Mission Success with Software Development Principles: Application to Mars'98 Missions

A paper presented at the 25th Annual Software Engineering Workshop, Nov 29-30, 2000, GSFC

Milton Lavin, The Center for Space Mission Information and Software Systems
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA 91109

Background

The objective of this paper is to illustrate the use of experience-based development principles in order to eliminate mission software defects prior to launch. The application context is selected from software problems contributing to two recent NASA mission losses -- the Mars Climate Orbiter (MCO) Mission in September 1999 and the Mars Polar Lander (MPL) Mission in December 1999. In the case, of MCO, the failure to achieve Martian orbit was traced to a units error in the Small Forces software used in navigation. The presumed cause of the MPL loss during descent is a defect in the software responding to the touchdown sensor. In each case, the application is presented with the intent of demonstrating the potential synergy and effectiveness of employing a comprehensive set of software development practices to root out defects. Development of the Software Principles began just prior to the MCO loss, and they were completed in June 2000. Because the MCO and MPL Loss Reports were inputs, these two retrospective applications cannot be construed as validating the content or use of these principles.

Effective software development is dependent both on a mature development process (standards, procedures and policies) and on the continual incorporation into the process of lessons learned from historical projects. These "lessons learned" are typically more detailed than general principles of software engineering because they are idiosyncratic to each development context -- in JPL's case, robotic missions into deep space. It is particularly important to document experience and incorporate it into the development process when many project managers have limited experience in software development. Tighter development schedules and reduced budgets are another factor that stimulates the search for more effective ways to build quality into the product while satisfying programmatic demands. Thus for several reasons, it was decided to make these lessons learned more accessible to the JPL development community by 1) analyzing key factors in recent mission successes and failures and 2) distilling the findings into concise principles to be observed in planning and implementing a software development.

Development of Software Development Principles for Flight Systems

Stimulated by the retirement of large numbers of its senior technical staff and the concurrent growth in the number of active projects, in 1998 the JPL Space and Earth Sciences Program Directorate undertook an initiative to document good mission and spacecraft design practice. The resulting document titled " JPL D-17868, Design, Verification/Validation and Operations Principles for Flight Systems" contained a set of concise guidance on 31 general topics (e.g. Risk-Based Design Trade-Offs, Single Failure Tolerance/Redundancy, Critical Sequence Telemetry Monitoring) and 28 Detailed Topics (e.g., Power-On Reset State, Slosh Dynamics, Pyro Design and Firing Margins). Use of these principles is intended both to prevent defect injection and to eliminate defects that were not caught in the first instance.

These principles are composed as a “shall statements” to emphasize their importance, and they are called “principles” rather than “requirements” to give projects leeway in applying them. In the JPL culture, a principle may be modified or perhaps violated for good reason, but modifying a requirement entails a more rigorous and formal process. Adherence is verified by

- Documenting deviations from these principles in the Project Implementation Plan (PIP) and the Software Development Plan
- Revisiting these principles during reviews of detailed plans, requirements, designs, and test results

The initial version of D-17868 contained a brief section titled “Software Design Margins and Software Verification”, which has been expanded into a set of 103 software development principles in the current version of this document. These JPL Software Development Principles are organized around the following ten life cycle activities, with a separate section devoted to the special needs of flight software development:

Exhibit 1: Content of Software Development Principles

Topic:	No. of Principles:
Systems Definition/Systems Engineering	13
Planning and Monitoring	17
Cost Estimation	4
Risk Management	3
Organization and Staffing	6
Design and Implementation	14
Integration and Test	16
Configuration Management	3
Software Acquisition	2
Product and Process Verification	4
Flight Software Development	20
Total	102

The Software Principles are based on experience in recent flight projects -- both successes and failures. Primary sources were discussions with software managers, system engineers, and developers spanning all major software-intensive mission systems, plus the findings and lessons learned in the cited sources:

- Investigations of the two recent Mars Mission losses
- An internal study of the root causes of flight software cost growth and schedule slip in 8 current JPL missions
- Recommendations from a recent GSFC/JPL Quality Mission Software Workshop
- NASA's Lessons Learned website.

