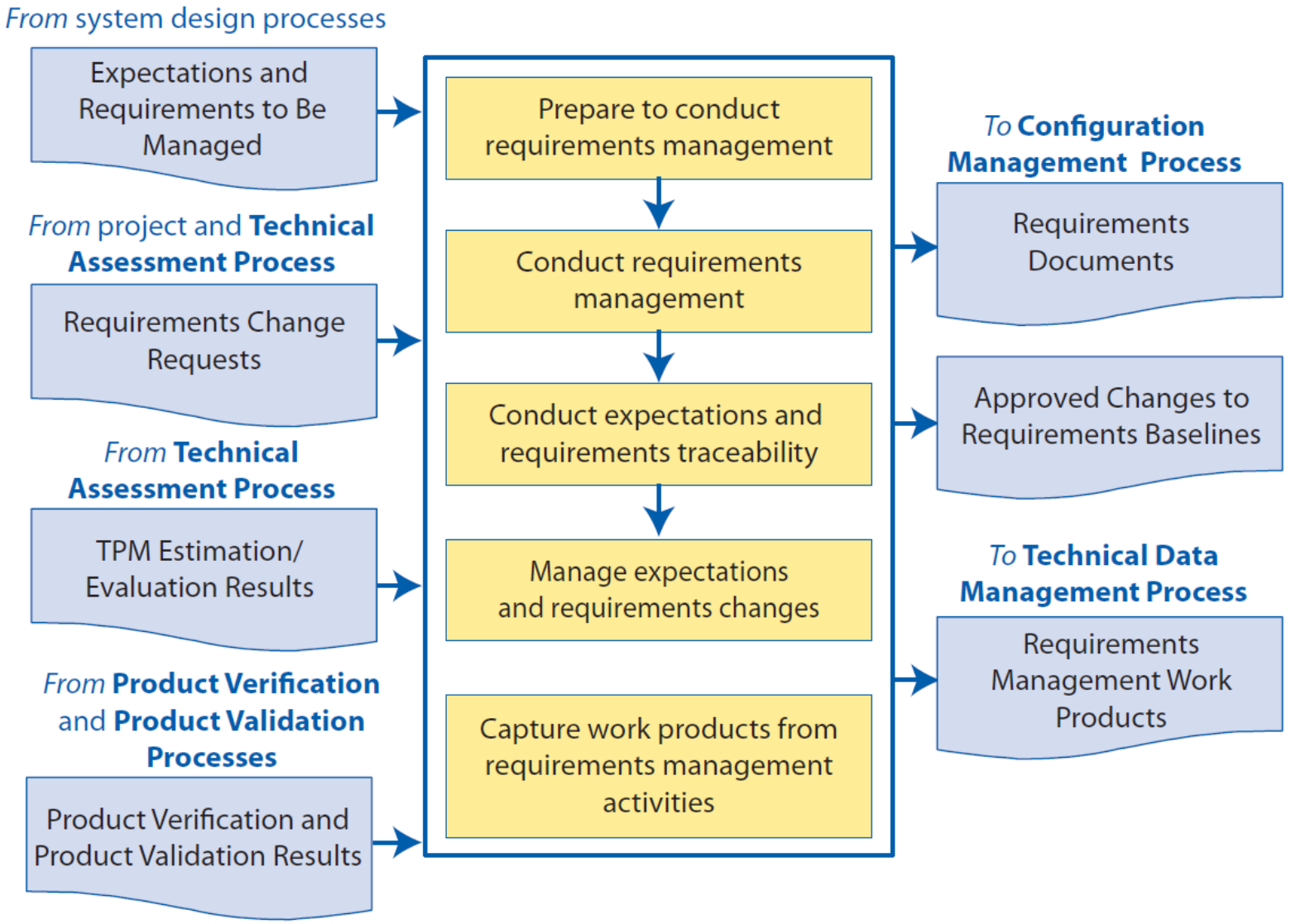
Requirements Management

Purpose

Manage requirements of the project's products and product components and to ensure alignment between those requirements and the project's plans and work products.



Requirements Management Process



When Requirements Management Is Not Done Well…

- Requirements are accepted by staff from any source they deem to be authoritative.
- The project experiences a high level of requirements changes.
- There are high levels of rework throughout the project.
- There is an inability to prove that the product meets the approved requirements.
- Lack of requirements traceability often results in incomplete or incorrect testing of the product.

Common Software Requirements Problems

Defining and documenting requirements is not a simple task, common problems that occur during or because of this activity and which are to be avoided, include:

- **Not enough detail in the software requirements**
- **Fault management requirements for hardware and software**
- **Failing to define needed requirements, including safety requirements.**
- **Writing requirements ambiguously.**
- **Using inexperienced personnel to define the requirements.**
- **Incorrect understanding of underlying assumptions or constraints.**
- **Including unneeded features or capabilities.**
- **No clear method for allocating requirements to software subsystems.**
- **Failing to spend enough time or resources on requirements definition.**
- **Pointing to other sources for the requirement information**

How Would You Design and Code These Software Requirements?

- The XXXX software shall neither generate inaccurate data nor inaccurately display data which could potentially cause Range Safety to incorrectly conclude that a safe for launch or safe flight condition exists.
- All GN&C functions shall implement deterministic behavior in the presence of detectable numerical errors.

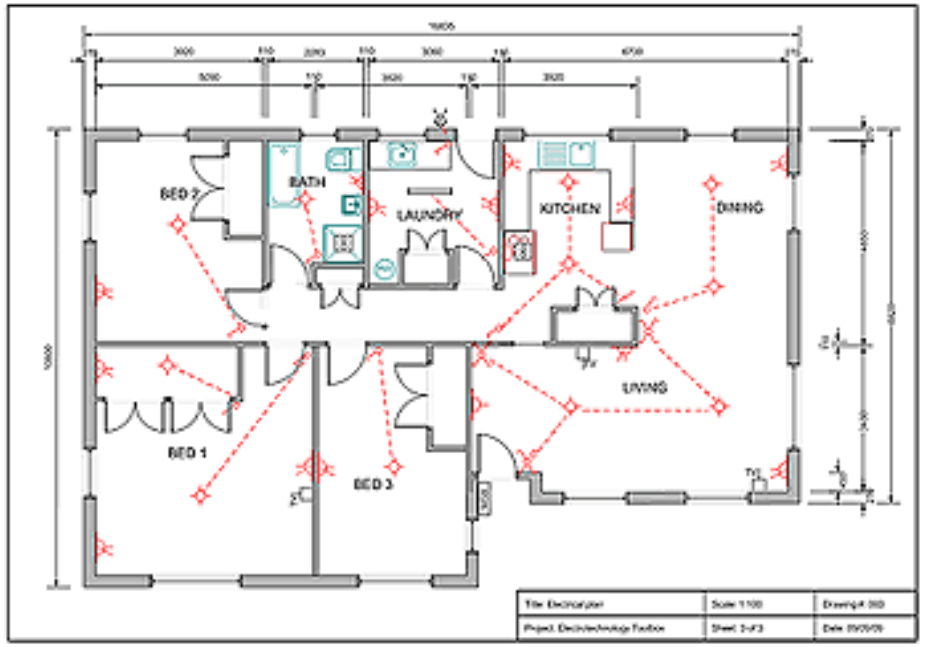
Software Architecture

NPR 7150.2D Requirements on Software Architectures

- 4.2.3 The project manager shall **transform the requirements for the software into a recorded software architecture.** [SWE-057]
- 4.2.4 The project manager shall **perform a software architecture review** on the following categories of projects: [SWE-143]
  - a. Category 1 Projects as defined in NPR 7120.5.
  - b. Category 2 Projects as defined in NPR 7120.5 that have Class A or Class B payload risk classification per NPR 8705.4.

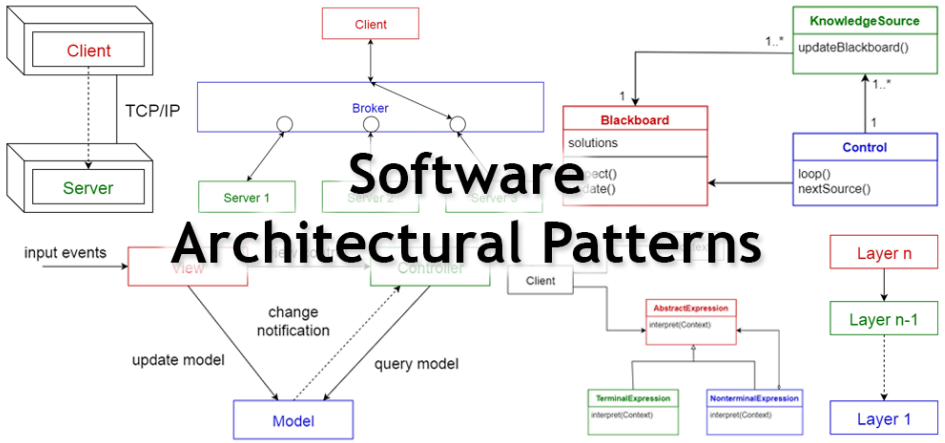
Questions

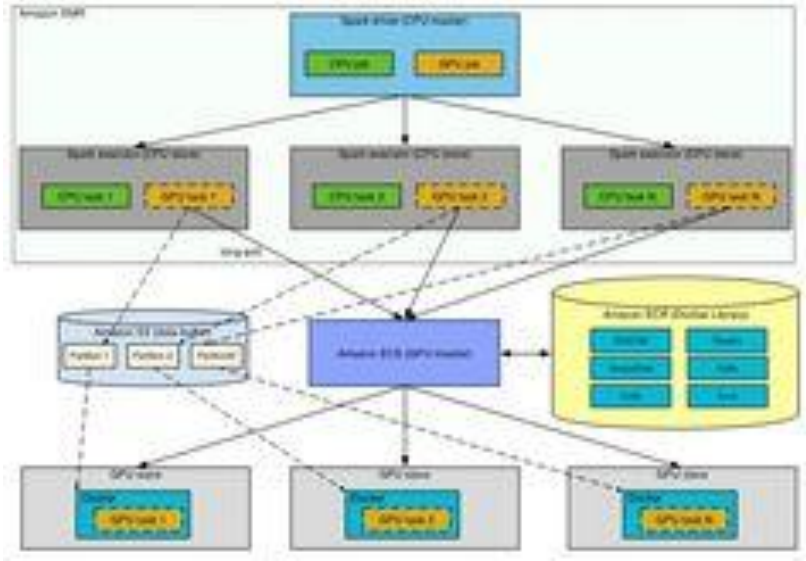
- Would you build a house without an architecture plan?
- What are some of the architectural features that you would want in your house?



What is Architecture?

- Architecture is an essential software engineering responsibility,
- Architecture addresses the structure, not only of the software, but also of its functions, the environment within which it will work, and the process by which it will be built and operated
- Just as importantly, however, architecture also deals with the principles guiding the design and evolution of a software program
  - Complexity, uncertainty, and ambiguity in the design of complicated systems may be reduced to workable concepts
  - In the best practice of architecture, this aspect of architecture must not be understated or neglected





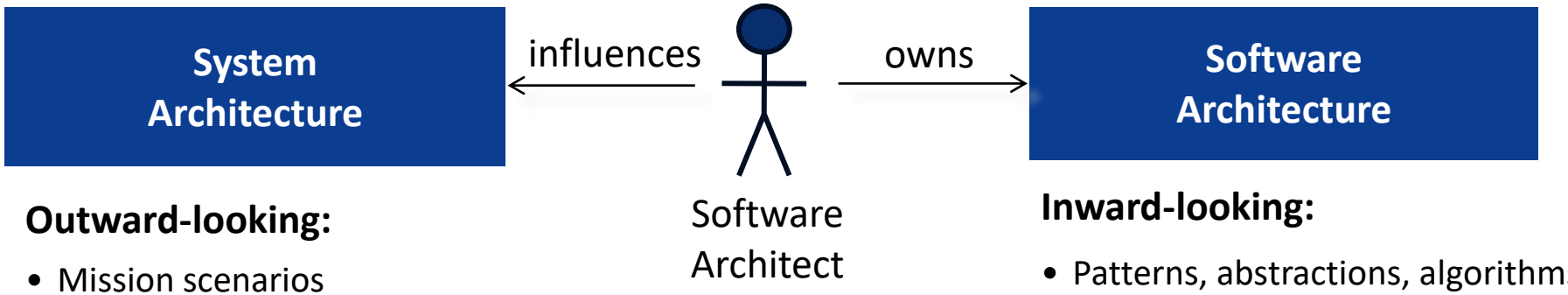
Two Aspects of "Architecture"

- **a**rchitecture — What gets built
  - Describes components and interfaces
  - Specifies details of assembly and integration
- **A**rchitecture — Why it gets built the way it does
  - Identifies properties of interest beyond just the requirements, and from all essential points of view
  - Defines workable abstractions and other patterns of design that give the design its shape and reflect fundamental concepts of the domain
  - Guides design and maintains principles throughout the development lifecycle
  - Builds on a body of experience and refines concepts as necessary

Architecture is about managing complexity

Source: Bob Rasmussen, JPL

System Architecture vs. Software Architecture



Outward-looking:

- Mission scenarios
- Functional decomposition
- System analysis
- Performance requirements
- Resource allocations
- Command and telemetry dictionaries
- Flight rules and constraints
- Control laws
- Failure modes analysis
- Fault protection
- Test procedures
- …

Inward-looking:

- Patterns, abstractions, algorithms
- Monitoring and control
- Data representation and data management
- Concurrent threads, processes, memory management
- Real-time execution, throughput
- Synchronization
- Inter-process communication
- Languages, libraries, operating systems
- Verification and validation
- …

Software Architect Essential Activities

- Understand what a system must do
- Define a system concept that will accomplish this
- Render that concept in a form that allows the work to be shared
- Communicate the resulting architecture to others
- Ensure throughout development, implementation, and testing that the design follows the concepts and comes together as envisioned
- Refine ideas and carry them forward to the next generation of systems

Source: Bob Rasmussen, JPL

Software Architecture Documentation

The actual format for recording and describing the architectural concept is left to the software project team. As a minimum, include the following:

- An assessment of architectural alternatives.
- A description of the chosen architecture.
- Adequate description of the subsystem decomposition.
- Definition of the dependencies between the decomposed subsystems.
- Methods to measure and verify architectural conformance.
- Characterization of risks inherent to the chosen architecture.
- Documented rationale for architectural changes(if made).
- Evaluation and impact of proposed changes.

Summary for Software Architectures

- Architecture is not just high-level design
  - It includes quality attributes, rationale, and principles
- Architecture is not a one-time effort
  - Make software architecture a driving force throughout the lifecycle
  - Good architectures don't step aside once development starts
- Embrace well-architected software as a response to system complexity
  - Weak architecture …
    - Can't be analyzed or validated for correct behavior, except case by case
    - Can't be changed with confidence, even to correct errors
    - Can't be operated with confidence, other than the way it was tested
    - Can't be reused easily or inherited from
- Conduct software architecture reviews to …
  - Inspect quality attributes, principles of design, verifiability, and operability
  - Give team members a clearer understanding of the project

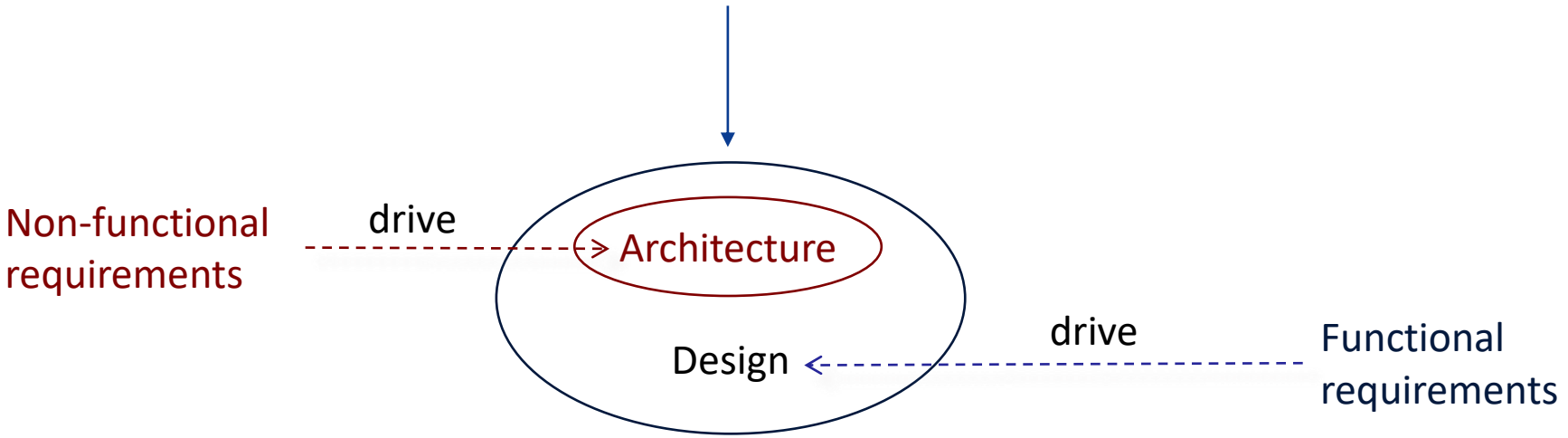
Software Design

NPR 7150.2D Requirements on Software Design

- 4.3.2 The project manager shall **develop, record, and maintain a software design** based on the software architectural design that describes the lower-level units so that they can be coded, compiled, and tested. [SWE-058]

Architecture versus Design

All architecture is design, but not all design is architectural



Architects intentionally limit their focus and avoid the details of how elements do what they do. Detailed designs and implementation details are left to downstream engineers/experts.

Downstream engineers are expected to respect the architecture to ensure properties promised by the architect are present in the product.

What is the Design?

- Software design - activities that fit between requirements and implementation or coding
- Starts with the architectural design and describes the lower-level components and interfaces so they can be coded

Programmer needs
A good understanding

To build
an elephant!

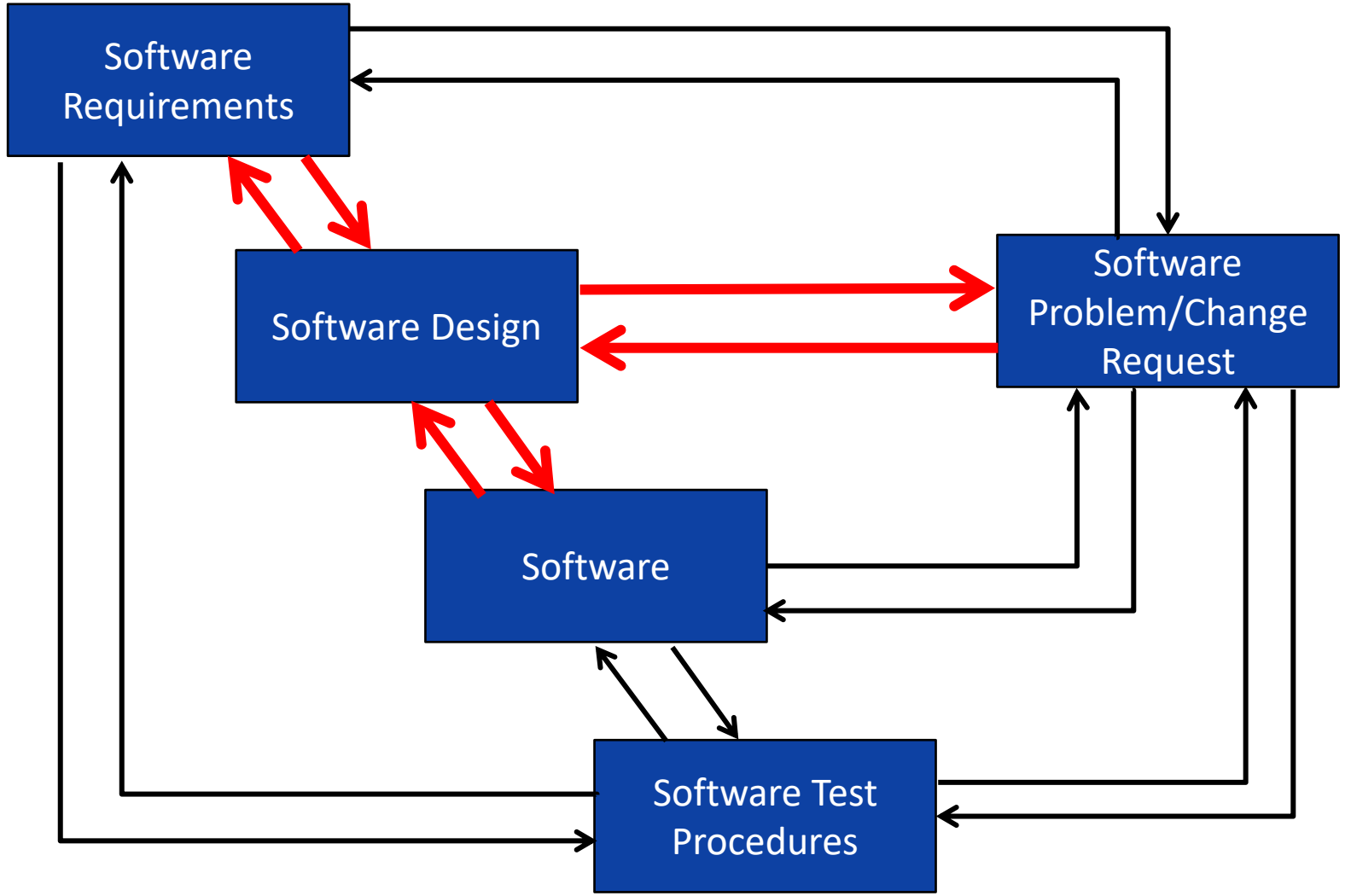
Transforms the "What" to "How"

Activities During Design

- Typically design is divided into 2 stages:
  - Preliminary Design: External design describes the real-world design; Architectural design deposes the requirements into software subsystems and defines high level interfaces
  - Detailed Design: Further descriptions of the subsystems; Decomposition of subsystems into components; Describes the internal interfaces
- Formal reviews PDR, CDR are held after each step. Design is baselined at CDR.

Bi-directional Traceability

Bidirectional traceability is defined as a traceability chain that can be traced in both the forward and backward directions



The project shall perform and maintain bidirectional traceability between the software requirements and the software design.

Software Design Considerations (1 of 2)

Many things need to be considered during design (for example: “ilities”):

- Compatibility: how will it work with other software?
- Extensibility: Can it be changed easily for new capabilities?
- Fault-Tolerance: Can software recover from failures?
- Maintainability: How easily can functional modifications or bug fixes be made?
- Modularity: Are components easy to implement or test in isolation?
- Reliability: Can software perform its required functions over a specified period of time?

Software Design Considerations (2 of 2)

- Reusability: Can software be used in multiple applications with little or no modification?
- Robustness: Can software operate under stress or tolerate unpredictable/invalid input?
- Security: Is it able to withstand hostile acts?
- Usability: Is the software convenient to use?
- Performance: Does the software perform within specified time limits?
- Scalability: Does the software adapt to increases in data or users?
- Safety: Have the safety aspects of the system been considered?

