SOFTWARE MANAGEMENT & VERIFICATION

Dena Gruca

Overview

- NASA Guidelines/Procedures
- Typical Project Lifecycle
  - Beginning Phase
  - Preliminary Design Phase
  - Critical Design Phase
  - Software Development Phase
  - Software Test Phase
    - Regression Testing
  - Operations Phase
  - Project Closeout/Retirement Phsae

NASA GUIDELINES AND PROCEDURES

NASA Procedural Requirements (NPR)

- Agency level documents for systems engineering
  - NPR 7150.2B NASA Software Engineering Requirements
  - NPR 7123.1B NASA Systems Engineering Processes and Requirements
- Armstrong (Dryden) Center Procedural Requirements (DPR) for systems engineering
  - DPR 7150.2A
  - DPR 7123.1C

Requirement Tailoring

- NPRs are written under the assumption that every project is going to space
- AFRC projects rarely go to space; more aeronautics oriented
- How do we meet intent of NPRs?
  - Create Center-level documents tailored from the NPR
  - Create a trace matrix that shows how our center-level requirement meet the intent of the agency level requirements
  - Work with headquarters to get approval of center level documents
- But not every project at the center is the same…
  - Each project tailors the Center-level requirements during the beginning phases in their project documents (Systems Engineering Management Plan, Software Management Plan)
  - Create trace matrix to show how they meet intent of center level requirements
  - Work with center management to get approval of these tailored documents

TYPICAL PROJECT LIFECYCLE AT NASA ARMSTRONG

(from a Systems Engineer's perspective)

Beginning Phase: Project Definition

- When starting a project here at NASA, I like to ask the project team these questions:
  - What is the project?
    - What is the goal of the project?
    - Who will need to be involved?
    - What resources will you need?
  - How will you develop your software?
    - Or will you be the ones developing the software?
      - Contract out? Commercial-Over-The-Shelf (COTS)?
    - What platform will you use?
  - How will you manage/configure your software?
  - What safety measures will you take?

Beginning Phase: Project Definition

- Then, the project works to answer those questions by:
  - Agreeing on objectives, success criteria, and milestones
  - Creating top level requirements
    - What are the main features/functions of the system?
    - Are there any safety hazards/risks when using/testing the software?
    - Start considering top level software architecture
  - Considering how to test requirements
    - Are requirements definite/measureable?
      - Bad example: “The software shall provide output.”
        - How much output? What is the output format? How often should it provide output?
      - Good example: “The software shall provide output data at a rate of 25 Hz.”
  - Creating a guesstimated schedule for completing all top-level tasks
  - Agreeing on how to track changes, where to store changes (Configuration Management Plan)
  - Considering creating coding guidelines/standards

