Jet Propulsion Laboratory
California Institute of Technology

Heavyweight Quality, Agile Methods

Agility in Flight Mini-Workshop

DJ Byrne

Embedded Software Engineer, Flight Software Applications.

Jet Propulsion Laboratory, California Institute of Technology

2019-07-31

Quality Lives in Artifacts

Jet Propulsion Laboratory
California Institute of Technology

- Any final product consists of a set of artifacts
  - Primary
    - Executable image(s), Code base, User documentation
  - Secondary / supporting
    - Test results, Design descriptions, Reviews, Sign-offs
- Any work not captured in artifacts only matters *if* it improved some artifacts' quality
  - Prime example: email threads of technical discussions

Heavyweight Artifacts

- Heavyweight methods seek quality by creating an exhaustive set of artifacts
  - So nothing is overlooked
  - Can lead to low-value work
    - One size fits all, vs Tailoring
  - Is each artifact valuable to each project?
    - If not, why can you live without that information?
  - Example: waterfall methodology
  - Excellent reference: **NASA System Engineering Handbook** (SP-2016-6105)
    - https://www.nasa.gov/feature/release-of-revision-to-the-nasa-systems-engineering-handbook-sp-2016-6105-rev-2

Agility in Artifacts

Jet Propulsion Laboratory
California Institute of Technology

- Focus on what matters
- Does creation order really matter?
  - A stitch in time saves nine.
    - If you're going to do it eventually, when will it save other effort?
    - One good test may be worth 1,000 "shall" statements
- Form should follow function – choose a format that reflects the content's intended use
  - Text? Spreadsheet? Database? Script?
- Bake important things into tools
  - Documentation becomes lightweight pointers into the tools
- If it's quick to say, say it!

Keynote Example Sources

Jet Propulsion Laboratory
California Institute of Technology

- MSL (Mars Science Laboratory, a.k.a Curiosity)
  - 1) TDS (Terminal Descent Sensor ***Derived*** from Electra software-defined radio)
    - CMMI-3 certification scrutiny
  - 2) Radar Calibration File Generator (***New*** development)
- Mars 2020
  - 3) TDS (***Inherited as-is*** from my MSL-self)
  - 4) SECC (Second Chance, ***Inherited for update*** from MSL)
  - 5) MOXIE (Mars OXidation In-situ Experiment, ***Customization*** from IML (Instrument Management Library))
- All NASA software, Class B
  - per "NASA Procedural Requirements", NPR 7150.2B

Management / Development Plan

Jet Propulsion Laboratory
California Institute of Technology

Ex, TDS: "Your signature only means that you trust ***their*** signature, not that you read it."

Qualities

- Stakeholders
  - Who must say "**Yes**"?
  - Who can say "**No**"?
  - Subject Matter Experts?
- Schedule
  - Receivables
  - Deliverables, *chunked by release cycle*
- Budget
  - Rule of thumb: 10% on management itself
- CM (Configuration Management)
- Training / hiring: what skills you need
  - Languages, operating systems, tools...
- Acronyms, Glossary

Methods

- Heavyweight:
  - Comprehensive plan is the 1st deliverable, signed by a half-dozen people. Details processes
- Lighten up:
  - Stakeholders can agree to be advised of changes rather than consulted for each one.
  - Receivables list needs **what**, **when**, **from-whom**, **to-whom**, and **status**. Describe the minimum needed for clarity between f*rom-whom* and *to-whom*.
  - Schedule and budget establish that on **this-date** someone delivers **this-item.** Granularity to make progress evident.

Requirements

Ex, RCF: All stakeholders agreed, in writing, "Do what Elaine says."

Qualities

- What does success look like?
  - Functionality, yes, and...
  - Capacity ("for 20,000 users...")
  - Performace
  - Reliability, fault handling
  - Testability
- All stakeholders have the same idea
- Link development with test

Methods

- Heavyweight:
  - "Shall" statements
  - Requirements are complete and signed before development begins.
- Lighten up:
  - Delegate
  - Write the tests first, and require that each test passes. *Pro-tip: prioritization*
  - Go ahead and code it 1st, calling it "requirements exploration"
  - "Implement the flowchart on this whiteboard picture"
  - "The data-packet shall be this table"
  - Put the pictures and tables in CM!

Design Description

Ex, SECC: Started with a Table of Contents. Every new question went in there.

Qualities

- Architectural Views
  - Physical
  - Behavioral / Functional
  - Operational
  - Data and Interfaces
- Trade studies
- Detailed Design

Methods

- Heavyweight:
  - Many pages
  - How to do it wrong:
    - Write docs after the fact
    - Never read it, or update it
- Lighten up:
  - Memory map as header file, laid out and commented well enough to be readble by non-coders
  - Trade-studies captured as trouble-ticket comments, or email threads, and pointed to
  - Design baked into the code anyway, so manage the comments with mark-up tags.

Test Plan

Ex, RCF: Started with Makefile "make test" calling sub-tests. Plan was "implement that".