A Design Strategy

- Determine which design decisions are the most difficult to make or most likely to change
- Use information hiding to design each hard decision into a component specification
  - Make the decisions affecting the largest portion of the system first
  - Place the decisions “most likely to change” in modules first
  - Then place other hard decisions and decisions likely to change into modules
- Continue process until all design decisions are hidden in a component and provide low-level implementation assignments

Rules of Software Design

- Make sure design is clearly stated (avoids misinterpretation!)
  - All design criteria, requirements, and constraints should be listed in design
- Document design decisions
- Check design for consistency (Avoids issues with separately developed modules that don't fit together)
- Always design for extension and contraction (Changes are inevitable!)
- Do not connect independent concerns
- Design external functionality before internal functionality
  - View solution as a black box and decide how it will interact with its environment—Then design the inner organization of the "box"
- Choose reused software carefully
  - Exercise caution if reusing only part of a reusable component;
  - Check that it meets requirements;
- Keep design as simple as possible
  - Minimize dependencies –Design components so they know about as few other components as possible
  - Use as few parameters as possible
  - Minimize number of calls between components
- Prototype when applicable
- When possible, use proven patterns to solve design problems
- For flight software, consult Software Design Principles
- When crossing between paradigms, build an interface layer that separates the two

Take Advantage of the Software Engineering Design Principles in Developing Your Software Designs

- **Design principles in the following topic areas:**
  - **Resource Margins**
  - **Dead Code Exclusion**
  - **Initialization/Safe Mode**
  - **Input Data Errors**
  - **IO Failures**
  - **Resource Oversubscription**
  - **Incorrect Memory Use/Access**
  - **Thread Safety**
  - **Resource Usage Measurement**
  - **Invalid Data Handling**
  - **FSW Modification**
  - **Data Interface Integrity**
  - **Command Receipt Acknowledgement**
  - **Toggle Commands**
  - **Coding Standards**
  - **Fault Protection**

https://swehb.nasa.gov/ -> D. Topics-> (Tab) Software Design Principles

Toggle Commands Example

1. Principle

Design both internal and external commanding to place the system into an explicitly specified state.

2. Rationale

Making assumptions about the system state can lead to malfunctions

- **Discussion of cross-cutting issues of software safety (NPR 1750.2 SWE-134) and how the design principles support implementation of the NPR**

Software Design Metrics

- Number of components designed
- Traceability percentage between the software design and software requirements
- Number of units designed
- Number of CSCI designed
- Estimated SLOC count
- Updated software cost estimate
- UML metrics

Summary For Software Design

- A good design follows a few key principles:
  - Separate the interface from the implementation
  - Determine what is common and what is variable with an interface and an implementation
  - Allow substitution of variable implementations through a common interface
  - Determining what should be common vs. variable should depend on the goals, nothing extra
- There are many modeling languages, both graphical and textual, (UML) that can help describe your design and its behavior
- BUT—Good design still requires a thorough understanding of the requirements and a lot of careful thought and planning!

Software Implementation or Coding

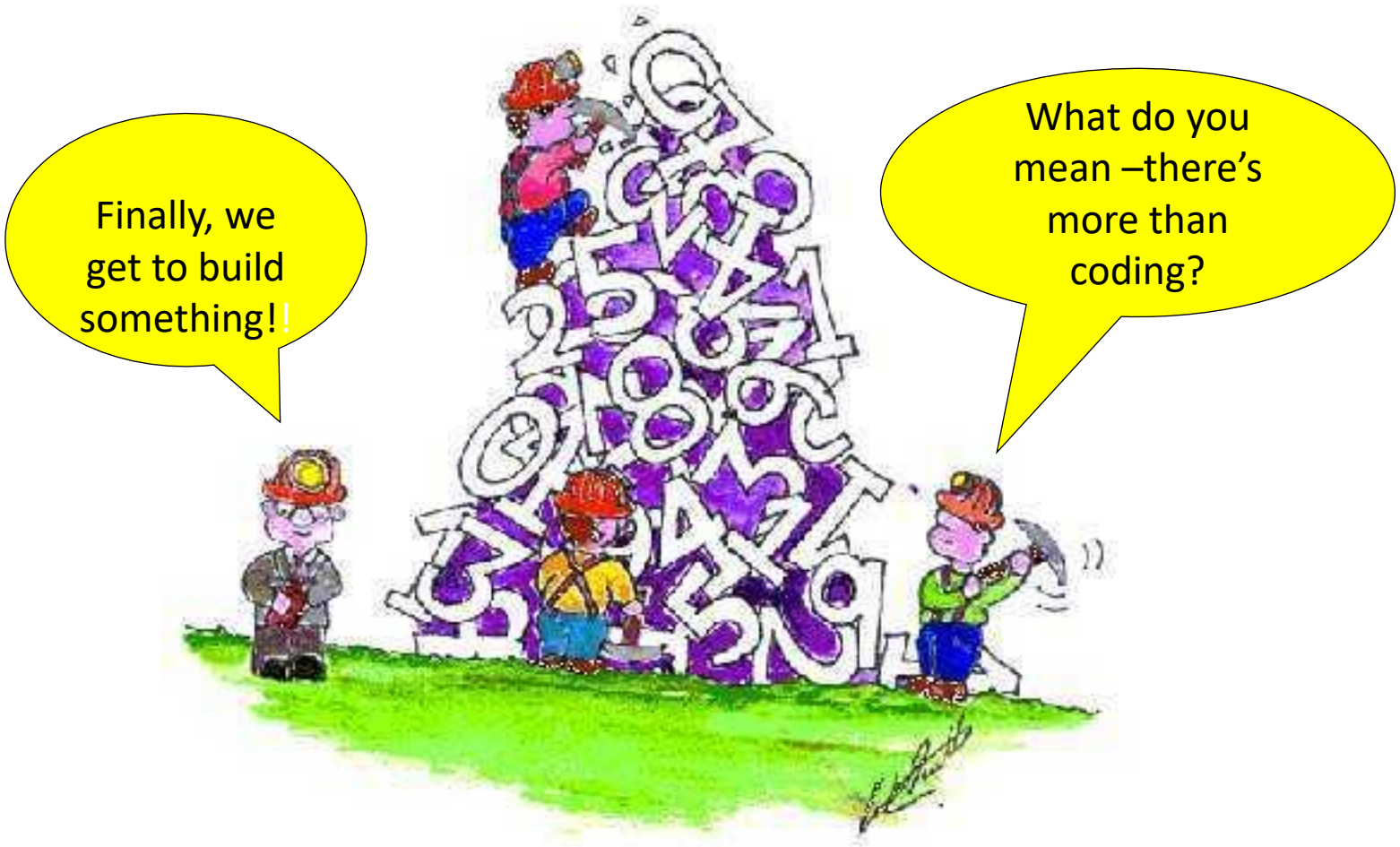
NPR 7150.2D Requirements During Implementation

- 4.4.2 The project manager shall implement the software design into software code. [SWE-060]
- 4.4.3 The project manager shall select, define, and adhere to software coding methods, standards, and criteria. [SWE-061]
- 4.4.4 The project manager shall use static analysis tools to analyze the code during the development and testing phases to, at a minimum, detect defects, software security, code coverage, and software complexity. [SWE-135]
- 4.4.5 The project manager shall unit test the software code. [SWE-062]
- 4.4.6 The project manager shall assure that the unit test results are repeatable. [SWE-186]

NPR 7150.2D Requirements During Implementation

- 4.4.7 The project manager shall provide a software version description for each software release. [SWE-063]
- 4.4.8 The project manager shall validate and accredit the software tool(s) required to develop or maintain software. [SWE-136]
- 3.11.8 The project manager shall identify, record, and implement secure coding practices. [SWE-207]
- 3.11.9 The project manager shall verify that the software code meets the project's secure coding standard by using the results from static analysis tool(s). [SWE-185]

Implementation



Software Implementation –More Than Coding!

- Software implementation consists of implementing the requirements and design into code, data, and documentation
- Software implementation also consists of following coding methods and standards
- Unit testing is also a part of software implementation.
- Other implementation activities:
  - Peer-reviews, code walkthroughs
  - Use of static analyzers
  - Building test drivers and simulators
  - Development of build procedures
  - Documentation, may include unit development folders, build test plans and results, software version description, users guide, operations manual, maintenance manual
  - Following coding standards
  - Maintaining software configuration control
  - Reporting metrics
  - Generating / Maintaining traceability information
  - Responding to changes!
  - Other possibilities: prototyping, user training, build testing

Top 15+ Best Practices for Writing Super Readable Code

- Commenting & Documentation
- Consistent Indentation
  - Keep your indentation style consistent.
- Avoid Obvious Comments
- Code Grouping
  - More often than not, certain tasks require a few lines of code. It is a good idea to keep these tasks within separate blocks of code, with some spaces between them.
- Consistent Naming Scheme
- DRY Principle
  - DRY stands for Don't Repeat Yourself. Also known as DIE: Duplication is Evil.
  - The principle states:
    - "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."
  - The same piece of code should not be repeated over and over again.
- Avoid Deep Nesting
  - Too many levels of nesting can make code harder to read and follow.

Top 15+ Best Practices for Writing Super Readable Code by Burak Guzel

Top 15+ Best Practices for Writing Super Readable Code

- **Limit Line Length**
  - **Our eyes are more comfortable when reading tall and narrow columns of text. This is precisely the reason why newspaper articles look like they do.**
- **File and Folder Organization**
  - **Technically, you could write an entire application code within a single file. But that would prove to be a nightmare to read and maintain.**
- **Consistent Temporary Names**
- **Capitalize SQL Special Words**
  - **Database interaction is a big part of most web applications. If you are writing raw SQL queries, it is a good idea to keep them readable as well.**
- **Separation of Code and Data**
  - **This is another principle that applies to almost all programming languages in all environments**

Top 15+ Best Practices for Writing Super Readable Code by Burak Guzel

Top 15+ Best Practices for Writing Super Readable Code

- Object Oriented vs. Procedural
  - Object oriented programming can help you create well structured code. But that does not mean you need to abandon procedural programming completely.
- Read Open Source Code
  - Open Source projects are built with the input of many developers. These projects need to maintain a high level of code readability so that the team can work together as efficiently as possible. Therefore, it is a good idea to browse through the source code of these projects to observe what these developers are doing.
- Code Refactoring
  - When you "refactor," you make changes to the code without changing any of its functionality. You can think of it like a "clean up," for the sake of improving readability and quality.

Top 15+ Best Practices for Writing Super Readable Code by Burak Guzel

Software Builds/Releases

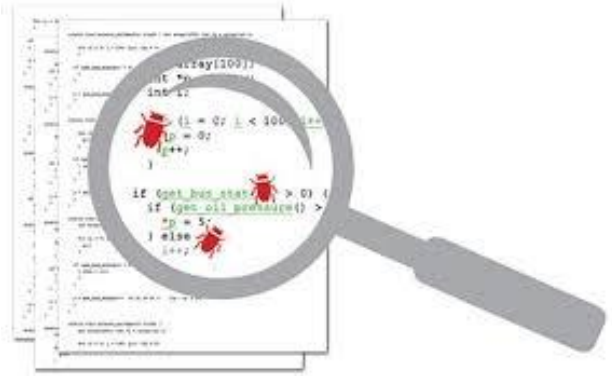
- A software build is: A portion of the system that satisfies an identified subset of the total software requirements
- A software release is: a build that is delivered to a customer for formal testing
- Why do we need builds?
  - Enables early testing of the software system
  - Allows early delivery of capabilities needed for testing other items (like hardware)
  - Enables feedback on usability features
  - Allows us to workaround uncertain requirements, long lead items
  - Enables better progress tracking and schedule estimation

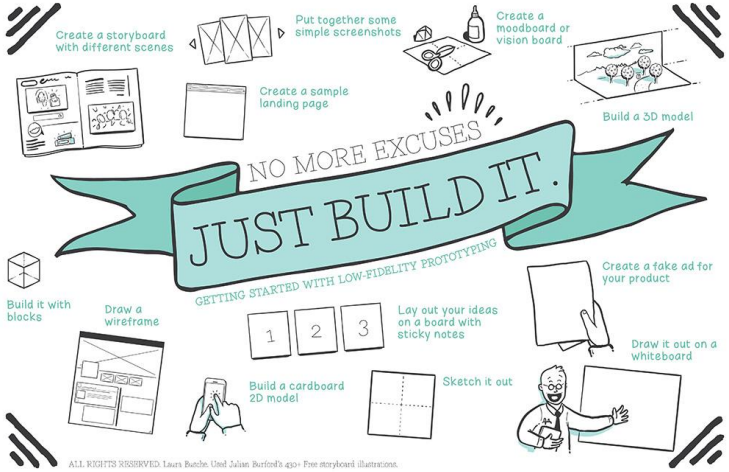
Software Build Guidelines

- Keep the first build simple-especially if new application, computer, etc.
- Each build should contain complete testable functions of the system and add to the capabilities of the previous build
- Work around long lead times (Hardware deliveries, operational computers, etc.)
- Plan capabilities critical to operational use of software early
- Don’t postpone “hard stuff” (high risk requirements, complex capabilities)
- Delay capabilities where requirements are incomplete or unstable until later builds
- Plan requirements critical for usability, stability, performance for net to last build
- Plan for a “clean-up” build
- Don’t plan a build with a long duration (longer than 8-9 months)

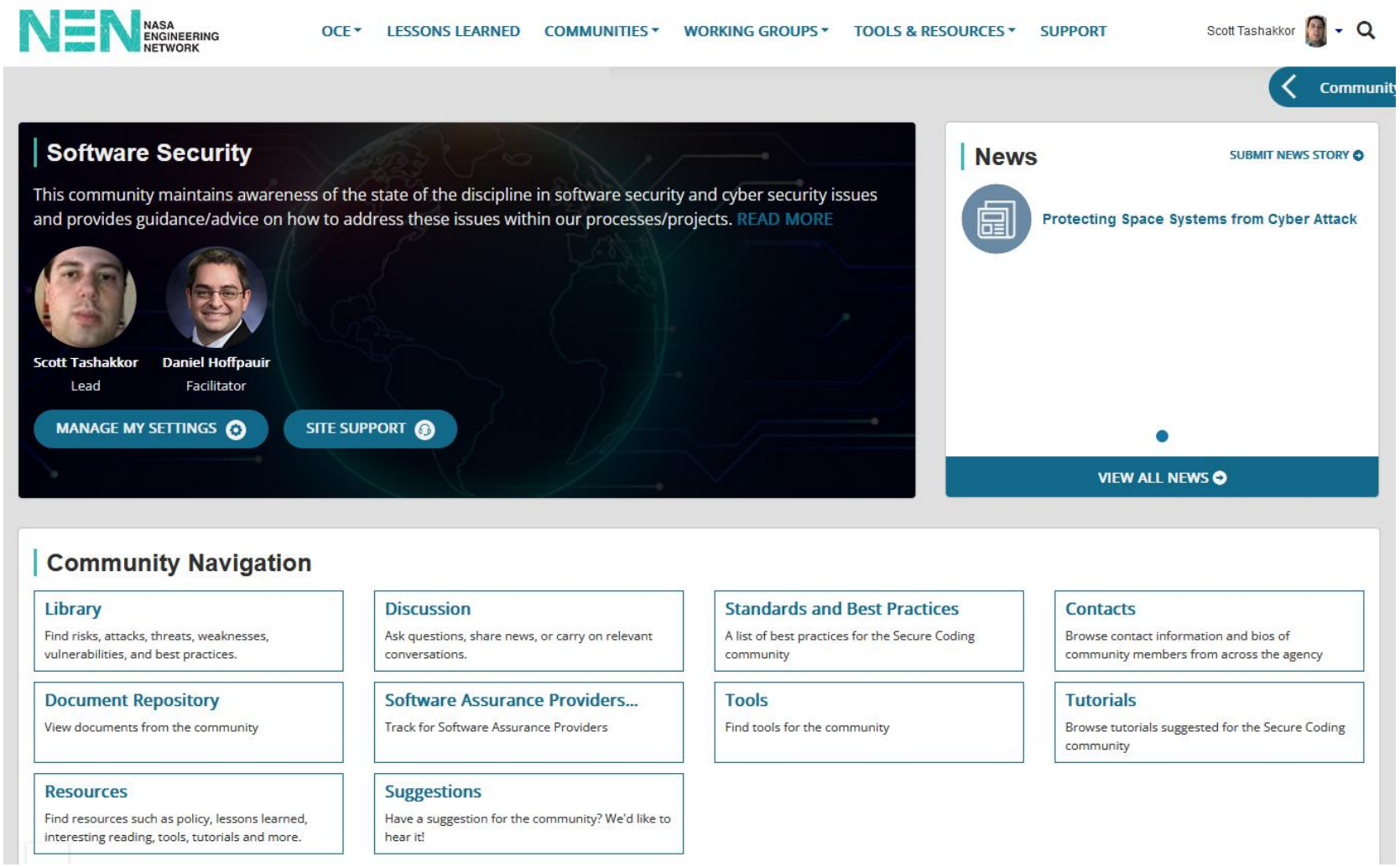
Other Implementation Topics

- Static analyzer tools –allow the analysis of the software without actually running it
  - Different analyzers focus on different types of errors: violations of coding standards, input/output flaws, security vulnerabilities, coding errors such as memory leaks, unreachable code, etc.
- Prototyping may be necessary for some parts of the systems, e.g., to verify that performance req. can be met, or test interface requirements
- Peer reviews/walkthroughs –Should be done on safety critical code, code performing critical functions, complex functions
- Considerations for COTS:
  - Verify that COTS meets your requirements
  - Make sure you are using COTS as intended
  - Make arrangements for maintenance of COTS components
- Many other activities performed during implementation are covered in other areas of the class





Secure Coding Community of Practice Site



https://nen.nasa.gov/web/coding

And Then There's Documentation!

Software Version Description (VDD)

- Identification of system
- Executable software
- Software life cycle data
- Archive and release data
- Instructions for building software
- Data integrity checks
- Open problem reports, including workarounds
- Change requests implemented in current software version since last VDD

User's Manual

- Software summary
- Access to software (initiating a session, running software, etc.)
- Processing reference guide (capabilities, back-up, recovery, messages, etc.)
- Assumptions, limitations, safety concerns
- Information that is unique for version of the software

Measures in Implementation

- Implementation progress:
  - planned vs. actual schedule
  - # modules coded/unit tested vs. # modules planned
  - SLOC Developed vs Planned
- Functionality:
  - # modules delivered in build/release vs. planned #
- Volatility:
  - # of requirements changes vs. time
- Quality:
  - # of errors found in peer reviews vs. expected #
  - # of peer reviews planned vs. # completed
  - Coding standard errors found per module
- Management:
  - staffing vs. planned staffing

Even with all this------

Most people think implementation is “the fun part!”

Software Testing

Software Testing Requirements NPR 7150.2D

- 4.5.2 The project manager shall establish and maintain: [SWE-065]
  - ***a) Software test plan(s).***
  - ***b) Software test procedure(s).***
  - ***c) Software test(s), including any code specifically written to perform test procedures.***
  - ***d) Software test report(s).***
- 4.5.3 The project manager shall ***test the software against its requirements.*** [SWE-066]
  - Note: A best practice for Class A, B, and C software projects is to have formal software testing planned, conducted, witnessed, and approved by an independent organization outside of the development team.
- 4.5.4 The project manager shall ***place software items under configuration management prior to testing.*** [SWE-187]
  - Note: This includes the software components being tested and the software components being used to test the software, including components like support software, models, simulations, ground support software, COTS and MOTS.

Software Testing Requirements NPR 7150.2D

- 4.5.5 The project manager shall ***evaluate test results and record the evaluation.*** [SWE-068]
- 4.5.6 The project manager shall ***use validated and accredited software models, simulations, and analysis tools*** required to perform qualification of flight software or flight equipment. [SWE-070]
  - Note: Information regarding specific verification, validation and credibility techniques and the analysis of models and simulations can be found in NASA-STD-7009 and NASA-HDBK-7009.
- 4.5.7 The project manager shall ***update software test and verification plan(s) and procedure(s) to be consistent with software requirements.*** [SWE-071]
- 4.5.8 The project manager shall ***validate the software system on the targeted platform or high-fidelity simulation.*** [SWE-073]
  - Note: Typically, a high-fidelity simulation has the exact processor, processor performance, timing, memory size, and interfaces as the target system.
- 4.5.9 The project manager shall ***ensure that the code coverage measurements for the software are selected, implemented, tracked, recorded and reported.*** [SWE-189]

Software Testing Requirements NPR 7150.2D

- 4.5.10 The project manager shall ***verify code coverage is measured*** by analysis of the results of the execution of tests. [SWE-190]
  - Note: If it can be justified that the required percentage cannot be achieved by test execution, the analysis, inspection or review of design can be applied to the non-covered code. The goal of the complementary analysis is to assess that the non-covered code behavior is as expected.
- 4.5.11 The project manager shall plan and ***conduct software regression testing*** to demonstrate that defects have not been introduced into previously integrated or tested software and have not produced a security vulnerability. [SWE-191]
- 4.5.12 The project manager shall ***verify through test the software requirements that trace to a hazardous event, cause or mitigation technique.*** [SWE-192]
- 4.5.13 The project manager shall ***develop acceptance tests for loaded or uplinked data, rules, and code that affects software and software system behavior.*** [SWE-193]
  - Note: These acceptance tests should validate and verify the data, rules, and code for nominal and off-nominal scenarios.

Software Testing Requirements NPR 7150.2D

- 4.5.14 The project manager shall ***test embedded COTS, GOTS, MOTS, OSS, or reused software components to the same level required to accept a custom developed software component for its intended use.*** [SWE-211]

What is a Testing?

Testing

The ***execution*** of an Object Under Test (OUT) under specific ***preconditions*** with specific ***stimuli*** so that its ***actual behavior*** can be compared with its ***expected or required behavior***

- *Preconditions:* pretest mode, states, stored data, or external conditions
- *Stimuli:*
  - Calls, commands, and messages (control flows)
  - Data inputs (data flows)
  - Trigger events such as state changes and temporal events
- *Actual Behavior:*
  - *During Test:*
    - Calls, commands, and messages (control flows)
    - Data outputs (data flows)
  - *Postconditions*: post-test mode, states, stored data, or external conditions

Question

Who performs testing on your projects? Check all that apply.