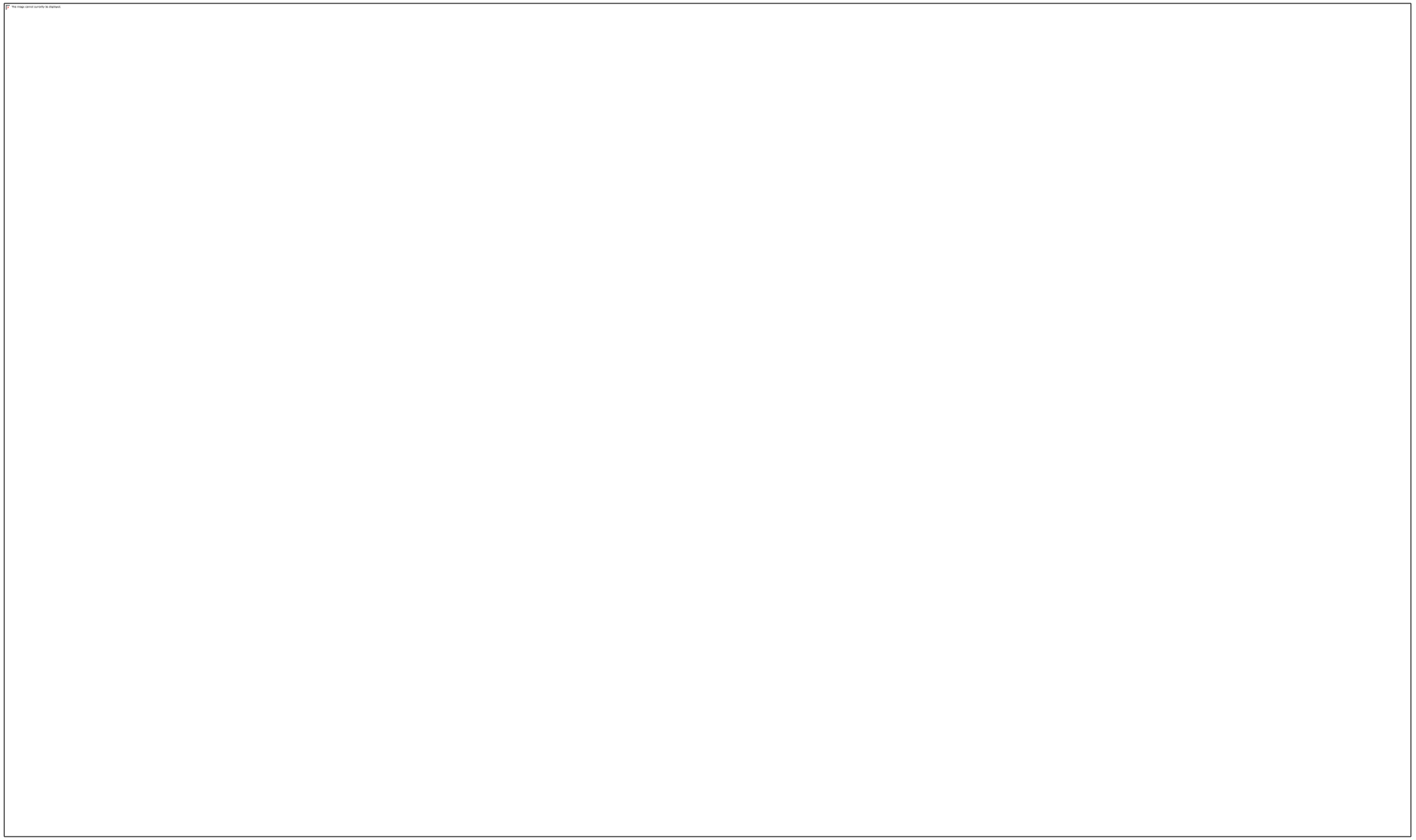
Preliminary Design/Top Level Design

- The project presents Project Definition Review to an independent team (2 peers, or a formal panel), gathers feedback, and stores documents in project folder
- Then, project software team works to answer these questions:
  - How will software interface with hardware?
  - What does the software architecture look like?
  - Knowing high level requirements, is it better to develop it, or buy it?
  - What software requirements are needed to meet higher level requirements? This drives development and/or purchasing criteria
    - What speed is data coming in? Going out?
    - What are the size of data packets?
    - What is communication route? Ethernet, Wireless, Bluetooth and/or is it stored internally (Hard drive, jump drive)?
    - How is it displayed? Or is it displayed? (Connected monitor, touchpad?)
    - What functions are needed?
    - What algorithms are needed?
    - What parameters are used? (Input/Output/Internally)

Side note on Requirements

- Requirements are the cornerstone of any project
- Tedious task, but otherwise…
  - How do you know what you want?
  - How do you know what to include/exclude in design?
  - How do you know what to develop/purchase?
  - What do you test against to know your product works the way you want it to?