25% of the Software Principles treat design -- i.e., they specify desirable product behaviors or desired relationships among the components of a system or subsystem. The remaining 75% are almost equally split between 1) planning and organization and 2) a specification of the engineering process. This distribution of content was not planned; it evolved in the process of identifying and addressing the most pressing needs to introduce more disciplined development practices. Examples of design and engineering process principles follow:

Design Example: "The software self-test and built-in test routines shall be removable for flight. If not removable, the test routines shall not cause flight hardware damage or interfere with the proper execution of the flight software if tests are inadvertently executed."

Engineering Process Example: "Test planning and the design of test cases shall be based on the premise that the software contains serious errors that must be detected via thorough identification of off-nominal, implausible, and otherwise unexpected conditions arising from

- Defective software logic design
- Incorrect initialization of parameter values
- Erroneous parameter values in data input files
- Hardware failures, transient or anomalous hardware behavior, and unexpected hardware-software interactions
- Processor resets"

The process of developing these Software Principles was iterative, with each version subject to extensive peer review by developers, subject matter experts, line managers, and project managers. In the early phase of development, the selection of principles was guided by an editorial board, armed with the following evaluation criteria:

- Be relevant to JPL's business and culture
- Make a significant difference in cost, schedule, or quality
- Omit what is widely practiced
- Be applicable to a wide spectrum of projects
- Be useful to project managers, project element managers, reviewers, and developers

After the first workshop review, the principles were baselined, and further changes were approved by a change board. Projects in the planning and early implementation phases will be

the initial users of these Software Principles -- Mars Exploration Rover, and Outer Planets/Solar Probe are two examples.

Retrospective Application of Software Development Principles to Mars'98 Missions

NASA's Mars '98 missions were implemented in the Faster/Better/Cheaper mode, with cost containment a major consideration. Subsequently, NASA has modified the F/B/C philosophy to emphasize risk management and make mission success the paramount factor. As a result, missions currently under development are re-instituting more formality into the development process with special attention to the detailed technical reviews that are the preferred venue for application of much of the material in the Software Principles.

JPL managed the development and operation of both the Mars' 98 missions -- the Mars Climate Orbiter (MCO) and the Mars Polar Lander (MPL)The spacecraft used in the two missions were designed and developed via a systems contract with Lockheed-Martin Astronautics (LMA). The development was guided by a documented process that incorporated activities meant to ensure a quality product. Yet in each case, critical components in the mission software were flawed. What happened?

MCO was lost during attempted insertion into Mars orbit because its actual altitude was much lower than predicted due to a navigational error not discovered until after the fact. The root cause of the error was the use of English units (pounds-force seconds) instead of metric units (Newton-seconds) in the Small Forces navigation software that was used to interpret the impulses applied to the spacecraft during periodic angular momentum desaturation (AMD) maneuvers. The use of metric units as well as the data formats to employ were specified in a navigation software interface specification (SIS) published by JPL in 1996. Although this SIS was referenced in the Small Forces software requirements, neither the requirement to use metric units nor the mandated data format was observed by the software developer. These errors were not caught in the walkthroughs of Small Forces requirements, design, or code. Nor were the errors caught in the pre-flight tests of the MCO navigational software. Subsequently, the error in data format was identified and corrected, permitting attempts to reconcile the anomalous results yielded by the navigation process with the magnitudes of the Small Forces calculated from the AMD files. However, the accumulated errors from numerous desaturation maneuvers were so difficult to detect that there was considerable uncertainty about the true trajectory of MCO at the time of Mars orbit insertion. Although a final trajectory correction was considered, it was not executed because the required command sequence had not been baselined and thoroughly tested.