- ☐ Project-internal Software Testers
- ☐ Independent Software Testers
- ☐ Independent Verification and Validation Testers
- ☐ Software Developers
- ☐ System Engineers
- ☐ Software Quality Assurance Engineers
- ☐ Others

Test Planning

- Plan before testing begins
  - Plan as soon as relevant stage complete
  - System test planning can start when requirements document is complete
  - Allows for acquisition/allocation of test resources
- Focus testing on components most likely to have issues (high risk, complex, many interfaces, demanding timing constraints, etc.)
- Involve the right people: quality engineers, software engineers, systems engineers, etc.
- Include coverage of user documentation
- Capture planning in a software test or software verification plan

Test Case Design / Test Procedures

- Include tests to:
  - Confirm software does what it is supposed to do
  - Confirm software does not do what it should not do
  - Confirm software behaves in an expected manner under adverse or off-nominal conditions
  - Cover range of allowable inputs, boundary conditions, false or invalid inputs, load tests, stress tests, interrupt execution and processing, etc.
  - Evaluate performance
- Do not guess at how the software works
  - If requirements not clear enough to write test procedures, ask questions of appropriate project team members
- Do not assume tester understands intricacies of the software design
  - Test procedures must be easy to follow

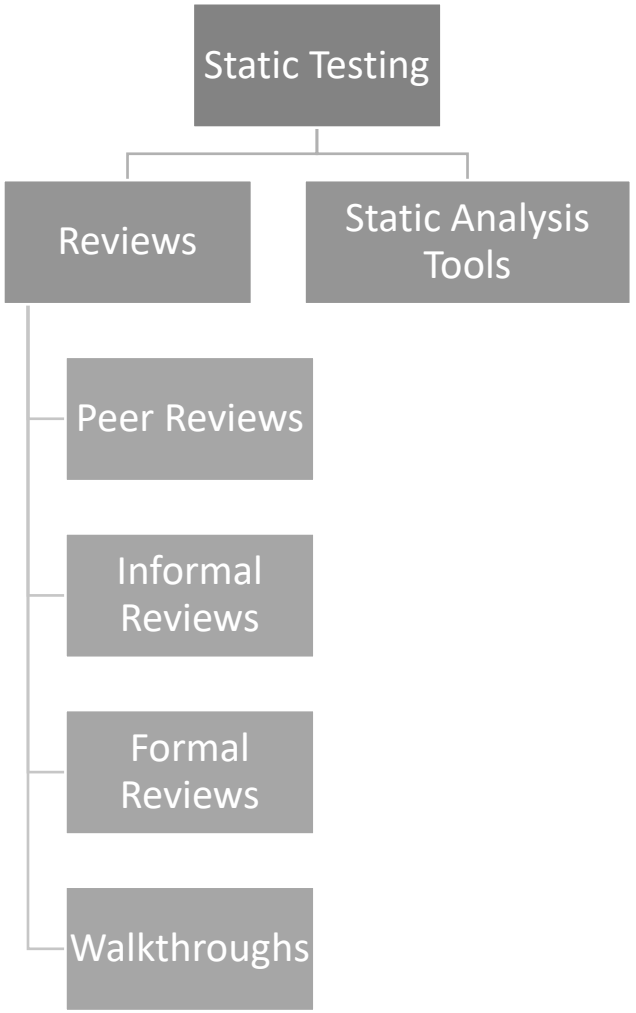
Software Test Procedure Guidelines

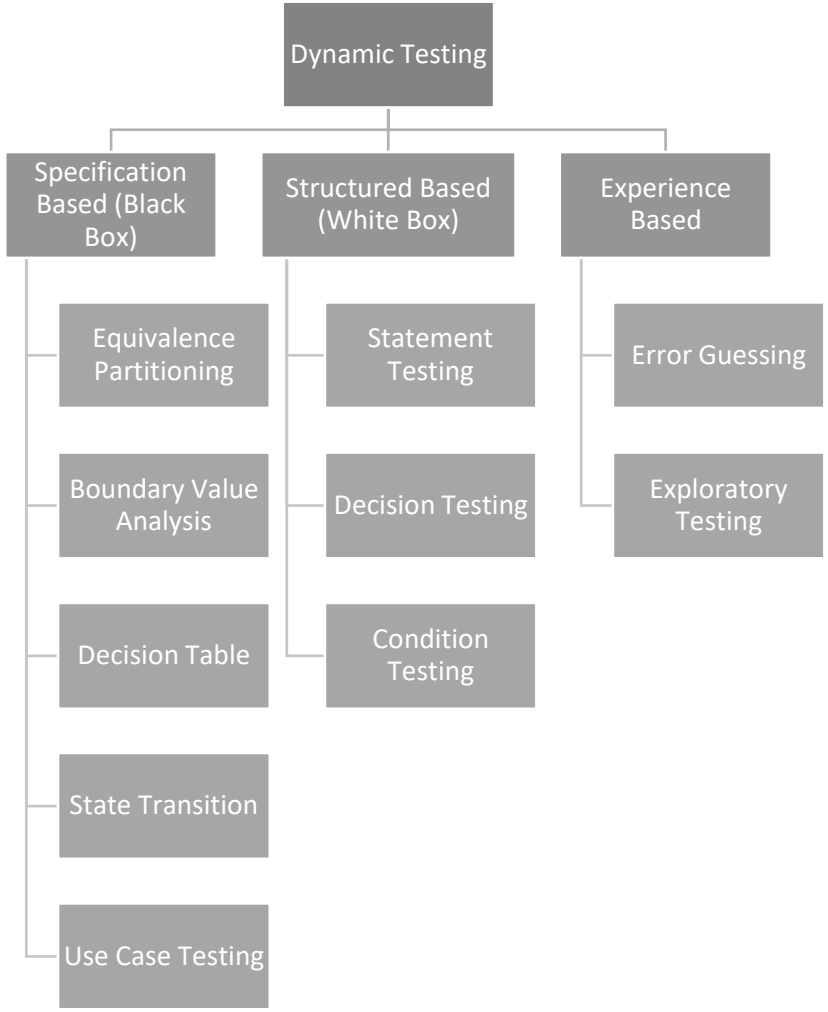
- The project should establish test cases, in terms of inputs, expected results, and evaluation criteria,
- Software test procedures, should cover the software requirements and design, including: as a minimum:
  - the correct execution of all interfaces (including between software units),
  - statements and branches;
  - all error and exception handling;
  - all software unit interfaces including limits and boundary conditions;
  - end-to-end functional capabilities,
  - performance testing,
  - operational input and output data rates and timing and accuracy requirements,
  - stress testing,
  - worst case scenario(s),
  - fault detection, isolation and recovery handling,
  - resource utilization,
  - hazard mitigations,
  - start-up, termination, and restart (when applicable); and all algorithms.

Software Test Procedure Guidelines

- Legacy reuse software should be tested for:
  - all modified reuse software,
  - for all reuse software units where the track record indicates potential problems and
  - all critical reuse software components even if the reuse software component has not been modified.
- All software testing should be in accordance with the defined test cases and procedures.
- Based on the results of the software testing, the developer should make all necessary revisions to the software, perform all necessary retesting, update the SDFs and other software products as needed.
- Regression testing should be performed after any modification to previously test software.
  - 4.5.11 The project manager shall plan and conduct software regression testing to demonstrate that defects have not been introduced into previously integrated or tested software. [SWE-191]

Comparison Of Types





Independence in Software Item Testing

- For Class A, B and Safety critical class C software:
- The person(s) responsible for software testing of a given software item should not be the persons who performed detailed design, implementation or unit testing of the software item.
- This does not preclude persons who performed detailed design, implementation or unit testing of the software item from contributing to the process, for example by contributing test cases that rely on knowledge of the software items internal implementation.

Software Assurance Witnessing

- The software test procedure developer should dry run the software item test cases and procedures to ensure that they are complete and accurate and that the software is ready for witnessed testing.
- The developer should record the results of this activity in the appropriated software Development folders (SDFs) and should update the software test cases and procedures as appropriate.
- Formal and acceptance software testing are witnessed by software assurance personnel to verify satisfactory completion and outcome.
- Software assurance is required witness or review/audit of software testing and demonstration.

Testing on the Target Computer System

- Software testing should be performed using the target hardware.
- The target hardware used for software qualification testing should be as close as possible to the operational target hardware and should be in a configuration as close as possible to the operational configuration.
- Typically, a high-fidelity simulation has the exact processor, processor performance, timing, memory size, and interfaces as the target system.

SWE-073, "The project manager shall validate the software system on the targeted platform or high-fidelity simulation."

Software Development Facility Line 3

- Flight Software Qualification Line
- 3 Core Stage Engineering Development Unit Flight Computers

Capturing Results

- Capture outcome of tests used to verify requirements, functionality, safety, etc.
- Capture decisions based on outcome of tests
- Provide evidence of thoroughness of testing
  - Differences in test environment and operational environment and any effects those differences had on test results
  - Test anomalies and disposition of related corrective actions or problem reports
  - Details of test results (e.g., test case identifications, test version, completion status, etc., along with associated item tested)
  - Location of original test results (output from tests, screen shots, error messages, etc., as captured during actual testing)

Analyzing Results

Analyze results to:

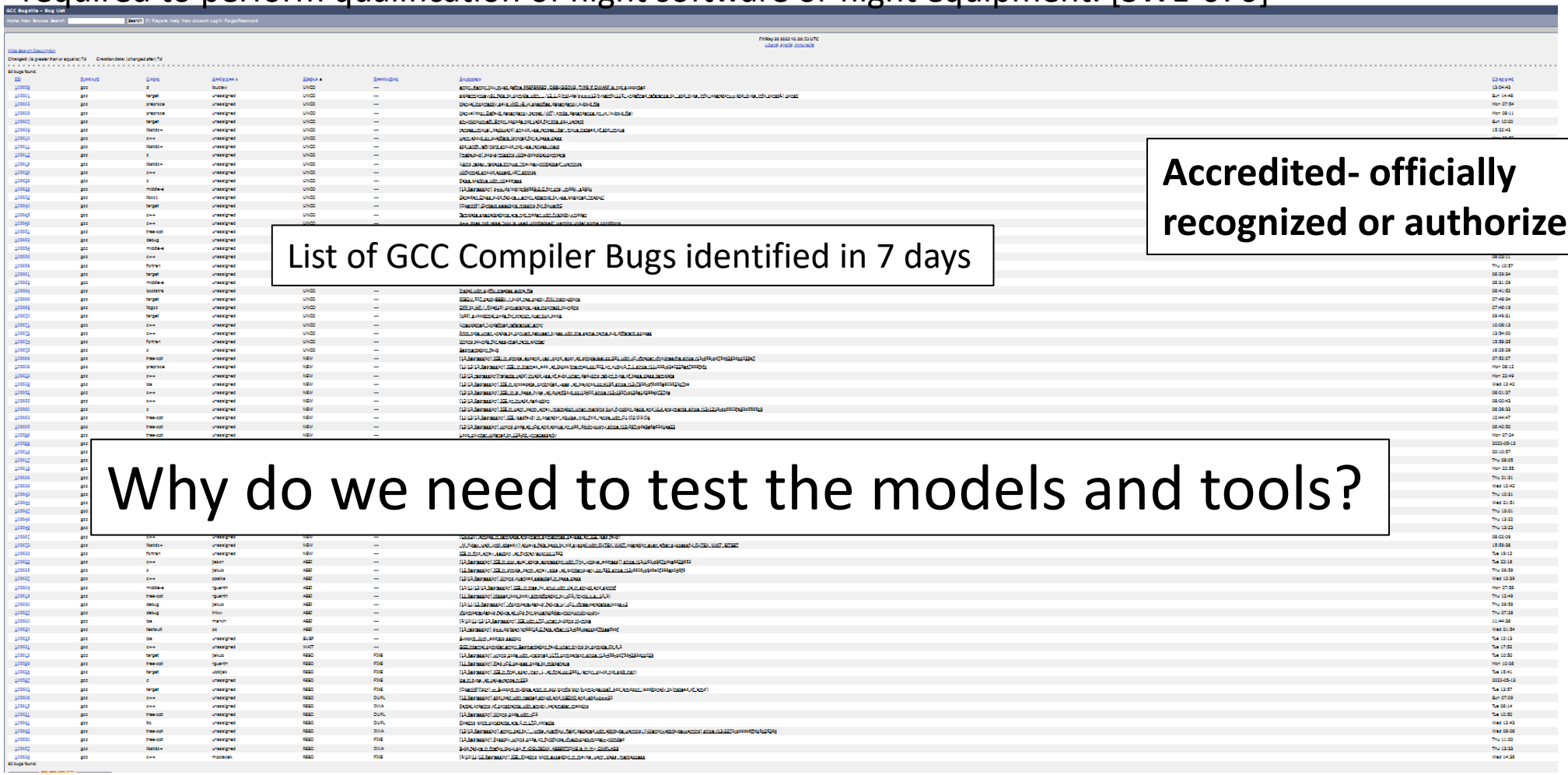
- Evaluate quality of tested products and effectiveness of testing processes
- Identify and isolate source of errors found in software
- Verify testing was completed as planned
- Verify requirements have been satisfied
- Verify safety-critical elements were properly tested
- Verify all identified software hazards eliminated or controlled to acceptable level of risk
- Report safety-critical findings used to update hazard reports

Analyzing Results

- Compare actual to expected results
- Identify discrepancies or mismatches in specification or behavior
- Document discrepancies individually for ease of tracking through the resolution process
- Determine cause of issue, including problems with testing methods, criteria, or environment
- Identify changes required to address discrepancies
- Evaluate and record impact of changes needed to correct issues/discrepancies
- Plan for any repeat of testing effort
- Obtain and record approval for changes to be made versus those to be addressed at different time
- Measure and predict quality of the software based on the testing results (typically, a software assurance activity)

Accredited software models, simulations, and analysis tools

- The project manager shall use validated and accredited software models, simulations, and analysis tools required to perform qualification of flight software or flight equipment. [SWE-070]

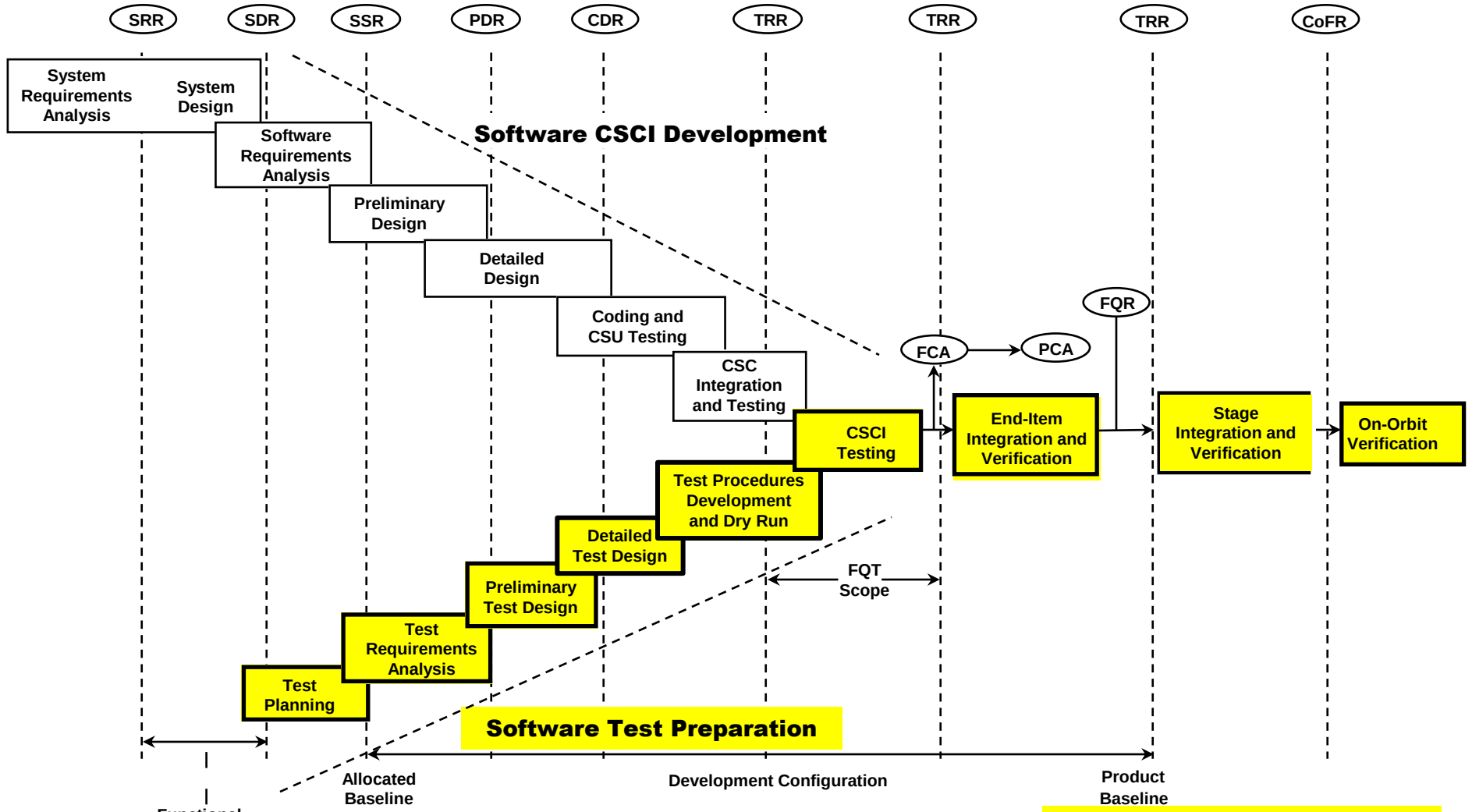


List of GCC Compiler Bugs identified in 7 days

Accredited- officially recognized or authorized.

Why do we need to test the models and tools?

Flight Software Testing Life-Cycle



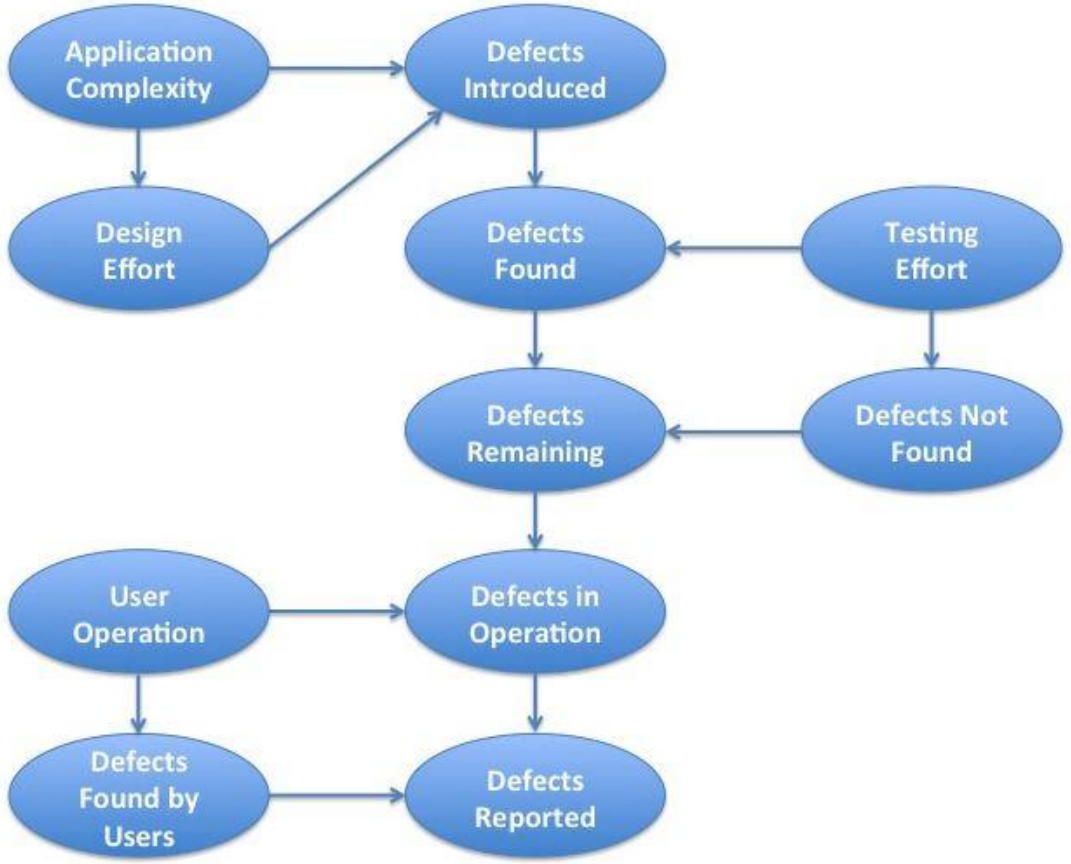
Reviews

SRR - System Requirements Review
SDR - System Design Review
SSR - Software Specification Review
PDR - Preliminary Design Review
CDR - Critical Design Review
TRR - Test Readiness Review
FCA - Functional Configuration Audit
PCA - Physical Configuration Audit
FQR - Formal Qualification Review
CoFR - Certificate of Flight Readiness

Major test cycles applicable throughout software development

Sample Software Test Metrics

- Defects or problem reports found
- Static code analysis metrics
- Code coverage
- Test schedule metrics
- Test Procedure Development Status
- Software Release/Build Status
- Number of tested requirements



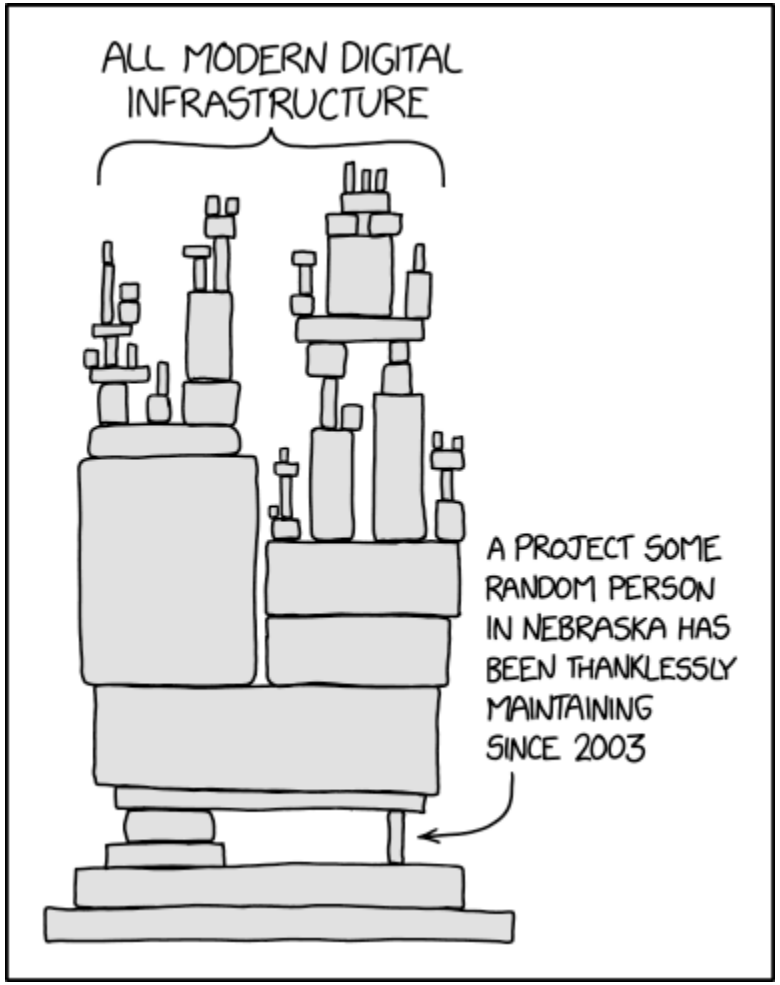
- Traceability – Software Requirements to Test Procedures
- Defects or problem reports open and closed, trending for closure

Summary

- Key points
  - Requirements drive testing
    - Detail in the requirements
    - Derived requirements
  - Testing approach and coverage
  - Testing completeness
  - Data (metrics and measurements)

Software Maintenance

Software Maintenance



Software Operations, Maintenance, and Retirement Requirements

- 4.6.2 The project manager shall plan and implement software operations, maintenance, and retirement activities. [SWE-075]

- 4.6.3 The project manager shall complete and deliver the software product to the customer with appropriate records, including as-built records, to support the operations and maintenance phase of the software's life cycle. [SWE-077]

- 4.6.4 The project manager shall complete, prior to delivery, verification that all software requirements identified for this delivery have been met or dispositioned, that all approved changes have been implemented, and that all defects designated for resolution prior to delivery have been resolved. [SWE-194]

- 4.6.5 The project manager shall maintain the software using standards and processes per the applicable software classification throughout the maintenance phase. [SWE-195]

- 4.6.6 The project manager shall identify the records and software tools to be archived, the location of the archive, and procedures for access to the products for software retirement or disposal. [SWE-196]

Software Maintenance

- The Software Maintenance phase of the software life cycle begins after successful completion of formal test and delivery of the software product to the customer.
- The Software Operation phase spans the time from execution of the software product in the target environment to software retirement.
- The Software Maintenance phase overlaps the Software Operation phase and continues until software retirement or discontinuation of software support
- The results of planning for operations, maintenance and retirement of software are captured in the Software Maintenance Plan for implementation.