Software Requirements



Preliminary Design/Top Level Design

- How the team answers those questions:
  - Create an Interface Control Document (ICD) to show how software will interface with hardware and/or external software
  - Create a configuration item list (subsystem list)
    - Use for creating development “sandboxes”, tracking changes, sorting through history of system development
    - One for each program, device, or subsystem (categories: "Display drivers”, “OS”, “App1”, “App2”, etc.)
  - Start writing lower level software requirements into a matrix/spreadsheet; this is the cornerstone for testing and verifying software
  - Start creating a test plan approach
    - Who is needed to run and evaluate the test? What will you test? Where will you test (in lab? on a bench or specific box?) How will you capture test results and test anomalies?
    - Should you be using scripts or a specific software test tool?
  - Create a hazard analysis
    - What happens if… the software fails or dies? the data is lost? someone pushes the wrong button? the software catches a virus?

Critical Design/Detailed Design

- Project presents Preliminary Design Review to independent team, gathers feedback, and stores documents in project folder
- Then the team works to complete these tasks (almost to the “real work”):
  - Baseline all requirement lists/documents
  - Baseline the system verification test plan
  - Create a data dictionary and/or interface documents
  - Start outlining the software design description documents(s)
  - Review preliminary hazards, any new ones?
  - Outline the subsystem/unit level test plans (who/what/where/how?), if needed
  - Agree upon coding guidelines for project

Software Development Phase

- Project presents Critical Design Review to independent team, gathers feedback, and stores documents in project folder
- Now the software team gets to start on the “real work”
  - Develop code
  - Generate software design document(s)
  - Generate draft user’s guides (if needed)
  - Generate draft load procedures (if needed)
  - Update data dictionaries and interface documents to reflect code development
  - Create draft verification test procedures (can have multiple procedures)
    - Test against requirements (each requirements should be a test point)
    - Test for anomalies, outside boundaries
    - Test data speed, data loss
  - Create a trace matrix to verify each requirement is tied to a test or test point
  - Perform informal integration and testing of subsystems
  - Update software hazards
- Software team presents code walkthroughs
- Project prepares to present Test Readiness Review