MPL was lost during the entry, descent, and landing (EDL) sequence due to presumed premature engine shutdown. (MPL transmitted no telemetry data during descent.) The presumed fatal error was traced to a database initialization error in the software that interrogated the touchdown sensors and controlled descent engine shut-down. EDL requirements specified activation of the touchdown sensor software at 40 meters above the Martian surface, and a subsequent requirements change specified that the transient signal generated by the touchdown sensor at the time of landing legs deployment be ignored until the spacecraft had descended to 40 m altitude. This mandate to ignore touchdown sensor input was not flowed down to the software requirements, contributing to a database initialization error. This error caused the touchdown sensor software to store the bogus transient signal received at leg deployment and then use this information to command descent engine shutdown when the landing sequence was initiated at 40

m. The error in the touchdown sensor software logic was not caught in a trace of system requirements to software requirements and thence to test scenarios and cases. It was not caught in detailed technical reviews of software requirements, design, and code -- primarily because knowledgeable mechanisms engineers and system engineers were not present. And this logic error was not caught in ATLO system testing because the EDL mission sequence was not tested from beginning to end under simulated flight conditions.

In the case of both the MCO units error and the error in the software logic controlling the MPL descent engine shutdown, the defect was introduced in the flowdown of systems requirements to software requirements, but the defect remained undetected because

- Design and code walkthroughs were ineffectively executed
- Design documentation did not facilitate the verification of design requirements
- System testing was not fully informed by a complete set of software requirements
- End-to-end testing and stress testing were not done effectively

Exhibit 2 summarizes the multiple opportunities to identify and correct each defect via the application of JPL's Software Development Principles -- beginning with planning for peer

Exhibit 2: Application of Software Development Principles to Detect Mars Climate Orbiter and Mars Polar Lander Software Defects

Software Development Principle	*MCO*	*MPL*
Planning and Monitoring		
3.2.7 Joint development planning for interfacing hardware and software		X
3.2.9 Identification of milestone and peer reviews	X	X
3.2.10 Participation of hardware engineers and operations team in reviews		X
3.2.12 Comprehensive peer review of intermediate products	X	X
Risk Management		
3.4.3 Early validation of interfaces, high-risk algorithms, and COTS	X	X
Design and Implementation		
3.6.3 Design traced to software and mission requirements	X	X
3.6.4 Documented analytical basis for logic design		X
3.6.6 Software logic to verify values of input and output parameters	X	X
Integration and Test		
3.7.6 Detailed testing of mission phase transitions		X
3.7.7 Testing to address Fault Tree Analysis and off-nominal hardware behavior		X
3.7.9 Stress testing used to aggressively find latent defects	X	X
3.7.13 Trace from final system test to mission requirements	X	X
Software Acquisition		
3.9.1 Project Implementation Plan to address management of software acquisition:		
• In-process JPL review of intermediate products	X	X
• JPL participation in pre-delivery testing	X	X
Product and Process Verification		
3.10.4 Acceptance test that exercises mission-critical systems	X	X
Flight Software		
4.5 Accommodation of nominal, off-nominal/transient inputs		X

reviews and ending with pre-delivery and acceptance testing. An "X" in Exhibit 2 denotes that the indicated activity was either missing or ineffectively implemented. (Principle numbering from the parent document is retained to help those who may wish to refer to that source.) Note that systematic application of the JPL Software Development Principles would have provided 10 opportunities to catch the MCO defect and 16 opportunities to catch the MPL defect.

Summary

Because software creation is complex and subject to a variety of human error in specification, design, and implementation, it is not surprising that serious defects were introduced into the MCO and MPL software during development. There were numerous opportunities for defect identification and removal, but they must be applied systematically as an ensemble because in practice, each is likely to be only partially effective. This is especially important in a development environment that is based on the philosophy, "Test as you fly, and fly as you test". It is hoped that codification of our mission experience as a set of Software Development Principles will facilitate defect avoidance and detection in future missions.

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

J. Casani et al., Report on the Loss of the Mars Climate Orbiter Mission, JPL internal document D-18441 (November 1999)

J. Casani, et al., Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, JPL internal document D-18709 (March 2000)

J. Hihn and H. Habib-agahi, Flight Software Cost Growth: Causes and Recommendations, JPL internal document D-18660 (February 2000)

M. Landano and J. Rose, Design, Verification/Validation and Operations Principles for Flight Systems, JPL internal document D-17868, Rev. A (November 2000)

C. Lin and H. Kea, GSFC/JPL Quality Mission Software Workshop Report (October 1999)

NASA Lessons Learned Information System site -- http://llis.nasa.gov/