Software Delivery

Delivery includes, as applicable, Software User's Manual, source files, executable software, procedures for creating executable software, procedures for modifying the software, and a Software Version Description. Open source software licenses are reviewed by the Center's Chief of Patent/Intellectual Property Counsel before being accepted into software development projects.

Other documentation considered for delivery includes:

a) **Summary and status of all accepted Change Requests to the baselined Software Requirements Specifications.**
b) **Summary and status of all major software capability changes since baselining of the Software Design Documents**
c) **Summary and status of all major software tests (including development, verification, and performance testing).**
d) **Summary and status of all Problem Reports written against the software.**
e) **Summary and status of all software requirements deviations and waivers.**
f) **Summary and status of all software user notes.**
g) **Summary and status of all quality measures historically and for this software.**
h) **Definition of open work, if any.**
i) **Software configuration records defining the verified and validated software, including requirements verification data (e.g., requirements verification matrix).**
j) **Final version of the software documentation, including the final Software Version Description document(s).**
k) **Summary and status of any open software-related risks.**

Operations Support

- Software team support of operations, including help desk activities, as applicable.
- Documentation required for operations support (e.g., as-built documentation, user's manual, source code, operations notes).
- Tools required for operations support (e.g., email systems, servers).
- Availability of problem reporting and corrective action (PRACA) system during operations.
- Participation in mission debriefs, as appropriate.
- Capturing of lessons learned during operations.
- Software assurance, including software safety, monitoring activities.
- Operational backups (e.g., hot backups for critical systems), including identification and planning of approach.

Software Maintenance Support

- Modification of software after delivery.
- Updates to system and software documentation to align with/reflect these modifications.
- Availability and use of a configuration management system for documenting, reviewing, analyzing modifications to code, documentation, and hardware test configurations.
- Tools required for maintenance activities (e.g., issue tracking systems, analysis tools, configuration control systems, compilers).
- Other resources required to perform maintenance activities such as documentation, development environment, test environment.
- Testing of modifications (including pre- and post-delivery).
- Delivery and installation of modifications, including generation of associated documentation such as version description documents (VDDs).
- Capture of maintenance metrics.
- Maintenance transition plan.
- Software assurance and software safety activities for updates.

Software Retirement Support

- Archival of software products, including capture in a configuration management (CM) system.
- Retention period for retired software products.
- Tools needed to complete retirement activities (e.g., CM system).
- Security measures for access to and use of retired software products.
- Transition plans for functionality and data if software being retired is being replaced by another software product.

Measures for Maintenance

Quality and Progress:

- # and severity of software errors,
- # errors opened, assigned, coded, tested, completed (corrections in operational version)
- # of change requests open, approved, assigned, coded, tested, complete
- average # of staff hours to complete (large, medium, small) error correction or change request
- # of staff hours available for maintenance
- # of errors by error type (requirements, operator, coding, interfaces, etc.)

Class Plan

Software's Role and Importance in NASA Missions

NASA Software Engineering & Assurance Policies, Requirements and Resources

Software Planning Requirements and Considerations

- Software Documentation
- Software Costing
- Software Processes
- Software Assurance
- Software Safety-Critical
- Software IV&V
- Software Classifications
- Software Reuse and Internal Sharing
- Software Cybersecurity
- Software Lifecycles and Reviews

Software Life-cycle Requirements

- Software Requirements
- Software Architecture
- Software Design
- Software Coding
- Software Testing
- Software Maintenance

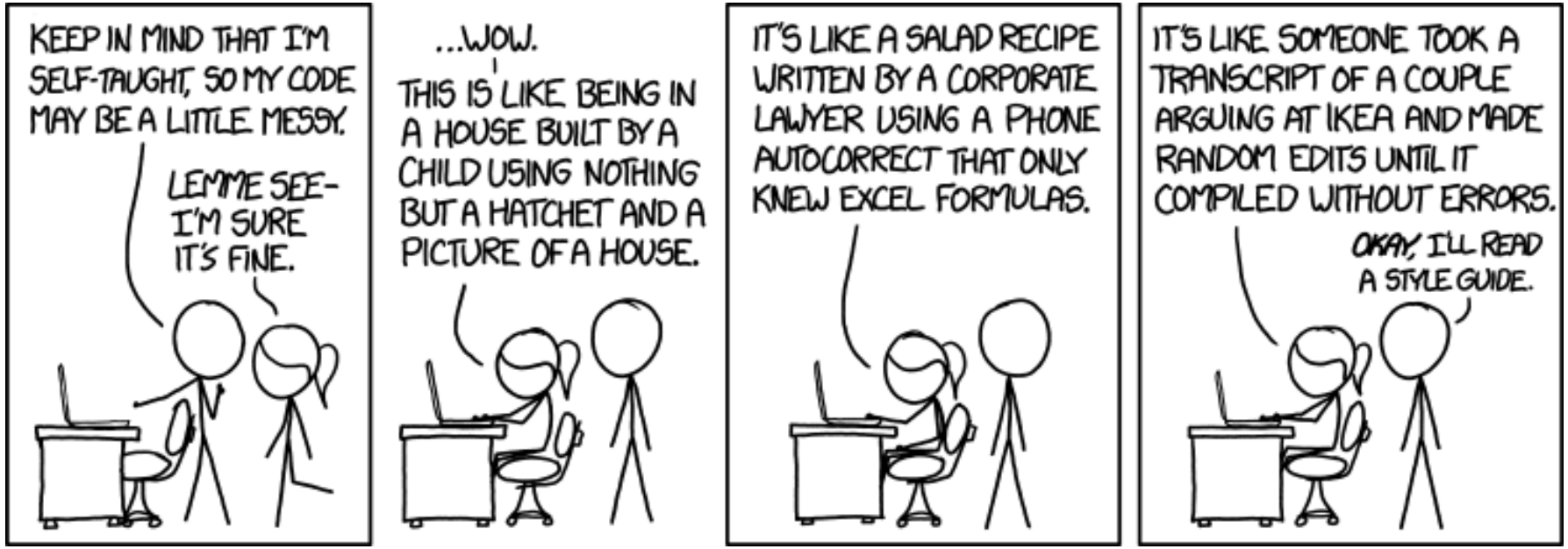
Software Development Supporting Requirements

- Software Peer Reviews
- Software Configuration Management
- Software Risks
- Software Measurements
- Software Defect Management
- Software Bi-Directional Traceability
- Software License Management
- Software Acquisition

Why do we do these things? Software Failures

Peer Reviews/Inspections

Inspection Approaches



Peer Reviews/Inspection Requirements

- 5.3.2 The project manager shall perform and report the results of software peer reviews or software inspections for: [SWE-087]
  - a. Software requirements.
  - b. Software plans, including cybersecurity.
  - c. Any design items that the project identified for software peer review or software inspections according to the software development plans.
  - d. Software code as defined in the software and or project plans.
  - e. Software test procedures.

  .
- 5.3.3 The project manager shall, for each planned software peer review or software inspection: [SWE-088]
  - a. Use a checklist or formal reading technique (e.g., perspective based reading) to evaluate the work products.
  - b. Use established readiness and completion criteria.
  - c. Track actions identified in the reviews until they are resolved.
  - d. Identify the required participants.
- 5.3.4 The project manager shall, for each planned software peer review or software inspection, record necessary measurements. [SWE-089]

Linus's Law

- Linus's Law is a claim about software development, named in honor of Linus Torvalds and formulated by Eric S. Raymond in his essay and book The Cathedral and the Bazaar (1999).
- **The law states that "given enough eyeballs, all bugs are shallow"; or more formally: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone."**
- Presenting the code to multiple developers with the purpose of reaching consensus about its acceptance is a simple form of software reviewing.
- Researchers and practitioners have repeatedly shown the effectiveness of various types of reviewing process in finding bugs and security issues, and also that code reviews may be more efficient than testing.

Defect Removal Efficiency

Table 4: Software Defect Removal Efficiency Ranges

Defect Removal Activity	Ranges of Defect Removal Efficiency
Formal requirement inspections	50% to 90%
Formal design inspections	45% to 85%
Formal code inspections	45% to 85%
Static analysis (automated)	55% to 90%
Unit test (manual)	15% to 50%
Unit test (automated)	20% to 60%
New function test	20% to 35%
Regression test	15% to 30%
Integration test	25% to 40%
Performance test	20% to 40%
System test	25% to 55%
Acceptance test (1 client)	25% to 35%
Low-volume Beta test (< 10 clients)	25% to 40%
Overall cumulative ranges	70% to 99%

Caper Jones DACS Software Tech News March 2010

Products for Peer Reviews

- NPR 7150.2 requires certain products to be inspected/peer reviewed
- Required software products depend on the classification of the project.

	Software Class					
Software Documentation	**A**	**B**	**C**	**D**	**E**	**F**
Software Requirements	X	X	X			X
Software Plans	X	X	X			X
Software Design Identified in Plans	X	X	X			X
Software Code identified in Plans	X	X	X			X
Test Procedures	X	X	X			X

Benefits

Among the most effective verification and validation practices for software	Useful for many types of products: documentation, requirements, designs, code
Simple to understand	Provide a way for sharing/learning of good product development techniques
Can result in very efficient method of identifying defects early in the product's life cycle	Serve to bring together human judgment and analysis from diverse stakeholders in a constructive way
Use a straight-forward, organized approach for evaluating a work product - **To detect potential defects in a product** - **To methodically evaluate each defect to identify solutions and track incorporation of these solutions into the product**	Can impact budgets: defects found and fixed early (rather than allowed to slip into later phases) cost less and require less rework
Add value and reduce risk through expert knowledge, infusion, confirmation of approach, identification of defects, and specific suggestions for product improvements – NPR 7123.1 (G.19)	One of the few V&V approaches that can be applied in the early stages of software development (before there is any code that can be run and tested)

Process

- Effective peer reviews/inspections
  - Are concerned with only the technical integrity and quality of the product
  - Are simple and informal
  - Concentrate on review of the documentation and minimize presentations
  - Use a round-table format rather than a stand-up presentation
  - Give a full technical picture of items being reviewed
  - Are planned, use checklists, include readiness and completion criteria
  - Capture action items, monitor defects, results, effort
- Team Reviews
  - Team of 5-7 people
  - Material reviewed in advance of meeting
  - Author leads review meeting
  - Solutions discussed and attempt made to choose best one
  - No follow-up on identified issues
  - Effectiveness measures collected
- Walkthroughs
  - Author prepares review material
  - Good for educating others on the material
  - Solutions often discussed as part of the review
  - No follow-up on identified issues
  - No effectiveness measures

Process

- **Best Practices – Process**
  - Defects found during inspections never used to evaluate author – goal is to improve product
  - Use checklists relevant to inspector's perspective
  - Use readiness and completion criteria
  - Limit inspection meeting to 2 hours
  - Track action items until resolved
  - Collect and use inspection data
    - Effort, number of participants, areas of expertise
    - Defects - list, total, type
    - Inspection outcome (pass/fail)
    - Item being inspection and type (requirements, code, etc.)
    - Date and time
    - Meeting length, preparation time of participants

Software Peer Review Base Metrics

Category	Base Metric	Description
Size	Size planned	Lines of code or document pages that you planned to inspection
Size	Size Actual	Lines of code or documents pages that were actually inspected or peer reviewed
Time	Time Meeting	Time required to complete the inspection, if done over several meetings then add up the total time required
Effort	Planning	Total number of hours spend planning and preparation for the review
	Meeting time	Total number of hours spent in the inspection meeting (multiply Time meeting by number of participates
	Rework	Total number of hours spend by the author making improvements based on the findings.
Defects	Major Defects found	Number of Major defects found during the review
	Minor Defect found	Number of Minor defects found during the review
	Major Defects Corrected	Number of major defect corrected during rework
	Minor Defects Corrected	Number of minor defect corrected during rework
Other	Number of Inspectors	Number of people, not counting observers, who participated in the review
	Product Appraisal	Review teams assessment of the work product (accepted, accepted conditionally, review again following rework, review not complete, etc.)
Derived Data	Peer Review Defects	The Peer Review Defect metric measures the average number of defects per peer review to determine defect density over time. Number of defects found per Peer Review = [Total number of defects] / [To number of Peer Reviews]

Summary for Ensuring Quality in Your Project

Remember … it's cheaper to build quality products than to go back and fix the problems

- Make sure your team understands the processes and implements them as defined
- Include quality activities in your plan and track their progress
- Have objective evaluators assess the Team's adherence to documented process and product standards

Software Configuration Management

NPR 7150 Software Configuration Management Requirements

- 5.1.2 The project manager shall develop a ***software configuration management plan*** that describes the functions, responsibilities, and authority for the implementation of software configuration management for the project. [SWE-079]

- 5.1.3 The project manager ***shall track and evaluate changes to software products.*** [SWE-080]

- 5.1.4 The project manager ***shall identify the software configuration items (e.g., software records, code, data, tools, models, scripts) and their versions to be controlled for the project.*** [SWE-081]

- *Note: The items to be controlled include tools, items, or settings used to develop the software, which could impact the software. Examples of such items include compiler/assembler versions, makefiles, batch files, and specific environment settings.*

NPR 7150 Software Configuration Management Requirements

- 5.1.5 The project manager shall establish and implement procedures to: [SWE-082]
- ***a. Designate the levels of control through which each identified software configuration item is required to pass.***
- ***b. Identify the persons or groups with authority to authorize changes.***
- ***c. Identify the persons or groups to make changes at each level.***
- 5.1.6 The project manager shall ***prepare and maintain records of the configuration status of software configuration items.*** [SWE-083]

NPR 7150 Software Configuration Management Requirements

- 5.1.7 The project manager shall ***perform software configuration audits*** to determine the correct version of the software configuration items and verify that they conform to the records that define them. [SWE-084]

- 5.1.8 The project manager shall establish and ***implement procedures for the storage, handling, delivery, release, and maintenance of deliverable software products.*** [SWE-085]

- 5.1.9 The project manager shall ***participate in any joint NASA/developer audits***. [SWE-045]

- 4.5.4 The project manager shall ***place software items under configuration management prior to testing.*** [SWE-187]

- *Note: This includes the software components being tested and the software components being used to test the software, including components like support software, models, simulations, ground support software, COTS, GOTS, MOTS, OSS, or reused software components.*

SAE/EIA-649B Configuration Management Standard

- SAE/EIA-649B Configuration Management Standard is the NASA CM standard
- A companion standard (EIA-649-2) to "SAE/EIA-649B Configuration Management Standard," provides a resource that standardizes Configuration Management (CM) requirements specific to National Aeronautics and Space Administration (NASA) agreements and design activities.
- This provides a template of CM requirements and user guidance for tailoring the requirements for each unique use case.

3.3.5 Software Change Control

For software, the customer controls the CSCI requirements (design specifications) and release to include all associated software documentation (i.e., Version Description Document (VDD), manuals, guides) and products (i.e., code, databases, PLDs). The suppliers have the responsibility to establish hardware and software integrated control authorities (control boards) to ensure the evaluation of all changes affecting the software within an integrated CI/CSCI product structure.

Both hardware and software deliverables are released using the same baseline definitions and functions described in this Standard.

(1) The Supplier shall prepare a VDD as specified in the agreement DRD-STD-VDD.

(2) The Supplier shall comply with NPR 7150.2B, Section 4.1.

Should be Section 5.1

Software Configuration Management

- Software Configuration Management is the process of applying configuration management throughout the software life cycle to ensure the completeness and correctness of software configuration items.
- SCM applies technical and administrative direction and surveillance to:
  - identify and record the functional and physical characteristics of software configuration items,
  - control changes to those characteristics,
  - record and report change processing and implementation status,
  - verify compliance with specified requirements.
- SCM establishes and maintains the integrity of the products of a software project throughout the software life cycle.
- Use of standard Center or organizational SCM processes and procedures is encouraged where applicable.

Configuration Items

Deliverable and non-deliverable software development products	
Documentation (plans, standards)	Source code
Object code	Executable
Data	Development and test tools (operating systems, compilers, etc.)
Development and test environments	Test cases/scenarios, data, scripts, reports
Flow charts, UML, input to code generators	COTS software
Build procedures	Defect lists, change requests
Metrics	Software assurance records
Requirements	Simulators, models, test suites
Interface documents	Databases
Training materials	Baselines and identification of their contents
Specifications	Traceability matrices
Presentations	Release notes

Change Control

- Levels of control configuration items must pass through
  - May differ by item type (e.g., documentation, code)
- Persons or groups with authority to authorize changes and to make changes at each level
  - Change control boards
  - Change authorization boards
  - Engineering change boards
  - Peer review teams
  - Project managers

Audits

- Provide checks to ensure that the planned product is the developed product… determine correct version of configuration items and verify they conform to documents and requirements that define them
- Performed
  - At time product released
  - Prior to delivery (assure products are complete, contain proper versions and revisions, and all discrepancies, open work, deviations and waivers properly documented and approved)
  - At end of a life cycle phase
  - Prior to release of new or revised baseline
  - As project progresses (prevent finding major issues at end when more costly to fix)
  - Incrementally for very large, complex systems focusing on specific functional areas with a summary audit to address status of identified action items

Software Risk Identification and Management

Software Risk Requirement in NPR 7150.2

Software Risk Requirement

- 5.2.1 The project manager shall ***record, analyze, plan, track, control, and communicate all of the software risks and mitigation plans***. [SWE-086]

Remember to Plan for Risk Management

- Risk Management means:
  - Identifying risks that threaten success of the project
  - Analyzing the risks to gain understanding and develop possible mitigations
  - Tracking the risks as conditions change
  - Communicating risk status to management
- Why should you do this?
  - Because surprises are usually unpleasant and this minimizes surprises
  - Because the earlier a potential problem is acknowledged and the more you know about it, the better you can deal with it
  - Because it's also an Agency requirement!

Software Risk Requirement Rationale

- The purpose of risk management is to identify potential problems before they occur so that risk handling activities can be planned and invoked as needed across the life of the product or project.
- Risk handling activities are intended to mitigate adverse impacts on achieving the project's objectives.
- "Generically, risk management is a set of activities aimed at achieving success by proactively risk-informing the selection of decision alternatives and then managing the implementation risks associated with the selected alternative."
- Identification and management of risks provide a basis for systematically examining changing situations over time to uncover and correct circumstances that impact the ability of the project to meet its objectives.

Use a Checklist to Help Identify Software Risk Items

ISWE

Project Development Phase:	**RISK Yes/No /Partial**	**ACTION Accept/ Work**
System Requirements Phase		
Are system level requirements documented? To what level? Are they clear, unambiguous, verifiable ?		
Is there a project wide method for dealing with future requirements changes?		
Have software requirements been clearly delineated/allocated?		
Have these system level software requirements been reviewed, inspected with systems, hardware and the users to insure clarity and completeness?		
Has Firmware and Software been differentiated, who is in charge of what and is there good coordination if H/W is doing "F/W"?		
Are the effects on command latency and its ramifications on controllability known?		
Can the Bus bandwidth support projected data packet transfers?		
Are requirements defined for loss of power? System reaction known or planned for? UPS (Uninteruptable Power Supplies) planned for critical components?		
Is an impact analysis conducted for all changes to baseline requirements		

Software Design Phase	**RISK**	**ACTION**
Is the Software Management Plan being followed? Does it need updating?		
Is the Requirements flow down well understood?		
Standards and guidelines sufficient to produce clear, consistent design and code?		
Will there be, has there been, a major loss of personnel (or loss of critical personnel)?		
Communication between systems and other groups (avionics, fluids, operations, ground software, testing, QA, etc.) and Software working well both directions?		
Requirements Have they been baselined & are they configuration managed? Is it known who is in charge of them? Is there a clear, traced, managed way to implement changes to the requirements? (i.e. is there a mechanism for in-putting new requirements, or altering old, established and working)? Is there sufficient communication between those creating & maintaining requirements and those designing to them ? Is there a traceability matrix between requirements and design? Does that traceability matrix show the link from requirements to design and then to the appropriate test procedures?		
Has System Safety assessed Software? Any software involved hazard reports? Does software have the S/W subsystem hazard analysis? Does software personnel known how to address safety critical functions, how to design to mitigate safety risk? Are there Fault Detection, Isolation and Recovery (FDIR) techniques designed for critical software functions?		
Has software reliability been designed for? What level of fault tolerance has been built in to various portions /functions of software?		
Need to create Simulators to test software? Were these simulators planned for in the schedule? Is there sufficient resources to create, verify and run these? How heavily does software completion rely on simulators? How valid (close to the flight) are the simulators?		

https://nen.nasa.gov/web/software/wiki/-/wiki/SPAN/Risk+Management

Identifying Risks

- Risks have two main parts: a condition, and a consequence
  - Condition: the event that might happen
  - Consequence: the effect on the project if it does
  - Often phrased as: “If condition, then consequence”
- Examples:
  - If the simulator doesn't arrive on time, then the start of testing will be delayed
  - We were promised staff coming off project x, but project x has been delayed. If we don’t get the promised staff, then our development effort may not be able to meet its schedule commitments
- Classify each risk after it is identified

Software Risk Identification Steps

- When identifying software risks, consider the following insights and suggestions:
  - Identify risks before they become problems.
  - Communication is the center of the Risk Management paradigm (see NPR 8000.4, Agency Risk Management Procedures and Guidelines).
  - Brainstorming is often used to identify project risks.
    - People from varying backgrounds and points-of-view see different risks.
    - A diverse team, skilled in communication, will usually find better solutions to the problems."
  - Use a checklist to avoid "missing" risks that have been identified on previous projects.
    - Use existing reference lists; NASA/SP-2007-6105, NASA Systems Engineering Handbook, includes a list of example sources of risk.