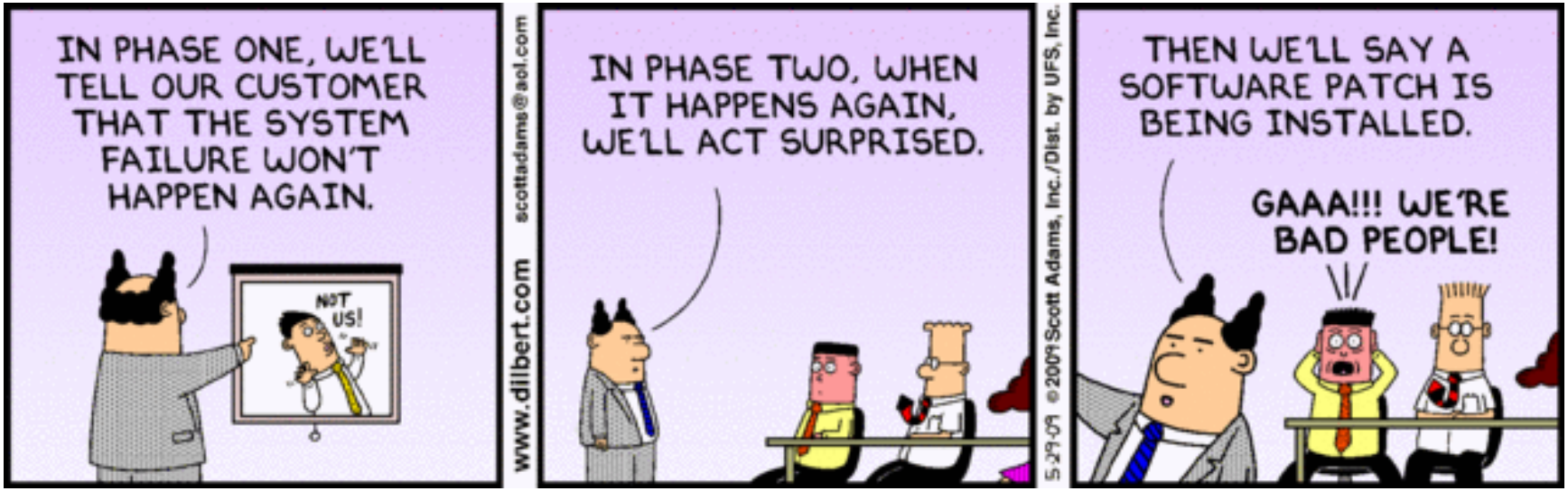
Software Test Phase

- Project presents Test Readiness Review to independent team, gathers feedback, and stores documents in project folder
- Next, the software team and systems team proves the design/code “works”:
  - Place software under configuration control and baseline software (give version number)
  - Approve Verification Test Procedures
  - Finalize ICDs and data dictionaries
  - Create a version description document
    - What’s included in this version? What’s new? What changed? File size/checksum?
  - If receiving software (COTS), review documentation package
  - Load software, if needed
  - Perform formal subsystem and system level tests as documented in the Verification Test Plan and Test Procedures, as outlined in the Test Readiness Review
  - Gather data and generate test results
  - Document discrepancies and redlines/changes to Verification Test Procedure
  - Create System Test Reports
- Project prepares to perform Formal Data Reviews

Regression Testing

- After/during initial Verification Testing, more testing may be needed
  - The team documents discrepancies, redlines/changes to procedures
  - Discuss changes needed with the project
    - What options are available? Change code or change procedure?
    - Does this change still meet all requirements? Create new ones?
    - How long will it take to make changes?
    - Will changes affect other subsystems? Other parts of the code?
    - Does the change create new hazards? Modify or eliminate pre-existing hazards?
    - How much re-testing is needed? Multiple tests? One test? Part of a test?
  - Team makes agreed upon changes
    - Submit software for configuration (new version number)
    - Update Version description document
    - Update any other affected documentation
  - Present changes to independent team/board, similar to Test Readiness Review
  - Retest
    - Perform formal subsystem and system level tests as documented in the Verification Test Plan and Test Procedures, as outlined in the updated Test Readiness Review

Regression Testing



Operational Phase

- Close all discrepancy reports
- Brief Final Test Results
- Generate User's Guides (if needed)
- Generate Load Procedures (if needed)
- Compile documentation, data, and software into a deliverable package, if providing to customer
- Support operations as needed
- If changes are needed, follow steps in regression testing phase

Software Configuration

- Most projects have a systems engineer that doubles as a software manager
  - Creates management and configuration documents
  - Sets up central code repository structure, manages permissions and folders
  - Coordinates software team meetings and aware of development status, issues
  - Usually creates Test Plan for testing the software
  - Heavily involved in all phases, especially requirements, tracing requirements to test procedures, and assists in writing procedures
- In design phase, software team agrees upon a central repository. SVN, github, etc.
- In development phase, team uses the central repository to save project in a “sandbox” folder
  - General rule of thumb, check software in as often as possible (daily, weekly)
- In testing/retesting phase, submit team submits “release” versions/tags in separate branch/folder to software manager
- Software manager tracks software versions, version documentation, and test results

Side note on Software Configuration

- TRACK CHANGES!
  - Know what you are testing/operating
  - Easy to determine what changes affects testing/operations
  - Saves time and money for everyone; less troubleshooting, less retesting, better communication
  - Provides history of development
- TEST CHANGES!
  - “One little change” can affect a larger part of the system with unintended consequences (e.g., adding one more parameter generates buffer overflow)
- Same is true for documentation
  - Documentation can also be kept in repository, or other central location, such as shared folder
  - Documentation is tedious and “no one ever reads it”…until something happens and someone else has to carry on where you left off

Project Closeout (aka Project Retirement)

- Start double checking that all documentation is completed and in one central location
- Write reports
- Reduce folder permissions from read/write-access to read-only and ready documentation/code for archival
- In some cases, consider if code is viable for re-use and how to handle reusable code
- Have a end-of-project celebration

Thank you for your time

Questions?