Qualities

- Describe Unit, Integration, Acceptance, Regression testing
- Test approaches: Test, Demonstration, Analysis, Inspection, Simulation, ...
- Problem reporting, tracking
- Tools you will use
  - Scripting languages? Logic Analyzer? Oscilloscope?
- Venues
  - Standalone workstation
  - Software simulators
  - Hardware – breadboard, engineering model, high-fidelity, flight article
- Identify critical capabilities (safing, hardware checkout, re-programming)

Methods

- Heavyweight:
  - Driven by RVM (Requirements Verification Matrix)
- Lighten up:
  - Continuous regression testing set up with the very first test, so later tests get written to plug in.
  - Group procedures into test sets by:
    - Delivery cadence, e.g., boot PROM before re-loadable images.
    - Venue, and personnel
    - Criticality
  - List every test procedure

Test Procedure(s)

Ex, SECC: 100 requirements, plus ~130 additional Verification Items.

Qualities

- Exercise specific requirement(s)
- List venues, steps, expected results

Methods

- Heavyweight:
  - Map procedures to requirements
  - Scrupulously record date, time, testers
- Lighten up:
  - "Run script #17"
    - Which of course captures its own results, possibly saved to CM
  - Template test procedure in CM no later than the fourth test. Peer-review the template. What are the input files? Where does output go? How are errors handled?
  - SUCCESS affirmatively shown; not just "did not see a failure"
  - Cross-cutting

Test Result(s)

Ex, TDS: Results were a filled-in copy of procedures, and comparison with golden runs.

Qualities

- Unambiguously decide whether development of some feature can stop now. I.e., requirements have been met.
- Can easily be more data storage than the rest of the artifact collection.

Methods

- Heavyweight:
  - Recall that VnV may rest on a Requirements Verification Matrix (mapping requirements to results)
  - Define point by point, in text, what to observe in each result to check-off each requirement.
- Lighten up:
  - If the requirement was "pass this test", the mapping is pretty simple.
  - Capture a set of "golden runs" for comparison in later re-runs. So you've walked through the output thoroughly, and the regression tests is simply "output the same thing"

User Docs

Ex, MOXIE: User wrote a command's help msg up front as the specification.

Qualities

- Command Dictionary
- Telemetry Dictionary
- Flight Rules
- Idiosyncracies
- Known bugs
- Expected usage, Concept of Operations
- Consumable limits
  - e.g. write-cycles in a chip

Methods

- Heavyweight:
  - Individual, signed-off document
  - Customer may be the next level of integration, rather than end user
- Lighten up:
  - Enlist the user to write the parts they care about most, based on informal discussions with developer
  - Use a real technical writer to interview developers and create first draft.
  - When emailed a question, answer with a documentation draft section

RDD (Release Description Doc)

Jet Propulsion Laboratory
California Institute of Technology

Ex, TDS: Included annotated photos with PROM-burner and 5 chips.

Qualities

- Product Identification
- Development System Description
  - Tool chain, with versions
- By version:
  - Capabilities implemented
  - Change Requests included
  - Bugs fixed
- Known bugs
- Test Summary
- Checking, Building, and Loading instructions

Methods

- Heavyweight:
  - Signed-off document lives separately from code
- Lighten up:
  - Build instruction one-liner: "make build"
  - Another one-liner "make versions" can spit out versions of as-run compilers, interpreters, libraries, OS, packages, etc.
  - A picture is worth 1,000 words.
    - "Do it like this"

Reviews

Ex, SECC: Review was 22 slides, no new words. Guided tour of Design Doc sections.

Qualities

- Bring stakeholders onto same page
  - But, different sub-groups for different subject matter
- Peer reviews
  - Of each artifact, each algorithm, each data item...
- Milestone reviews (PDR, CDR, ORR, SRCR)

Methods

- Heavyweight:
  - Slide package summarizes key points
    - Rarely explains in depth
    - Not signed
    - Not updated
- Lighten up:
  - Artifacts already capture the important information. Use, refer to, or quote them in the review
  - Peer reviews as cross-training exercise

Final Sign-off

Ex, TDS (M2020): inherited as part of the hardware rather than being "delivered".

Qualities

- Stakeholders agree what has been delivered
- Lists all deliverables, and their status
  - Including test results
- Signed by:
  - Person who did the development
  - Person who did the testing
  - (ideally) Independent 3rd party, e.g. Quality Assurance
  - Person who paid the bills

Methods

- Heavyweight:
  - Big meeting with all stakeholders
  - Formal checklist
- Lighten up:
  - Barring objection, call it done

Tailoring Artifacts

- What do you need to know when?
  - Record it as it is decided, not a doc exercise after it's too late to get ROI (i.e., the 1st time the question is asked, not the 5th)
  - Rather than answer questions by email; write the artifact and email that for proof-reading ("is this clear?")
- Bake information into tool-chain.
  - Artifacts become lightweight collections of pointers and references
  - Documentation generators like doxygen, pydoc, etc, *if* documentation is going to be used

Tailoring Artifacts, cont

- Document, wiki, cocktail-napkin are all fine, if they supply:
  - Revision Control
  - Disaster recovery
  - Searchability
  - 30-year retention?

- "The Board views the endemic use of PowerPoint briefing slides instead of technical papers as an illustration of the problematic methods of technical communication at NASA."
  - -- Columbia Accident Investigation Board, August 2003