Software Risk Management Steps – Track, Control, Communicate

- Track software risks
  - Risks that are not eliminated need to be tracked throughout the project life cycle to ensure their mitigation strategies remain effective.
  - For low-risk items that are not formally included in the risk management plan, consider using a watch list so that they are not forgotten and to help ensure that they do not escalate to a higher level risk later in the project.
  - Additionally, conditions that the team has identified as risk triggers are also monitored and tracked until those situations are no longer risk factors. Risk status also needs to be tracked and weighed against risk criteria to determine if corrective action needs to be taken.
  - If a risk management tool is in use for the project, risks need to be added to and tracked using this tool. A tracking tool could be a simple spreadsheet or database for a small project, a tool purchased specifically for tracking risks, or part of an integrated tool used to track multiple aspects of the project.

Software Risk Management Steps – Track, Control, Communicate

- Control software risks
  - When a risk occurs, action needs to be taken. Those actions should have been included in the risk management plan and need to be implemented in this step. Their effectiveness also needs to be measured so adjustments to the plan can be made, if necessary.
- Communicate software risk information
  - Risk information is communicated to all relevant stakeholders throughout the project life cycle. Stakeholders include project managers, project technical personnel, test team members, and anyone else affected by or with the need to know about risks, their impact, and their mitigations. Project life cycle reviews are one mechanism for risk communication.

Software Measurements

Why Measure? - 1

NPR 7150.2D Requirements on Software Requirements

- 5.4.2 The project manager shall **establish, record, maintain, report, and utilize software management and technical measurements.** [SWE-090]
- 5.4.3 The project manager shall **analyze software measurement data collected** using documented project-specified and Center/organizational analysis procedures. [SWE-093]
- 5.4.4 The project manager shall **provide access to the software measurement data**, measurement analyses, and software development status as requested to the sponsoring Mission Directorate, the NASA Chief Engineer, the Center Technical Authorities, HQ SMA, and other organizations as appropriate. [SWE-094]
- 5.4.5 The project manager shall **monitor measures to ensure the software will meet or exceed performance and functionality requirements, including satisfying constraints.** [SWE-199]
- 5.4.6 The project manager shall collect, track, and report **software requirements volatility metrics**. [SWE-200]

A Thought!

"What gets measured, gets managed."

- Peter Drucker

There is so much power in this quote. If you've never tracked yourself, you don't even know how much power there is in tracking. I couldn't even explain it adequately. You wouldn't believe me. You'd think I was exaggerating. The simple act of paying attention to something will cause you to make connections you never did before, and you'll improve those areas - almost without any extra effort.

Why You Should Measure

- For the benefit of your current project
  - Use objective measurement data to plan, track, and correct project
- For the benefit of your future projects (and the rest of your Center's projects, too!)
  - Help create a basis for planning future projects
  - Help understand what baseline performance is for projects similar to yours
  - Provide organizational information to help improve software activities

Why Do Technical Performance Measurement?

ISWE

- Cost and schedule performance status is of little value unless the technical performance is acceptable
- We need to track potential risks and verify technical assumptions or estimates behind the plan, such as
  - Our productivity rate projections
  - Product size estimates
  - Product complexity estimates
  - Product performance assumptions
- We need to measure acceptability “as we go,”
  - Trends in production rates
  - Trends in performance
  - Interim acceptability

And a Few More Reasons to Measure---

- Forces advanced, detailed planning
- Helps in making development and management planning decisions consistent with the project scope and requirements
- Provides an objectivity in assessing progress which is often difficult during the heat of the battle
- Provides status relative to approved scope and requirements to support management control
- Allows corrective action in time to prevent the “crisis” or to minimize the impact of the crisis
- Improves ability to estimate completion costs and schedule variances by analysis of data and trends

BASIC Software Measurements

- Code Size (LSLOC)
  - Use a standard counter
  - Deliverable Code
  - Test Code
  - Comments
- Release Dates
  - Date and Code Sizes
- Defect Reports by date
  - Cumulative defects for your product by date of releases
- Effort
  - To repair
  - To implement a feature

- IEEE Software Magazine Jan/Feb 2018

Components of a Measurement Plan

1. Measurement objectives
2. The measures that will meet the objectives (and don't forget measures for the process areas)
3. Descriptions of how the measures will be collected and stored
4. The analysis methods for each of the measures
5. Communication of the measurement results
6. Commitment to the measurement plan from your team and your management

Candidate Management Indicators That Might Be Used On A Software Development Project:

- Requirements volatility: total number of requirements and requirement changes over time.
- Bidirectional traceability: Percentage complete of System level requirements to Software Requirements, Software Requirements to Design, Design to Code, Software Requirements to Test Procedures
- Software size: planned and actual number of units, lines of code, or other size measurement over time.
- Software staffing: planned and actual staffing levels over time.
- Software complexity: complexity of each software unit.
- Software progress: planned and actual number of software units designed, implemented, unit tested, and integrated overtime, code developed.
- Problem/change report status: total number, number closed, number opened in the current reporting period, age, severity.
- Software test coverage: a measure used to describe the degree to which the source code of a project is tested by a particular test suite
- Build release content: planned and actual number of software units released in each build.
- Build release volatility: planned and actual number of software requirements implemented in each build.
- Computer hardware and data resource utilization: planned and actual use of computer hardware resources over time.
- Milestone performance: planned and actual dates of key project milestones.
- Scrap/rework: amount of resources expended to replace or revise software products after they are placed under any level of configuration control above the individual author/developer level.
- Effect of reuse: a breakout of each of the indicators above for reused versus new software products.
- Cost performance: identifies how efficiently the project team has turned costs into progress to date.
- Budgeted cost of work performed: identifies the cumulative work that has been delivered to date.
- Audit performance: Are you following a defined processes, how many audits have been completed, audit findings, audit findings open/close numbers
- Risk Mitigation: Number of identified software risks, risk migration status
- Hazard analysis: number of hazard analysis completed, hazards mitigation steps addressed in software requirements and design, number of mitigation steps tested

Mapping of Organizational Goals to Metrics

ISWE

		Mapping of Organizational Goals to Metrics	
Goal Statements	Goal	Question	SA Metric
Continuously improve the quality of software requirements to assure safe and secure products are delivered in support of mission success and customer objectives.	Quality Software Requirements	Are the software requirements detailed enough for development and test?	Ratio of the number of detailed software requirements to the number of SLOC to be developed by the project.
			Percentage complete of each area of traceability.
		Are requirements stable?	Software requirements volatility trended after project baseline (e.g., # of requirements added, deleted, or modified; tbds).
		Do the software requirements adequately address the software hazards ?	Percentage complete of traceability to each hazard with software items. (New)
Assure quality, safe, and secure code is being delivered.	Quality Code	Is the code secure and has the code addressed cybersecurity requirements?	Number of cybersecurity secure coding violations per number of developed lines of code;
			List of types of secure coding violations found.
		Is the safety-critical code safe?	Software cyclomatic complexity data for all identified safety-critical software component;
		What is the quality of the code?	Number of defects or issues found in the software after delivery;
			Number of defects or non-conformances found in flight code, ground code, tools, and COTs products used.
		Do the requirements adequately address cybersecurity?	Number and type of identified cybersecurity vulnerabilities and weaknesses found by project.

Acquisition Considerations: Measuring the Contractor's Work

- Measurement must be part of deliverables
  - Make sure you specify a good set of measures in the RFP -- you can negotiate minor changes later if necessary
  - Amend existing contracts (eventually) to define measures
  - Generally should use the same sort of measures as in-house projects, e.g.,
    - Contractor earned value reports may cover software progress measures
    - Planned and actual delivery dates
    - Test results or count of outstanding problems

Acquisition Considerations: Measuring Government Work

- Should have acquisition process measures for Class A and B projects
  - For example, planned and actual effort
- Consider other objectives as well
  - Assure that government completes work on time
    - How long does contract / amendment take in the procurement office?
    - How long does it take to accept deliveries?
  - Assure quality of government work
    - Are requirements complete and stable?
    - Are acquisition processes passing audits?

Repeat The Thought

"What gets measured, gets managed." - Peter Drucker

There is so much power in this quote. If you've never tracked yourself, you don't even know how much power there is in tracking. I couldn't even explain it adequately. You wouldn't believe me. You'd think I was exaggerating. The simple act of paying attention to something will cause you to make connections you never did before, and you'll improve those areas - almost without any extra effort.

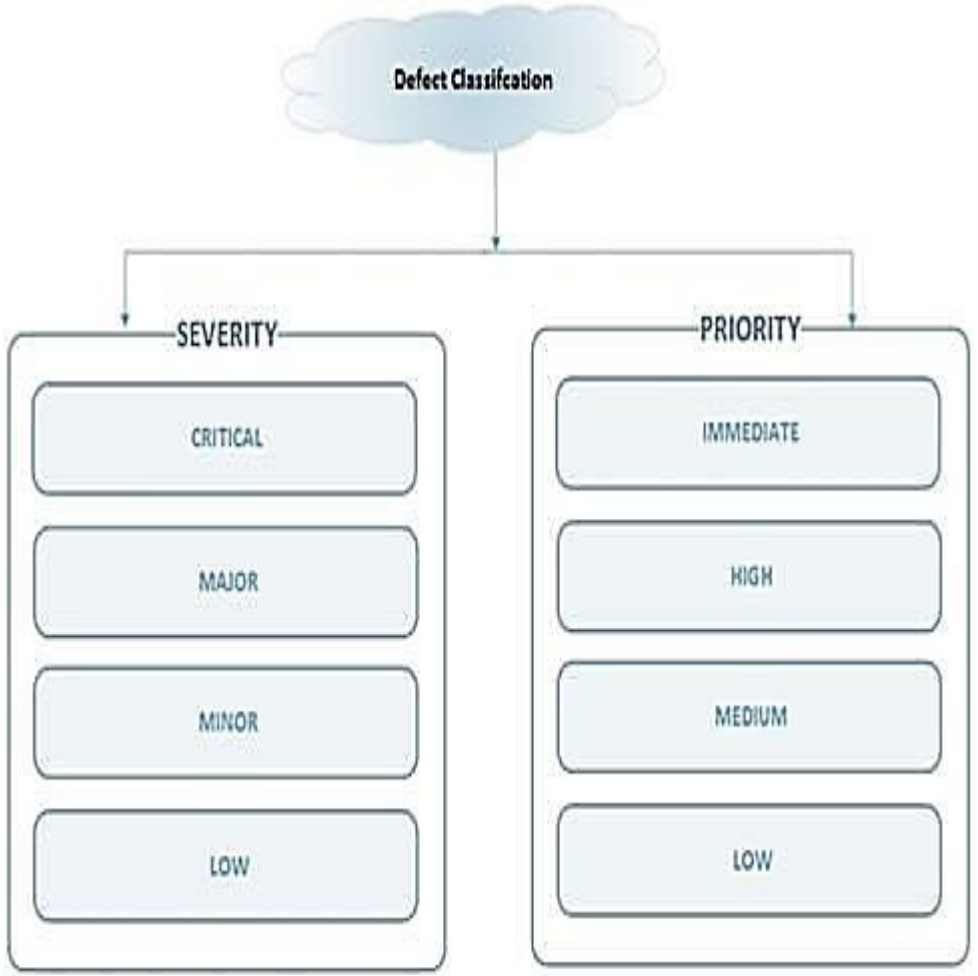
Summary for Software Measurements

- Some of the important features and advantages of metrics are:
  - Motivation – Involving employees in the whole process of goal setting and increasing employee empowerment. This increases employee job satisfaction and commitment.
  - Better communication and coordination – Frequent reviews and interactions between superiors and subordinates helps to maintain harmonious relationships within the organization and also to solve many problems.
  - Clarity of goals
  - Subordinates tend to have a higher commitment to objectives they set for themselves than those imposed on them by another person.
  - Managers can ensure that objectives of the subordinates are linked to the organization's objectives.
  - Everybody will be having a common goal for whole organization. That means, it is a directive principle of management.
- Measure your project performance quantitatively
- Believe the data, especially the trends
- Analyze the causes of trends - and do something about them
- Identify and track key technical performance parameters
- Exercise management judgment - use the data to control your project

Software Non-conformance or Defect Management

Software Non-conformance or Defect Management

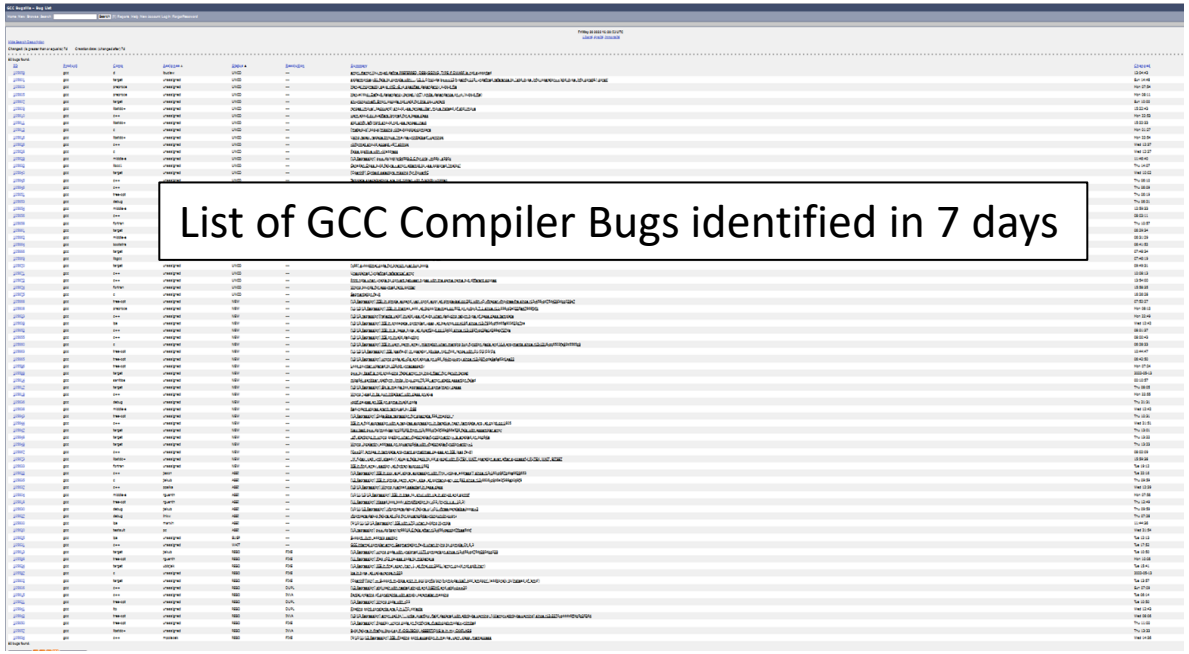
- 5.5.1 The project manager shall track and maintain software non-conformances (including defects in tools and appropriate ground software). [SWE-201]
- 5.5.2 The project manager shall define and implement clear software severity levels for all software non-conformances (including tools, COTS, GOTS, MOTS, OSS, reused software components, and applicable ground systems). [SWE-202]
- *Note: At a minimum, classes should include loss of life or loss of vehicle, mission success, visible to the user with operational workarounds, and an 'other' class that does not meet previous criteria.*



Software Non-conformance or Defect Management

- 5.5.3 The project manager shall implement mandatory assessments of reported non-conformances for all COTS, GOTS, MOTS, OSS, and/or reused software components. [SWE-203]

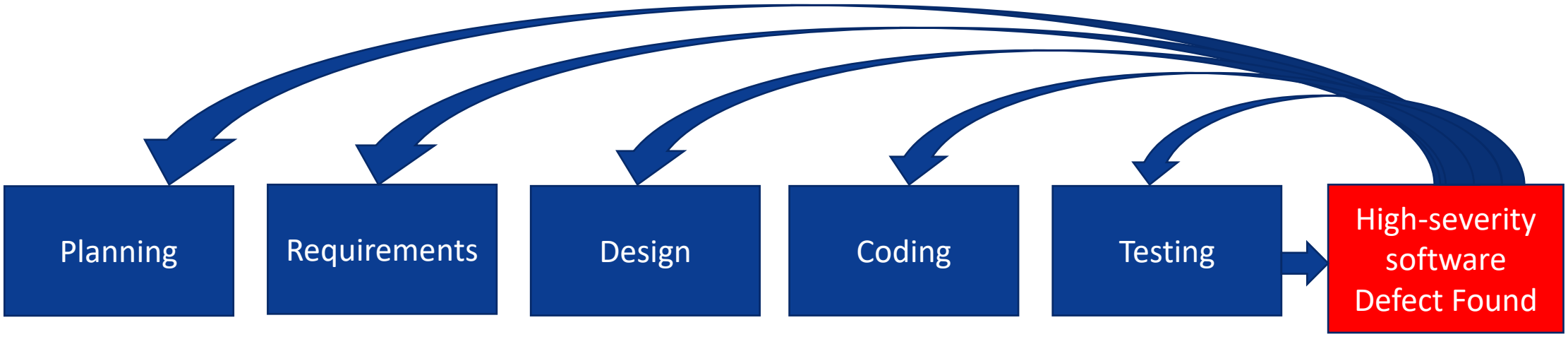
- *Note: This includes operating systems, run-time systems, device drivers, code generators, compilers, math libraries, and build and Configuration Management (CM) tools. It should be performed pre-flight, with mandatory code audits for critical defects.*



List of GCC Compiler Bugs identified in 7 days

Software Non-conformance or Defect Management

- 5.5.4 The project manager shall implement process assessments for all high-severity software non-conformances (closed loop process). [SWE-204]

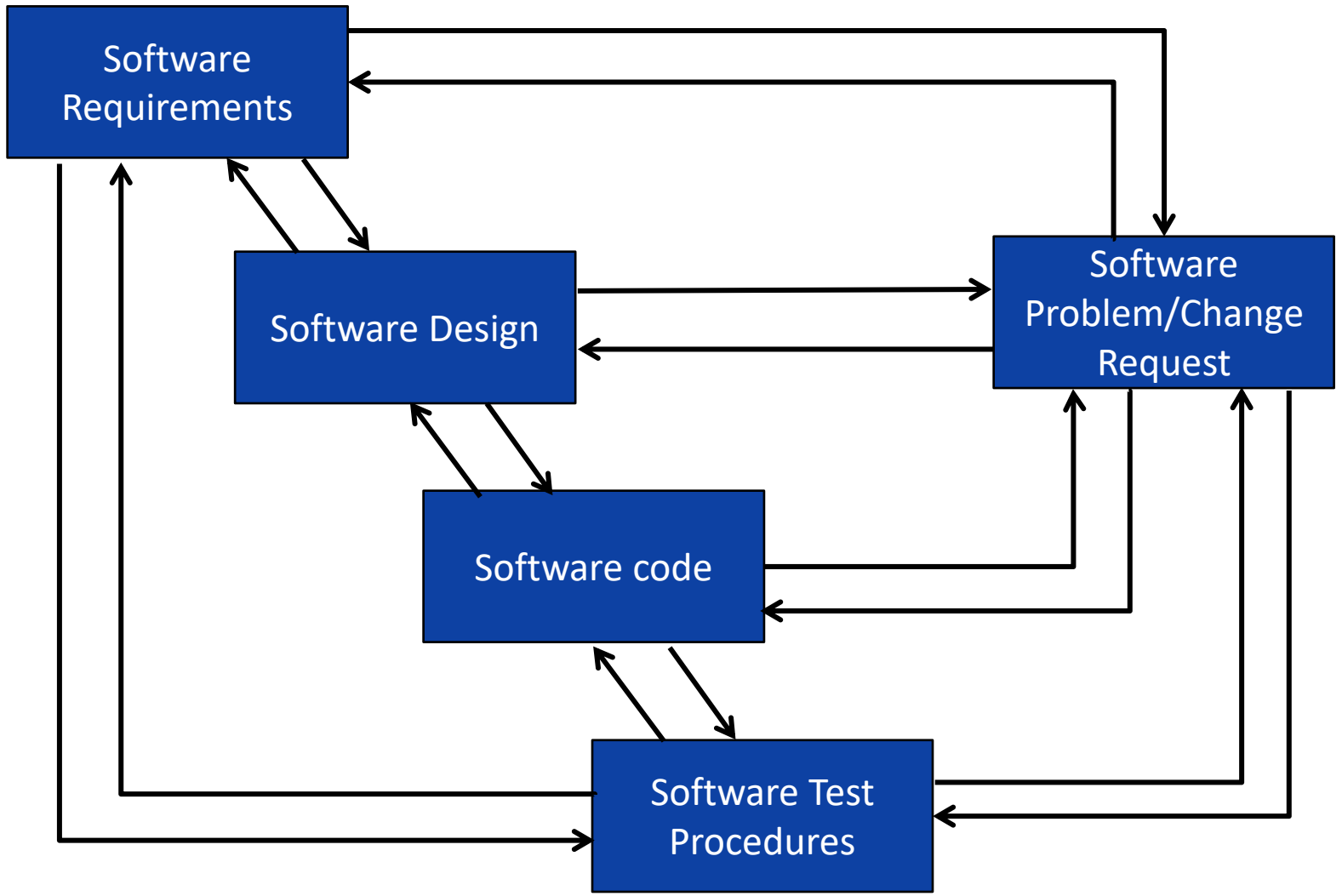


What caused the High-severity software Defect

Bidirectional Traceability

Bidirectional Traceability

Bidirectional traceability is defined as a traceability chain that can be traced in both the forward and backward directions



Bi-directional Traceability Requirement

3.12.1 The project manager shall perform, record, and maintain bi-directional traceability between the following software elements: [SWE-052]

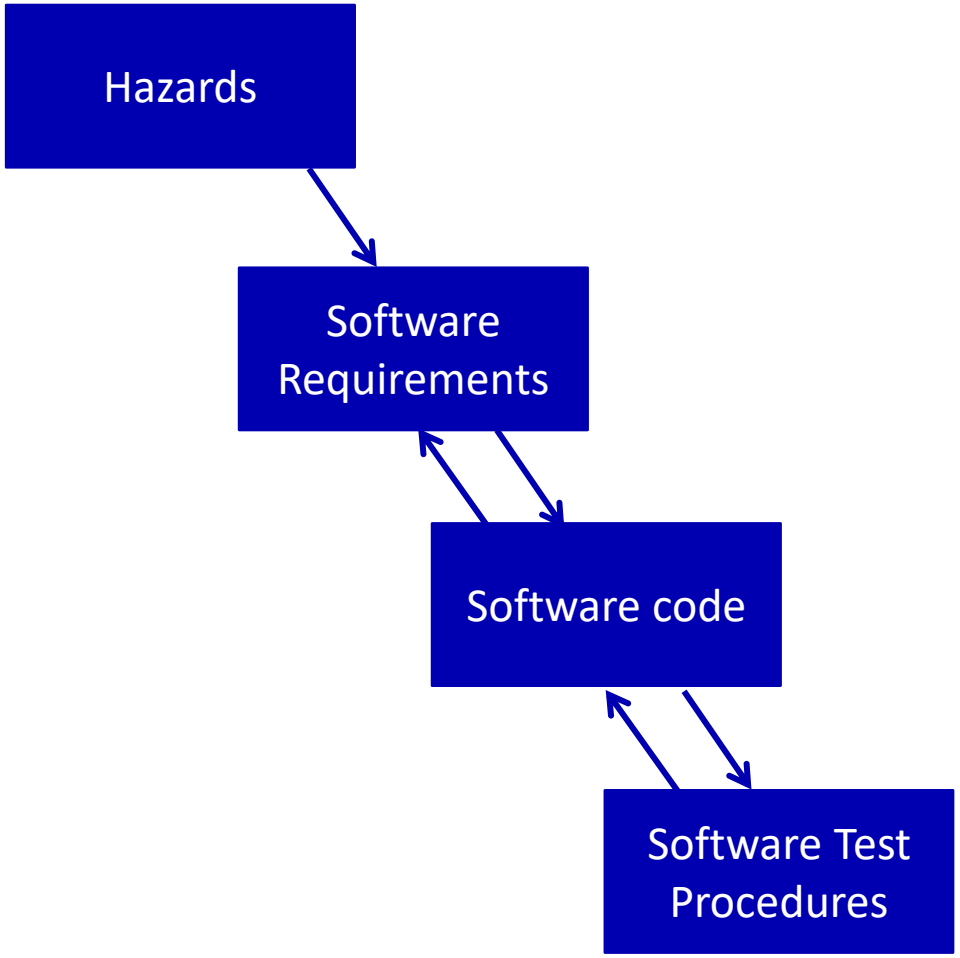
Note: The project manager will maintain bi-directional traceability between the software requirements and software-related system hazards, including hazardous controls, hazardous mitigations, hazardous conditions, and hazardous events.

Bi-directional Traceability	Class A, B, and C	Class D	Class F
Higher-level requirements to the software requirements	X		X
Software requirements to the system hazards	X	X	
Software requirements to the software design components	X		
Software design components to the software code	X		
Software requirements to the software test procedures	X	X	X
Software requirements to the software non-conformances	X		X

Safety-Critical Software Requirements

The project manager shall perform, record, and maintain bi-directional traceability between the following software elements: [SWE-052]

Software requirements to the system hazards



Software Requirement Sources

Other Software Requirement Sources

Hardware specifications
Computer\Processor\Programmable Logic Device specifications
Hardware interfaces
Operating system requirements and board support packages
Data\File definitions and interfaces
Communication interfaces including bus communications **Software interfaces**
Derived from Domain Analysis
Fault Detection, Isolation and Recovery requirements
Models
Commercial Software interfaces and functional requirements
Software Security Requirements
User Interface Requirements
Algorithms
Legacy or Reuse software requirements
Derived from Operational Analysis
Prototyping activities
Interviews
Surveys
Questionnaires
Brainstorming
Observation
Software Test Requirements
Software Fault Management Requirements
Hazard Analysis

System Requirements

Software Requirements

Software Licensing

The Problem: Why We Need Software Licensing Management

- No Agency/Enterprise Approach
- Limited Insight/Visibility
- Inefficiencies in Internal Planning

ANY software that has deployed within the Agency regardless of its function injects **RISK** into NASA's infrastructure from an IT Security and software compliance perspective…

…which has the potential to cost NASA **MILLIONS** per year in **WASTE**, **INEFFICIENCY**, and **NON-COMPLIANCE FEES**……

Benefits of Knowing Inventory

Cost Avoidance/Savings

- Increased opportunity to reduce software cost and avoid paying significantly higher prices
- Maximize software investments without purchasing new products and support
- Decreased costs for new projects and maintaining existing systems
- Cost avoidance on audits, cyber threats

Cyber Threat Detection

- Increased awareness of cyber threats to NASA's data (i.e., mission and personnel)
- Quick removal of legacy IT hardware/software decreases threat possibility
- Identify and mitigate outdated/unsupported software

Audit Prevention

- Reduced chance of audits, or of negative outcomes from audits
- Compliance with OFPP, GAO and OMB guidance on Software license management

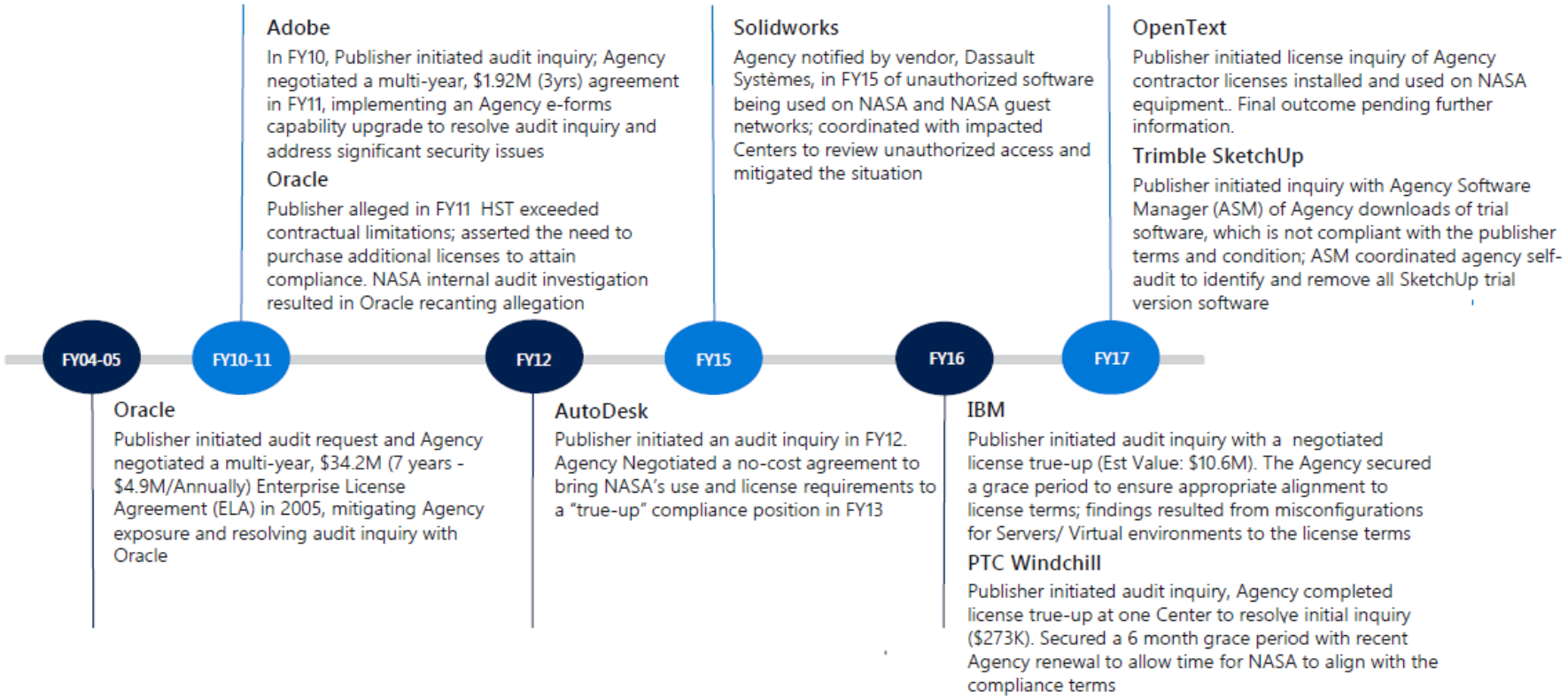
Avoid Hidden Dangers – Do Your Part!

Using unlicensed software could cost NASA MILLIONS and/or introduce security vulnerabilities into the NASA environment

It is critical to maintain software patch levels/versions and remove outdated/unsupported software from the environment to decrease the risks of cybersecurity threats to NASA's infrastructure

Software Publisher Audits

- Historical License Compliance



Agency Software Lifecycle Management Plan Vision, Goals, and Objectives

Vision

- Effectively manage software across the Agency and optimize software licensing and configurations

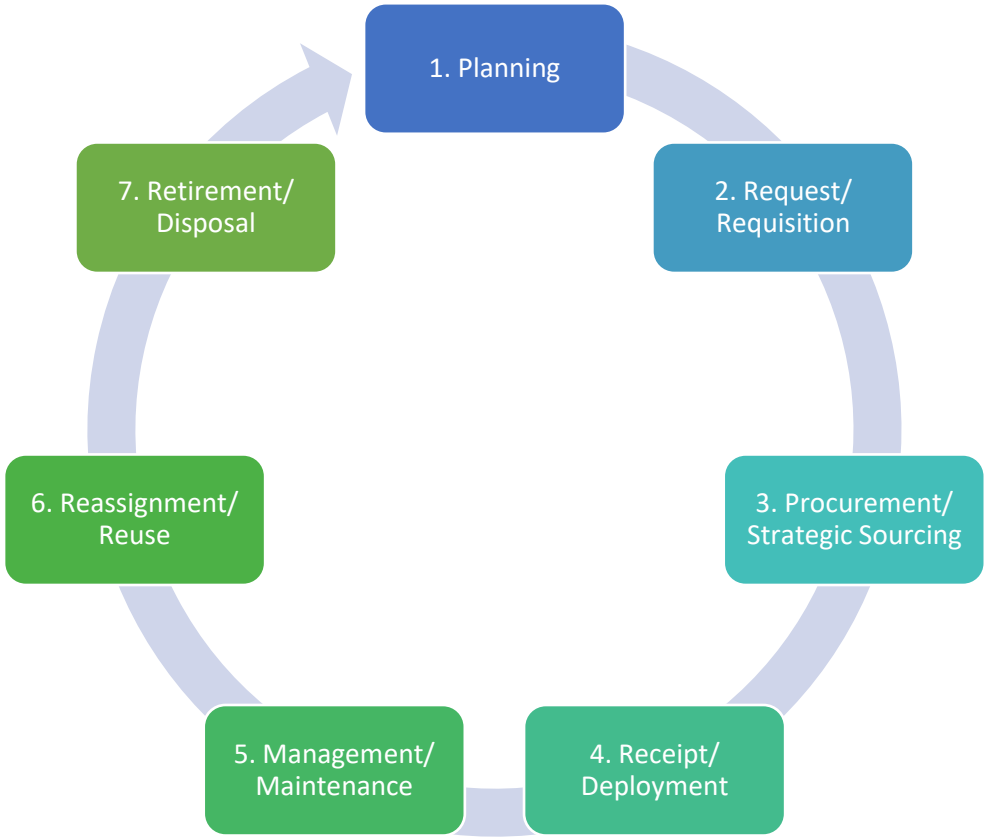
- Goals
  - Implement an effective Agency-wide Software Lifecycle Management process
  - Comply with the Megabyte Act of 2016 and OMB M-16-12 Category Management Policy 16-1: Improving the Acquisition and Management of Common Information Technology: Software Licensing
  - Support the achievement of the 2018-2021 IT Strategic Plan objectives

- Objectives
  - Centralized, standardized, streamlined lifecycle processes for managing software that delivers service to the customer in a timely manner and that is automated to the greatest extent possible
  - Greater insight into the software entering and existing in NASA's environment
  - Increased cost savings/cost avoidance through the improved management of NASA's software
  - Improved software related investment decisions
  - Reduced risk of security vulnerabilities related to software
  - Reduced risk of non-compliance license issues and costly audit findings

Approved Software List (CAP)

Software License Lifecycle

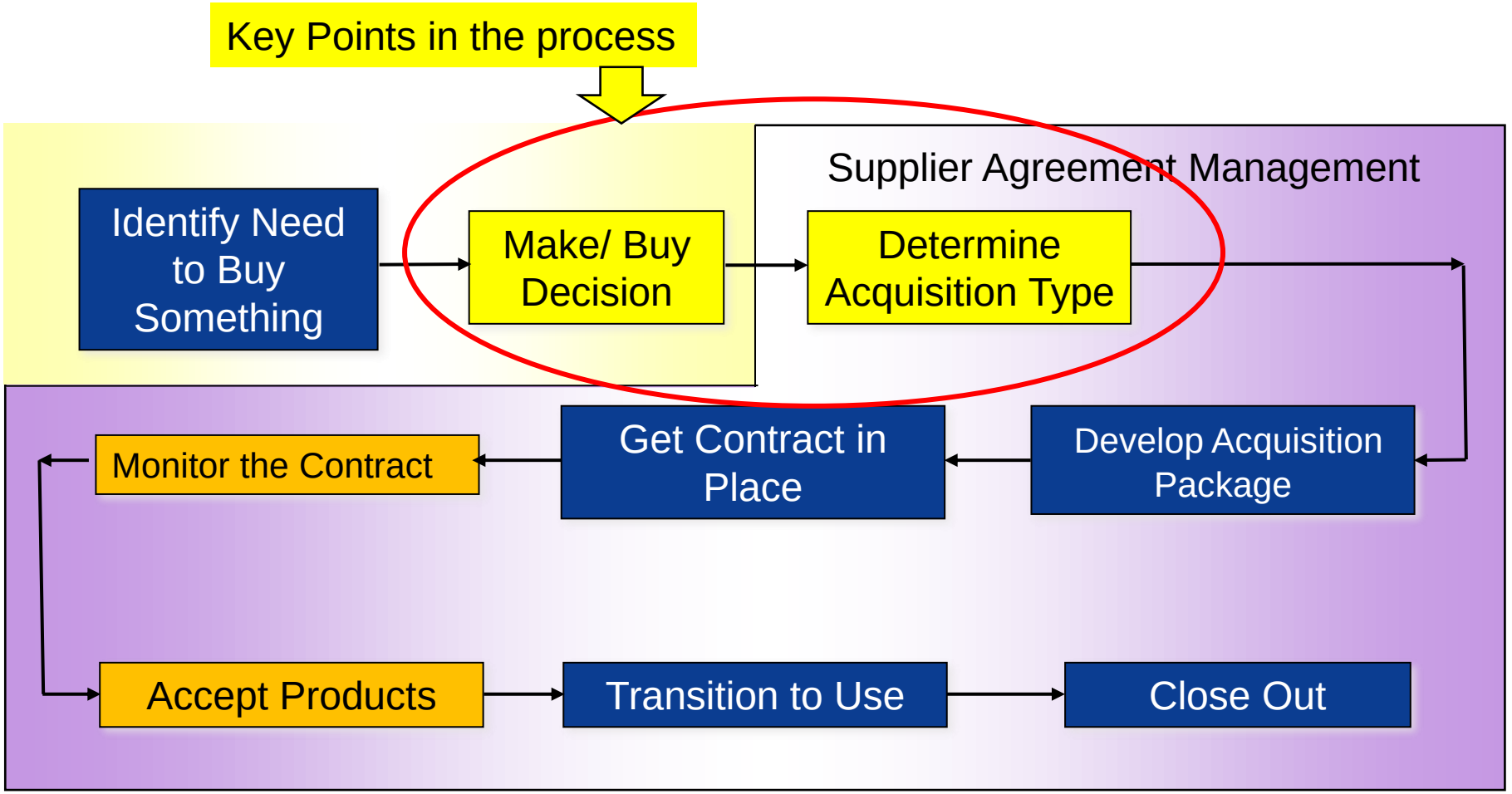
- The software license lifecycle at NASA consists of seven stages:
  1. Planning
  2. Request and Requisition
  3. Procurement and Strategic Sourcing
  4. Receipt and Deployment
  5. Management and Maintenance
  6. Reassignment and Reuse
  7. Retirement and Disposal
- Project Managers have critical responsibilities related to software licenses and subscriptions management.
  - Details are incorporated into NPR 7150.2D (section 2.1, 3.1) and the Applications Program (AP) Handbook*.



* https://sharepoint.msfc.nasa.gov/sites/ap/standards/SitePages/Home.aspx

NASA Software Acquisition Considerations

Road Map for an Acquisition



Beginning the Acquisition Planning

- First: What kind of item are we buying?
  - Commercial Off The Shelf (COTS) software?
  - COTS software with modifications?
  - Hardware, software tools or equipment?
  - Custom software?
  - Services from contractors to work with you on your teams?
- Begin working with procurement and management to determine options for acquisition
  - Possibilities might be:
  - Direct purchase (purchase order or credit card purchase)
  - Existing contract (using task order in place)
  - New contract (nothing exists to help your acquisition)
  - Part of larger (spacecraft) contract

NPR 7150 Applies to All Software Acquisitions

- NPR 7150 applies to software development, maintenance, retirement, operations, management, acquisition, and assurance activities.
- The requirements of NPR 7150 cover all software created, acquired, or maintained by or for NASA and apply to all of the Agency's investment areas containing software systems and subsystems.
- Put NPR 7150 on contracts, NASA project is still responsible either way

What Are Technical and Software Data Rights?

- The terms “Intellectual Property (IP) rights” or “data rights” refers to the government’s license rights in data.
- IP Rights are sometimes referred to as Rights in Technical Data and/or Computer Software
- As a general rule under government contracts, the contractor/developer is allowed to retain ownership of the technical data and computer software it developed.
- The government receives only a license to use that technical data and computer software.
- The scope of the license depends on the needs of the agency, source of funding for development, and the negotiations between the parties.
- Can apply to source code, executable code, documentation, test scripts, tools (including the software development and build environment)

What Are Technical and Software Data Rights?

- Determines who has the right to:
  - Use
  - Modify
  - Disclose
  - Distribute

- Determines how the government can use the technical data and software produced in an acquisition
- Influences the ability of the government to economically sustain systems
- Can influence the ability to interface to other systems
- Some commercial licenses are not in compliance with the Federal Acquisition Regulations (FAR)

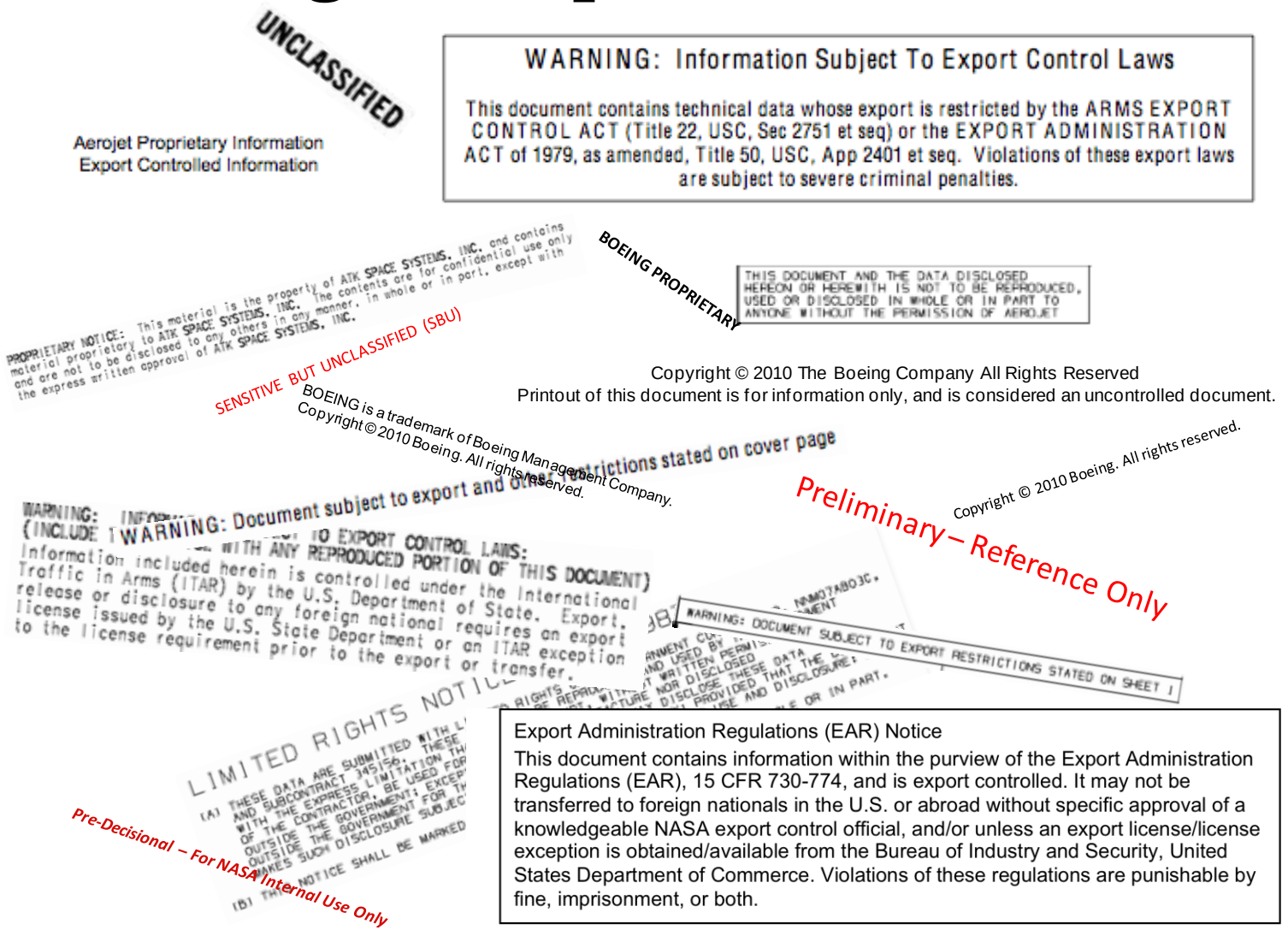
- Data rights (or lack of data rights) can have long term impacts
  - Use of the data the program office receives on the current program
  - Data that is provided to interfacing programs (especially in complex systems efforts)
  - Long term maintenance/sustainment of the current system
- It is always more expensive to try to negotiate data rights after the contract is let
  - It is vitally important to think about what data rights are needed well before the RFP is being prepared
- If data rights are important, ensure they are part of the evaluation criteria in a source selection

Data Rights Questions

- When writing an RFP:
  - What software data rights might you need?
  - What software/ software data might require additional data rights?
  - Who will need the software related data? And what data will they need?
  - What is the risk involved in not getting the rights to software data you need?
  - Will software data rights be used as an evaluation criteria?
  - How will user licenses be handled?
  - For commercial software
    - Are data rights provided to the public under the commercial license acceptable?
    - Are the commercial licenses in accordance with federal law?
    - Does the vendor have long term stability?
    - Is escrow a possibility? (Note – this does not totally solve most software rights issues)
  - For non-commercial software
    - Do we have a way to clearly identify what was developed with private or mixed funds?
    - Are the standard rights acceptable (unlimited, government purpose, restricted) or do I need specially negotiated rights?
    - Will we have rights to subcontractor provided software?

Data rights is a complex area – be sure to involve an IP Attorney and Contractor Officer as soon as possible if you anticipate complex data right needs

Marking Examples



CONTROLLED UNCLASSIFIED INFORMATION

https://www.archives.gov/cui

NPR 2810.7 **Controlled Unclassified Information**

Data Rights Questions

- Other Things to Consider
  - Ensure any license terms for COTS products do not conflict with any FAR provisions
    - Many of them do – so you need to check
  - If applicable, ensure FPGA code is included in the software data rights purposes
  - Provide adequate training on data rights for those who will evaluating them during the source selection
  - Consider both technical data rights and software data rights as needed
  - Continue to think about data rights throughout the program execution!

Electronic Access Requirements

- All software products acquired for NASA projects are to be made available in electronic format so they can be delivered accurately and used efficiently as part of the project. The electronic availability of the software work products, and associated process information, facilitates post delivery testing that is necessary for assessing as-built work product quality, and for the porting of products to the appropriate hosts. Electronic access to software projects reduces NASA's project costs.

- This access also accommodates the longer-term needs for performing maintenance, including defect repairs and software component augmentations, assessing operation or system errors, addressing hardware and software workarounds, and allowing for the potential reuse of the software on future NASA projects.

- Electronic access is needed during all phases of the software development life cycle. This enables software supplier activities to be monitored to ensure the software work products are being developed efficiently and that the end products that are called for in the project and software requirements are actually produced.

What Needs To Be Accessible?

- Software, executable and source code
- Models and simulations
- Programmable Logic Device logic and software
- Trade study data, including software tools used to help formulate analysis of alternative results if any scenarios need to be re-run later
- Prototype software, including prototype architectures/designs
- Data definitions and data sets
- Software ground products
- Software build products
- Build tools
- Software documentation, including data presented during any early design reviews
- Metric data
- Software cost data and parameters
- Software database(s)
- Software development environment
- Software Test Scripts and the results of software testing
- Results of software static analysis activities
- Bi-directional traceability for the software products
- Software analyses and compliance data

Summary

- Plan acquisition activities and identify potential suppliers
- Determine acquisition type and prepare acquisition documents
- Select suppliers and establish agreements (document all terms and conditions to be met)
- Execute the agreement
- Review supplier adherence to selected processes
- Report status to higher management
- Accept and transition the product

Class Plan

Software's Role and Importance in NASA Missions

NASA Software Engineering & Assurance Policies, Requirements and Resources

Software Planning Requirements and Considerations

- Software Documentation
- Software Costing
- Software Processes
- Software Assurance
- Software Safety-Critical
- Software IV&V
- Software Classifications
- Software Reuse and Internal Sharing
- Software Cybersecurity
- Software Lifecycles and Reviews

Software Life-cycle Requirements

- Software Requirements
- Software Architecture
- Software Design
- Software Coding
- Software Testing
- Software Maintenance

Software Development Supporting Requirements

- Software Configuration Management
- Software Risks
- Software Peer Reviews
- Software Measurements
- Software Defect Management
- Software Bi-Directional Traceability
- Software License Management
- Software Acquisition

Why do we do these things? Software Failures

Software Related Failures

“The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”

Edsger Dijkstra, 1972

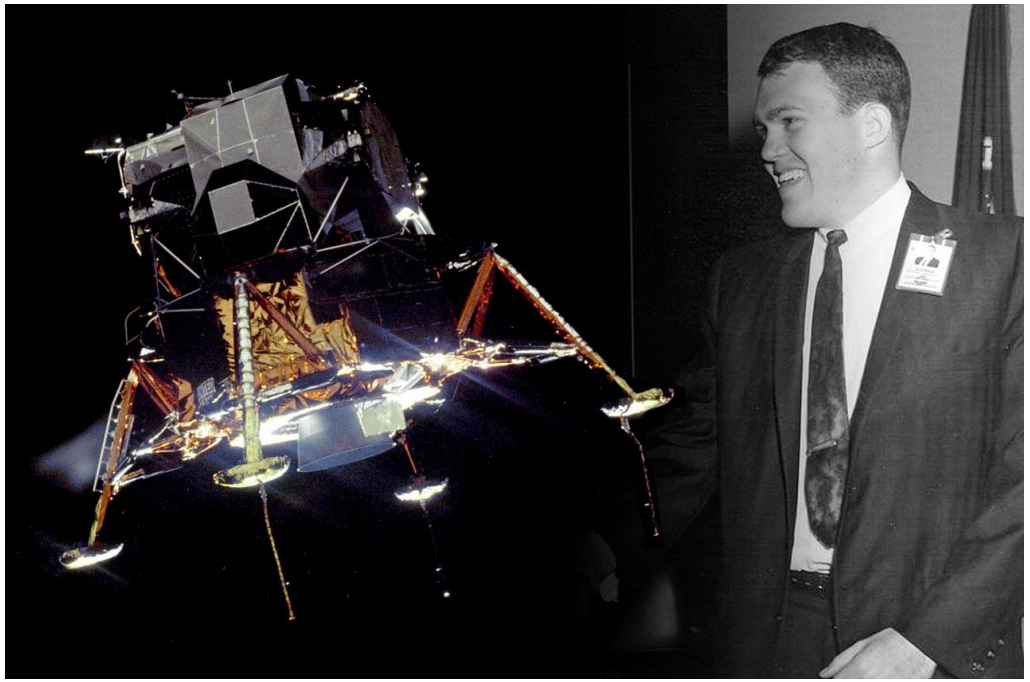
Why Software Projects Fail?

1. Unrealistic or unarticulated project goals
2. Inaccurate estimates of needed resources
3. Badly defined system requirements
4. Poor reporting of the project's status
5. Unmanaged risks
6. Poor communication: clients, developers, & users
7. Use of immature technology
8. Inability to handle the project's complexity
9. Sloppy development practices
10. Poor project management
11. Stakeholder politics
12. Commercial pressures

Others?

Why is software special?

- Software is invisible, intangible, abstract
- Software alone is useless - its purpose is to configure some hardware to do something
- Software doesn't have to obey the laws of physics
- Software is more complex for its "size" than other designed artifacts
- Software does not wear out
  - statistical reliability measures don't generally apply to software
- Software can be replicated perfectly
- Software is designed, not manufactured
  - Software can be re-designed after deployment



“Software is the easiest to change .... but in change, it is the easiest to compromise.”

The "Bug" Heard 'Round the World by John R. “Jack” Garman October 1981

Subsystem Failure Study Data

Table 1. Worldwide Subsystem Failures by Decade [4]

Subsystem	1980s	1990s	2000s
Propulsion	42%	38%	54%
Guidance and navigation	6%	16%	4%
Electrical	6%	8%	8%
Operational ordnance	2%	8%	0%
Structures	4%	6%	0%
Software and computing	0%	8%	21%
Pneumatics and hydraulics	4%	2%	0%
All other subsystems	0%	0%	0%
Unknown	37%	16%	13%

In addition to these specific failures, recent analyses of launch vehicle failure trends have shown that software and computing systems have become a much more frequent cause of failures recently than has occurred in the past.

Increase percentage contributed to software and computing

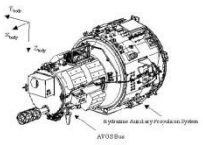
"Analysis of Launch Vehicle Failure Trends," Futron Corporation, August 7, 2006.

DEVELOPING SAFETY-CRITICAL SOFTWARE REQUIREMENTS FOR COMMERCIAL REUSABLE LAUNCH VEHICLES
Daniel P. Murray(1) and Terry L. Hardy(2)
(1)Federal Aviation Administration, Office of Commercial Space Transportation, 800 Independence Avenue, S.W., Room 331, Washington, DC, 20591, USA, Daniel.Murray@faa.gov
(2)National Aeronautics and Space Administration Goddard Space Flight Center, Mail Code 302, Greenbelt, MD 20771, USA, Terry.L.Hardy@nasa.gov

Software isn't any more fail-proof than hardware is… (it can and does break occasionally)

ISWE

	Coding (often statically detectable)	Design (Algorithmic)	Memory Use (corruption, heap memory, etc.)	Thread Use (race conditions; synchronization)	Code ReUse (not rechecking assumptions)	Fault Protection (over-reliance on reboot/reset)
60s	1962 Mariner 1 ("missing hyphen") 1963 Mercury (period instead of comma)		1968 Apollo 8 (memory corruption)	1969 Apollo 11 (1st moon landing)		
70s		1977- Voyager (navigation errors)				1971 Eole 1 (command confusion)
80s			1982 Viking 1 (memory corruption loss of contact)	1981 Shuttle (1st launch)	1988 Phobos (command confusion)	
90s		1993 Clementine (uncontrolled thruster firing)		1997 Pathfinder (priority inversion)	1996 Ariane5 (assumptions not verified) 1999 MCO (units adaptation omitted)	1996 Ariane5 (dual string; but same sw)
00s	2000 MPL (failure to reset variable)	2006 DART (navigation errors)	2004 MER (memory mngnt error) 2006 Jan MRO (memory corruption)	2006 Feb MRO (race condition)	2007 Dawn (code reuse)	2004 MER (uncontrolled reboot) 2006 DART (no backup controls) 2006 MGS (misdiagnosed fault) 2009 Dawn (fault protection)

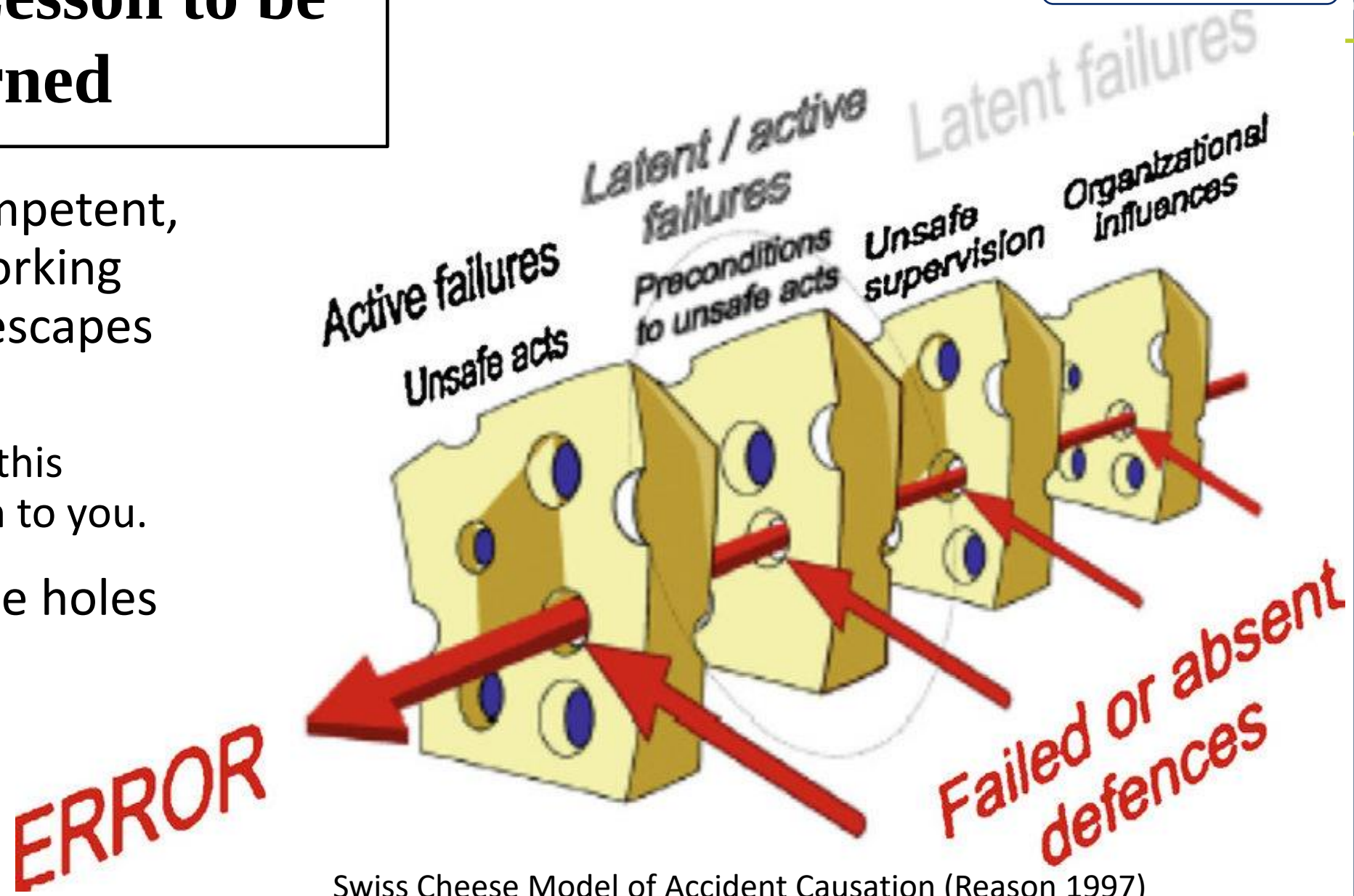


LARS
NASA/JPL Laboratory for Reliable Software

Detailed look at some of the Software Related Failures
Examples

The Main Lesson to be Learned

- Even with a competent, trained, hardworking team, process escapes can occur.
  - **DO NOT** think this cannot happen to you.
- "Sometimes, the holes line up."



Swiss Cheese Model of Accident Causation (Reason 1997)

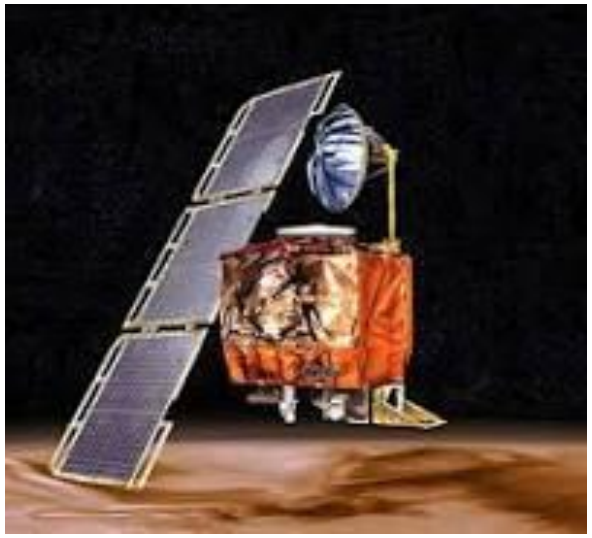
Intelsat 6

- Intelsat 6, a $157 million spacecraft , was stranded in a useless orbit March 14 , 1990 by a malfunction in its Titan 3 booster.
- Martin Marietta has traced the failure to a design error in the wiring associated with the separation electronics on its Commercial Titan
- When the core vehicle of the Titan's second stage shut down after a normal launch from a propulsion point of view, the vehicle's computer sent a spacecraft separation command. But the mismatch between the software and the wiring resulted in a signal being sent to the wrong wiring position, and the satellite stayed locked atop the booster.

- **The hardware engineers were supposed to go through a formal engineering change procedure to communicate any hardware changes to software engineers.**
- "The hardware guys thought they had communicated that change to the software side of the house," a Martin Marietta official said. But the communication breakdown occurred because an established change procedure was not used, the official said.
- The same communications breakdown was caught and fixed before the next Titan launch.
- STS-49 made repairs in space in time for the Intelsat 6 to participate in the broadcast of the 1992 Barcelona Olympics

NASA Mars Climate Orbiter

- *Incident Date: 9/23/1999 Price Tag: $125 million*
- *WASHINGTON (AP) -- For nine months, the Mars Climate Orbiter was speeding through space and speaking to NASA in metric. But the engineers on the ground were replying in non-metric English.*
- *It was a mathematical mismatch that was not caught until after the $125-million spacecraft, a key part of NASA's Mars exploration program, was sent crashing too low and too fast into the Martian atmosphere. The craft has not been heard from since.*
- *Noel Henners of Lockheed Martin Astronautics, the prime contractor for the Mars craft, said at a news conference it was up to his company's engineers to assure the metric systems used in one computer program were compatible with the English system used in another program. The simple conversion check was not done, he said*



Root Cause Analysis Case Study: Mars Climate Orbiter

http://youtu.be/UV3dNiR13CQ

The Mars Program Independent Assessment Team (MPIAT)

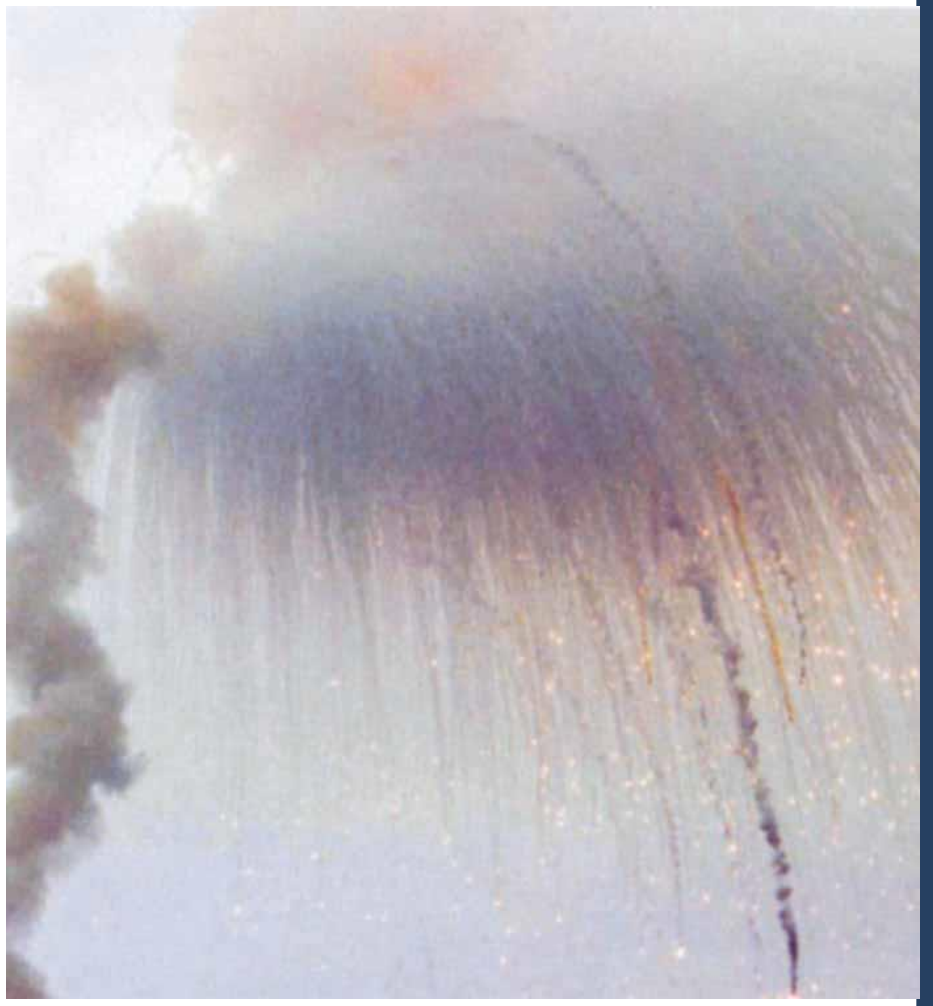
The MPIAT report found common characteristics among both successful and unsuccessful missions:

- **Experienced project management or mentoring is essential.**
- **Project management must be responsible and accountable for all aspects of mission success.**
- **Unique constraints of deep space missions demand adequate margins.**
- **Appropriate application of institutional expertise is critical for mission success.**
- **A thorough test and verification program is essential for mission success.**
- **Effective risk identification and management are critical to assure successful missions.**
- **Institutional management must be accountable for policies and procedures that assure a high level of success.**
- **Institutional management must assure project implementation consistent with required policies and procedures.**
- **Telemetry coverage of critical events is necessary for analysis and ability to incorporate information in follow-on projects.**
- **If not ready, do not launch.**

Ariane 5 Explosion

- *Incident Date: 9/1997 Price Tag: $500 million*
- *Ironic Factor: ****
- *(By James Gleick) It took the European Space Agency 10 years and $7 billion to produce Ariane 5, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.*
- *All it took to explode that rocket less than a minute into its maiden voyage last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a small computer program trying to stuff a 64-bit number into a 16-bit space.*
- **The number was larger than 32,767, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.**
- *This shutdown occurred 36.7 seconds after launch, when the guidance system's own computer tried to convert one piece of data -- the sideways velocity of the rocket -- from a 64-bit format to a 16-bit format.*
- *The number was too big, and an overflow error resulted. When the guidance system shut down, it passed control to an identical, redundant unit, which was there to provide backup in case of just such a failure.*
- *But the second unit had failed in the identical manner a few milliseconds before. And why not? It was running the same software.*

http://youtu.be/kYUrqdUyEpI



Ariane 5 Accident

- Why did this failure occur?
  - Why was Platform Alignment still active after launch?
  - SRI Software reused from Ariane-4
  - 40 sec delay introduced in case of a hold between -9s and -5s
- Why was there no exception handler?
  - An attempt to reduce processor workload to below 80%
  - Analysis for Ariane-4 indicated the overflow not physically possible
- Why wasn't the design modified for Ariane-5?
  - Not considered wise to change software that worked well on Ariane-4
- Why did the SRIs shut down in response?
  - Assumed faults caused by random hardware errors, hence should switch to backup
- Why was the error not caught in unit testing?
  - No trajectory data for Ariane-5 was provided in the requirements for SRIs
- Why was the error not caught in integration testing?
  - Full integration testing considered too difficult/expensive
  - SRIs were considered to be fully certified
  - Integration testing used simulations of the SRIs
- Why was the error not caught by inspection?
  - The implementation assumptions weren't documented

Software redundancy doesn't always work
Software reuse is risky

Mariner 1 Failure - Homework

$$\bar{\dot{R}}_n$$

Correct

$$\dot{R}_n$$

Wrong

Youtube Video:

"How a Tiny Mistake Destroyed America's First Interplanetary Space Probe"
Scott Manly

Titan IV B Centaur

- **Objective**: Titan IV B launch vehicle was equipped with a Centaur upper stage intended to deliver a Milstar satellite into geosynchronous orbit

- **Problem**:
  - After the Centaur separated from the Titan IV B, the vehicle began to experience anomalous rolls
    - The reaction control system eventually stabilized the vehicle during the transfer orbit coast phase but used 85% of its hydrazine fuel in the process.
  - When the vehicle attempted its second burn, it became unstable again and continued into its third burn tumbling.

- **Failure Analysis:**
  - Failed software development, testing, and quality assurance was ultimately the cause of the failure.
  - During development of the Centaur computer software, a decimal point was misplaced while manually entering the roll rate filter constant in the Inertial Measurement System flight software file.
  - This error was detected pre-flight but was not properly recognized or understood.
  - Although it was not needed, the software had been kept in for "consistency"

- **Date**: 4/30/1999

Ref: https://ntrs.nasa.gov/citations/20170009844

DART Failure

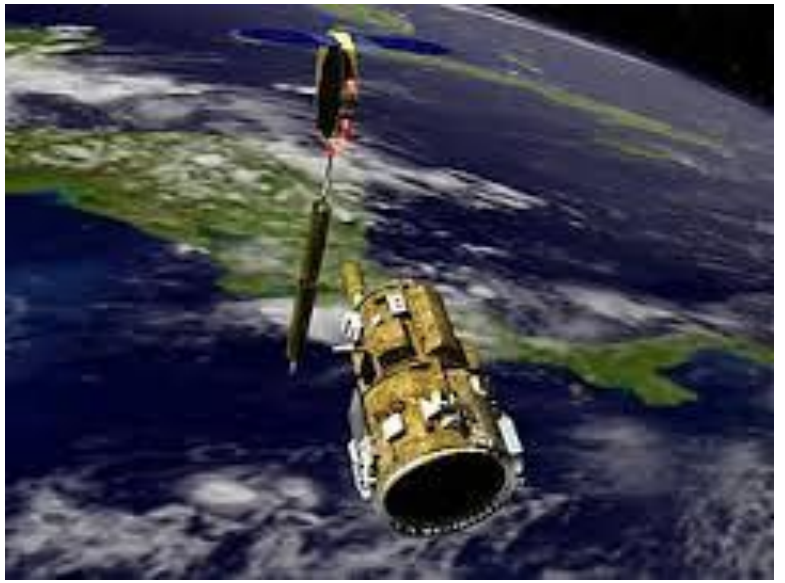
Demonstration of Autonomous Rendezvous Technology (*DART*)

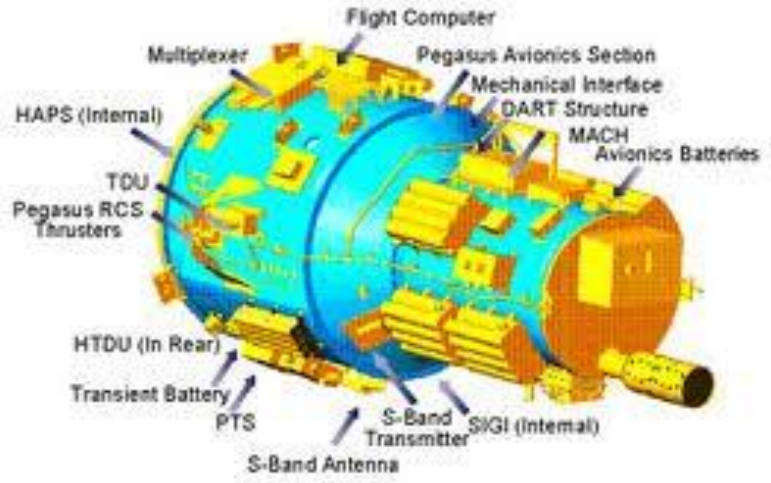
ISWE

What Happened:

When DART began its transfer out of the second staging orbit to begin proximity operations, ground operators observed that the **spacecraft was using significantly more fuel than expected for its maneuvers.** It became clear that the mission would likely end prematurely because of exhausted fuel reserves. Because DART had **no means to receive or execute uplinked commands,** the ground crew could not take any action to correct the situation.

During the series of maneuvers designed to evaluate AVGS performance, DART began to transition its navigational data source from the GPS to AVGS as planned. Initially, the AVGS supplied only information about MUBLCOM's azimuth (angular distance measured horizontally from the sensor boresight to MUBLCOM) and elevation relative to DART. However, as DART approached MUBLCOM, it overshot an important waypoint, or position in space, that would have triggered the final transition to full AVGS capability. Because it **missed this critical waypoint and the pre-programmed transition to full AVGS capability did not happen,** the AVGS never supplied DART's navigation system with accurate measurements of the range to MUBLCOM. Consequently, DART was able to steer towards MUBLCOM, but it was not able to accurately determine its distance to MUBLCOM. Although DART's collision avoidance system eventually activated 1 minute and 23 seconds before the collision, the inaccurate perception of its distance and speed in relation to MUBLCOM prevented DART from taking effective action to **avoid a collision.**





Multiple Root Causes and Recommendations on DART

- ***High Risk, Low Budget Nature of the Procurement***
  - **DART was selected by NASA as a high-risk, low-budget technology demonstration under a NASA Research Announcement (NRA). The government procured only the data, and set broad requirements. Most of the detailed design decisions about how to meet those requirements were left to the discretion of the contractor.**
- ***Training and Experience***
  - **a lack of training and experience led the design team to reject expert advice because of the perceived risks involved in implementing the recommendations.**
- ***Lessons Learned Analysis***
  - **Even though the DART team lacked training and experience, many of DART's inadequacies could have been addressed through review and proper application of mission experience and data (lessons learned) documented from previous NASA projects.**
- ***Guidance, Navigation and Control (GN&C) Software Development Process***
  - **The MIB determined that one of the root causes of the mishap was an inadequate GN&C software development process. Changes to the flight code and simulation models were often incorporated without adequate documentation.**
- ***Systems Engineering***
  - **inadequate, system-level integration process, which failed to reveal a number of design issues contributing to the mishap.**
- ***Schedule Pressure***
  - **Schedule pressure was identified as the cause for the inadequate testing of a late change to the navigation logic's gain setting.**

Multiple Root Causes and Recommendations on DART

- ***International Traffic in Arms Regulations (ITAR) Restrictions***
  - insufficient technical communication between the project and an international vendor due to perceived restrictions in export control regulations did not allow for adequate insight.
- ***Technical Surveillance/Insight***
  - the NASA DART insight team failed to identify issues that led to the mishap because of an inadequate assessment of project technical risk and insufficiently-defined areas of responsibility.
- ***Risk Posture Management***
  - the lack of adequate risk management contributed to a zero-fault tolerant design and inadequate testing that resulted in an insufficient collision avoidance system, among other things.
- ***Expert Utilization***
  - the DART team failed to fully use the resources of available subject matter experts.
- ***Contractor Review Processes***
  - internal checks and balances used by DART's prime contractor failed to uncover issues that led to the mishap, such as the undersized spherical envelope surrounding the AVGS range transition waypoint.
- ***Failure Modes and Effects Analysis (FMEA)***
  - analyses to identify possible hardware/software faults failed to consider a sufficient set of conditions that could lead to the mishap.

The Lewis Spacecraft

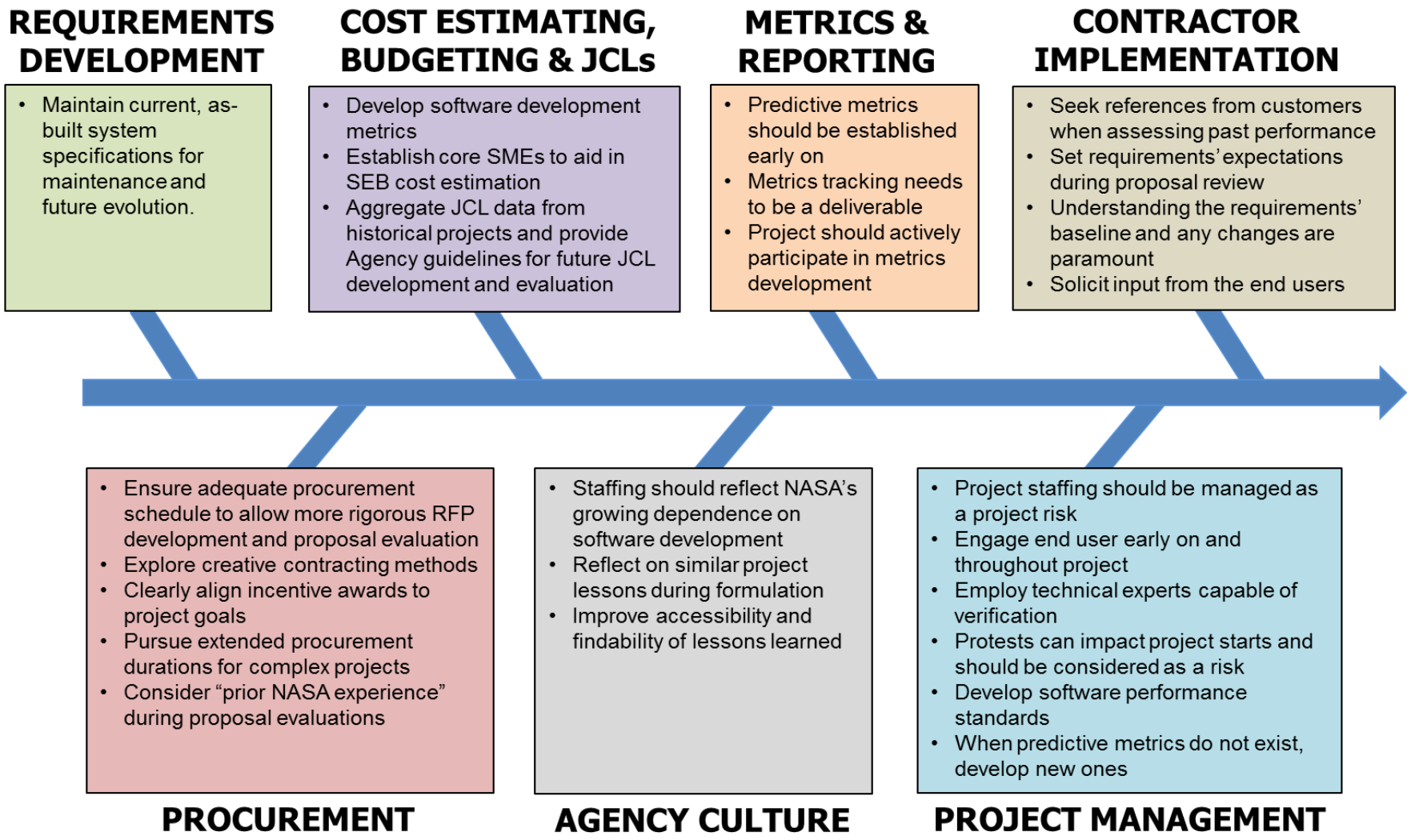
- The Lewis Spacecraft was procured by NASA via a 1994 contract with TRW, Inc., and launched on 23 August 1997. Contact with the spacecraft was subsequently lost on 26 August 1997. The spacecraft re-entered the atmosphere and was destroyed on 28 September 1997.

- The Lewis Spacecraft Mission Failure Investigation Board found that the loss of the Lewis spacecraft was the direct result of an implementation of a technically flawed Safe Mode in the Attitude Control System.

- This error was made fatal to the spacecraft by the reliance on that **untested** Safe Mode by the on orbit operations team and by the failure to adequately monitor spacecraft health and safety during the critical initial mission phase.

- Other causes cited included **requirement changes** without adequate resource adjustment, **cost and schedule pressures**, a Program Office move, inadequate ground station availability for initial operations, **frequent key personnel changes**, and **inadequate engineering discipline**.

Critical Lessons Overview



Software error doomed Japanese Hitomi spacecraft

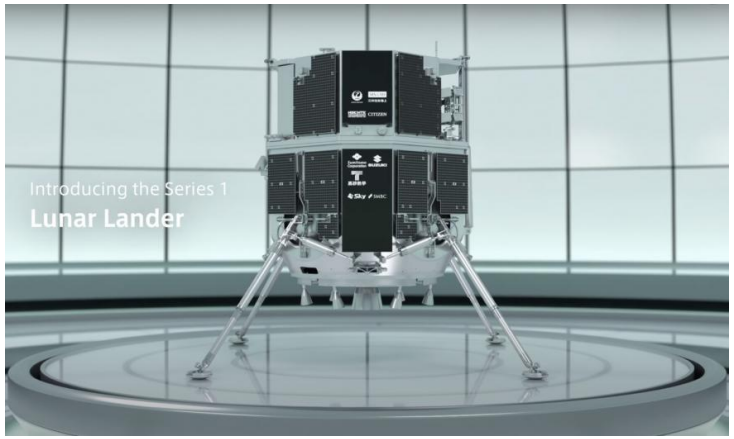
Japan's flagship astronomical satellite Hitomi, which launched successfully on 17 February, 2016 but tumbled out of control five weeks later, may have been doomed by a basic engineering error.

- The spacecraft automatically switched into a safe mode and, at about 4:10 a.m., fired thrusters to try to stop the rotation.
- But because the wrong command had been uploaded, the firing caused the spacecraft to accelerate further.
- (The improper command had been uploaded to the satellite weeks earlier **without proper testing**; JAXA says that it is investigating what happened.)

On 28 April, the Japan Aerospace Exploration Agency (JAXA) declared the satellite, on which it had spent ¥31 billion (US$286 million), lost.

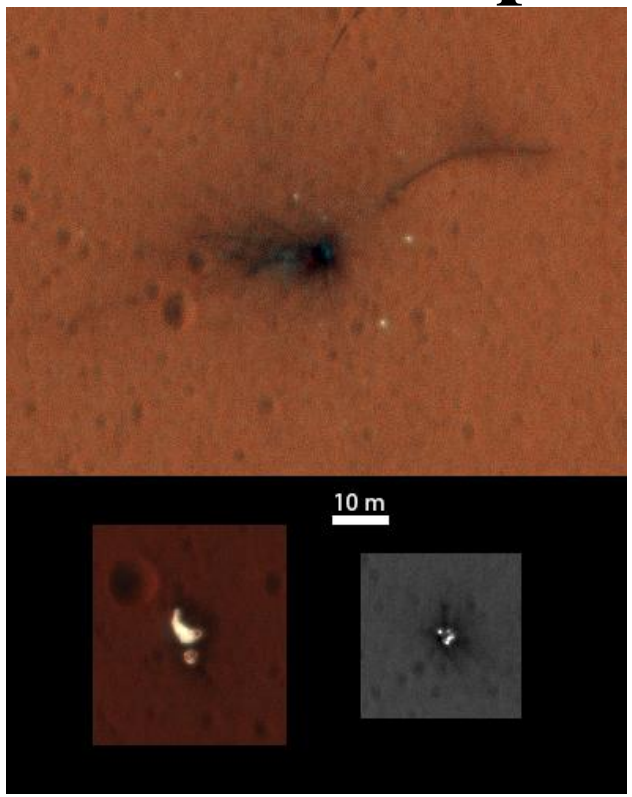
Japanese ispace company moon lander Hakuto-R crash

- Lander was launched and was attempting to land on moon
- Blame was placed on software issue
  - Lander passed over a lunar crater
    - Radar altimeter sensed sudden drop of 3 km
    - Software was programmed to identify this (sudden change) as a failure and disqualify the sensor
      - Data was correct though
    - Now Lander flying without measurement of ground, using estimation of what it thinks (using gyros/accelerometers)
    - Software “landed” on what it thought was surface, but never got feedback it touched down, so hovered until fuel ran out
    - Then dropped ~5km to surface
- Why was this not caught in testing?
  - Landing site was changed <u>after</u> all simulations were run and never done for the new landing site
- April 2023



https://www.youtube.com/watch?v=2JlUnOAiMm4

ESA's Schiaparelli Failure



3 November 2016New high-resolution images taken by a NASA orbiter show parts of the ExoMars Schiaparelli module and its landing site in color on the Red Planet.

October 2016

Schiaparelli was primarily meant to test European landing technologies, with science as a secondary objective. Recording the data during the descent has already achieved a lot of the mission's goals

Europe and Russia's ExoMars lander may have suffered **a computer glitch** during its descent to Mars last week, ultimately causing it to crash-land into the planet's surface, *Nature* reports. As the lander fell, the **mysterious software** bug may have caused the vehicle to think it was closer to the ground than it actually was, a lead researcher with the European Space Agency suggests. That may be why the whole landing sequence was thrown out of whack.

All this **seems to suggest a software error,** says Andrea Accomazzo, who is in charge of ESA's solar and planetary missions. Accomazzo thinks maybe Schiaparelli had a problem processing all the information it was getting from its sensors. This led the spacecraft to think it was at a lower altitude than it was during the fall, causing many of its landing operations to cut off early.

Additional Common Problems: Flight Software Lessons

- An appropriate high fidelity Flight Software test bed is non-negotiable for each flight Computer Processor Unit (CPU).
- Strong Flight Software Requirements Development, Review and Control are mission critical
- Flight Software needs to be engineered across all onboard systems
- Flight Software requires specialized code that shouldn't be underestimated in ability to impact mission viability
- Project-level advocacy of flight software lead role across all subsystems is essential
- Flight Software Branch should explain and recommend a risk mitigating end-to-end Flight Software development process to each project.
- Flight Software Organizations must voice concerns
- Closely question reuse assumptions when developing common software
- Use a defined evaluation process when selecting Off The Shelf software components
- Performance based contracting
- Carefully define deliverables, process, evaluation criteria and tracking metrics when writing Request For Procedures and contracts.

Summary

- **Most Failures have multiple root causes**
- **Lessons learned from space vehicle failures have shown the importance of developing valid software requirements and verifying that those requirements are effective and have been implemented properly.**
- **Software and computing systems are critical to safe launch vehicle operations and spacecraft.**

Questions for Discussion

- How does your organization acquire the evidence to understand that your system software will do what it is supposed to do, under adverse conditions, and won't do what it is not supposed to do (guard against emergent behaviors)?
- How does your organization track configuration management and evaluate change from a systems perspective?
- If your primary unit failed due to software errors, will it cause the same failure in your backup? What is your proper level of redundancy?
- Has the risk level of your project decreased, and your software testing plan increased to drive down risk?
- Do you have contingency plans for on-orbit anomalies? What anomalies have been tested for?
- How does your organization verify reused or modified code?

From:
Critical Software: Good Design Built Right
SYSTEM FAILURE CASE STUDIES
January 2012 ***Volume 6 Issue 2***

NASA Software Class Summary

Class Plan

Software's Role and Importance in NASA Missions

NASA Software Engineering & Assurance Policies, Requirements and Resources

Software Planning Requirements and Considerations

- Software Documentation
- Software Costing
- Software Processes
- Software Assurance
- Software Safety-Critical
- Software IV&V
- Software Classifications
- Software Reuse and Internal Sharing
- Software Cybersecurity
- Software Lifecycles and Reviews

Software Life-cycle Requirements

- Software Requirements
- Software Architecture
- Software Design
- Software Coding
- Software Testing
- Software Maintenance

Software Development Supporting Requirements

- Software Configuration Management
- Software Risks
- Software Peer Reviews
- Software Measurements
- Software Defect Management
- Software Bi-Directional Traceability
- Software License Management
- Software Acquisition

Why do we do these things? Software Failures

Course High Level Objectives

- To provide an introduction to NASA software engineering skills
- To help non software engineers, system engineers and project managers understand the software development processes and considerations
- To help NASA engineers make better software related decisions by knowing where to get information and guidance

Class Summary

Focus the standard on known software issues:

- Software Requirements
- Software Code
- Process assessments
- Analyses (including Hazard Analyses)
- Data needed for reporting

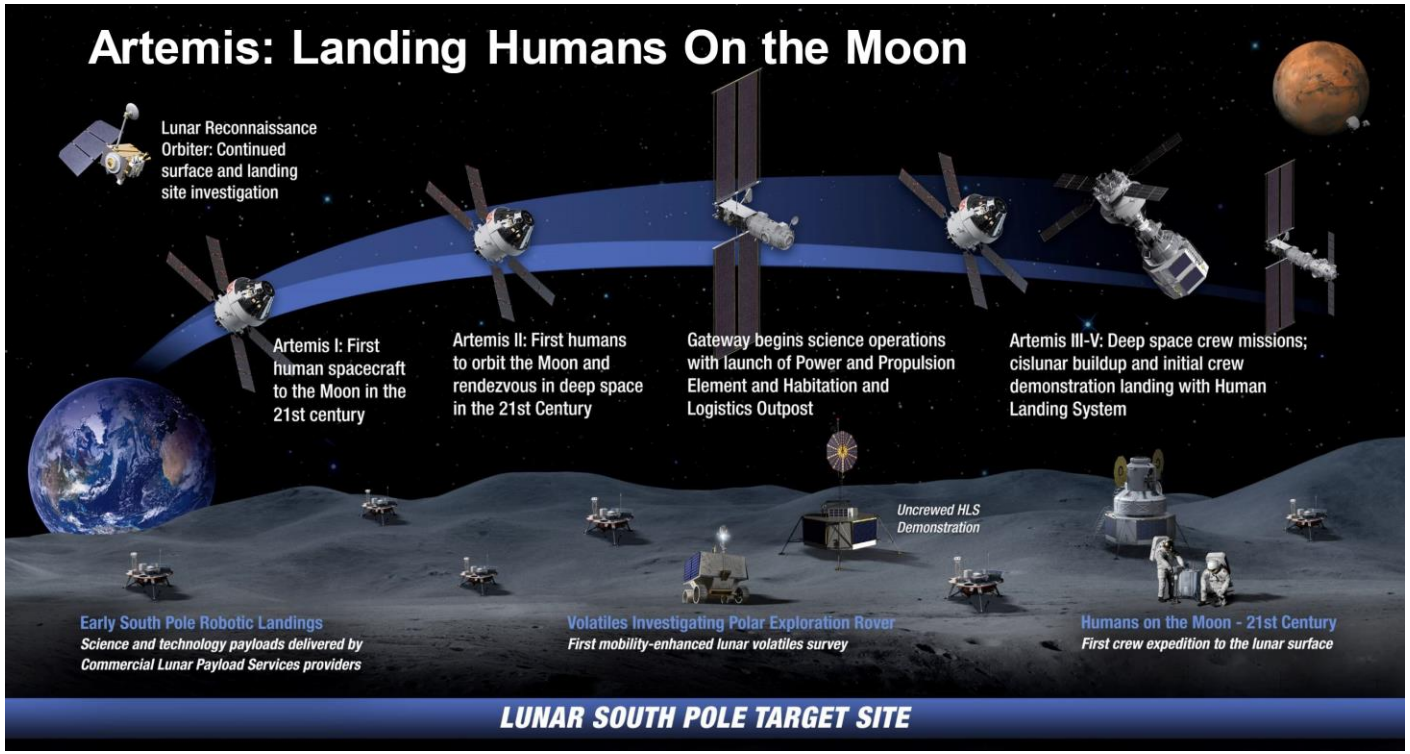
Summary

- The NPR provides a minimal set of requirements for software acquisition, development, maintenance, retirement, operations, and management
- The updated directive supports NASA programs and projects in accomplishing their planned goals (e.g., mission success, safety, schedule, and budget) while satisfying their specified requirements.
- The directive provides increased flexibility and tailoring options for software requirements for projects based on risk

Look at the software requirements and determine what you need to do for your project

Software's Role and Importance on NASA Missions

- **Software engineering and software assurance is a core capability** and a key enabling technology for NASA's missions and supporting infrastructure.
- All NASA **missions** have software involvement
- **NASA's success in increasingly dependent on software functions and capabilities.**
- NASA must become more efficient and effective in developing and validating quality software.



Future State

NASA missions will have more software, more complexity and more autonomous operations

We will need to invest in the software workforce to be able to support the NASA missions

Questions

Additional information can be found at

https://nen.nasa.gov/web/software

https://swehb.nasa.gov/

https://sma.nasa.gov/sma-disciplines/software-assurance

https://nsc.nasa.gov/SMAToolbox/

https://software.nasa.gov

https://open.nasa.gov

https://developer.nasa.gov

Acronyms

NPR 7150.2 Appendix B

- Select Acronyms:
  - CDR – Critical Design Review
  - EGS – Exploration Ground Systems
  - FAR – Federal Acquisition Regulations
  - FPGA – Field Programable Gate Array
  - FRR – Flight Readiness Review
  - FSW – Flight Software
  - FTE – Full Time Employee
  - I/O – Input/Output
  - ISWE – Introduction to Software Engineering
  - MCR – Mission Concept Review
  - MDR – Mission Definition Review
  - NDA – Non-disclosure agreement
  - NEN – Nasa Engineering Network
  - OPM – Office of Personnel Management
  - ORR – Operational Readiness Review
  - PDR – Preliminary Definition Review
  - PRR – Production Readiness Review
  - RFP – Request for Proposal
  - SAR – System Acceptance Review
  - SDR – System Definition Review
  - SIR – System Integration Review
  - SLOC – Source Lines of Code
  - SLS – Space Launch System
  - SQL - Structured Query Language
  - SRR – System Readiness Review
  - SWE – Software Engineering
  - SwRR – Software Readiness Review
  - TDT – Technical Discipline Team
  - TRR – Test Readiness Review
  - UML - Unified Modeling Language™
  - WYE – Work Year Equivalent