JPL Fault Protection Software Experiences

Response to:
(1) Growth in Flight Software (FSW) Complexity" V5, NASA OCE
(2) Reducing Complexity in Flight Software, July 11, 2007, JPL

Kevin Barltrop, Jet Propulsion Laboratory, California Institute of Technology
Dan Dvorak, Jet Propulsion Laboratory, California Institute of Technology

Background

- A recent memorandum, “Issue: Growth in Flight Software (FSW) Complexity V5, NASA OCE”, was written describing the challenges faced by NASA which result from the growth in complexity and size of flight software.
- A task was initiated to respond to the four special interests identified in that memo.
- This activity responds to Special Interest #3:
  - *“Fault protection logic accounts for a sizable portion of flight system software. Are there techniques which effectively manage the complexity of fault protection systems?”*

Background (cont'd)

- The specific actions for JPL identified in Special Interest #3 are:
  1. Share JPL's robotic mission fault protection experiences in a workshop specifically focused on fault protection, and include participation from all NASA FSW developing centers as well as representatives from industry and academia.
  2. Participate in the "Fault Protection Workshop" planned for January/February which will be coordinated by MSFC.
  3. Make specific recommendations for techniques that bound the fault protection capabilities required for mission success.
  4. Investigate and document approaches to fault protection used to date within NASA. What worked and what didn't?
  5. Investigate the feasibility of eliminating or at least minimizing fault protection software as a separate entity.

- This briefing addresses a subset of those actions.

Objectives of Briefing

- The objectives of this briefing are:

A. Share JPL experiences by describing the evolution of fault protection during its history in deep space exploration.
   - (Addressing Action #1)

B. Examine issues of fault protection scope and implementation that affect missions today.
   - (Collecting information for Actions #3 and #4)

C. Discuss solutions for the problems of today and tomorrow.
   - (Collection Information for Actions #3 and #5).

- Its content is an initial cut at identifying key issues and potential solutions, but we're hunting for feedback from the community.

Topics

- JPL Experiences in Fault Protection
- What are the key problems?
- What are some solutions?
- Case Studies to Illustrate in More Detail

Topic A. JPL Experiences in Fault Protection

JPL Experiences in Fault Protection

- We'll look at three threads for describing JPL experiences:

1. How have the requirements on fault protection evolved with the progression of more complex missions over the decades?
2. How have software solutions for fault protection evolved to meet the requirements?
3. How well have the software solutions matched the demands of the missions?

And how do each of the above points relate to the issue of fault protection software complexity?

How Have Fault Protection Requirements Evolved?

- The top level requirements for fault protection in JPL's deep space missions have largely remained the same for 30 years (ref 1):
  1. Operate with Limited Ground Contact
  2. Protect Fragile Systems (Low Margins)
  3. Protect Critical Activities
  4. Accommodate other Mission Constraints
  5. Maintain Single Fault Tolerance.
- Near-earth missions, by contrast, often lack one or more of these requirements.
- A key element of JPL's philosophy towards fault protection is that that it is better to solve the problem of creating fault protection software, than it is to ignore failure scenarios that would end the mission.
  - Underlying this is the belief that we have the knowledge, techniques, and experience to succeed in developing that software.
- Despite the constancy of the top level requirements, it is generally agreed that fault protection has become more complex!

How Have Fault Protection Requirements Evolved?

- If the top level requirements have not changed much, why has complexity increased?
  Because:
  - Character of Missions Have changed
  - Technologies have changed
  - Acceptable Level of Risk changes during the Project Lifecycle
  - Reliability Analyses for more complex systems are often 'late' in the Project Lifecycle
  - New players in this field introduce new approaches and learning curves

- These factors often contribute to the growth of *essential complexity* in new missions.

- In turn, because both the software approach and the lateness of development activities tend to amplify complexity, these factors contribute to the growth of *incidental complexity* in new missions.

What is Essential Complexity?

- **Essential complexity** is primarily a systems rather than software question.

  For example:
  - What activities must the application perform?
  - What functions are within those activities?
  - What information must be processed?
  - What are the requirements with respect to precision and speed?

- Essential complexity will continue to grow so long as we continue to demand that fault protection be applied to increasingly complex systems and activities.

What is Incidental Complexity?

- **Incidental complexity** arises during the process of creating the software to accommodate the essential complexity
  - In a perfect world the resulting software would have the minimal amount of complexity needed to satisfy the essential complexity.
  - In practice software development acts as a sort of multiplier for the essential complexity.
  - It's effect is made worse by failure to do appropriate hardware/software trade studies.

- What factors contribute to the growth of incidental complexity?
  - Choice of software architecture
  - Skill and experience of software team
  - Utility of development tools
  - Opportunity and resources to make good fixes versus quick fixes
  - Avionics choices that limit software capabilities

- Incidental complexity will grow along with growth in essential complexity if we continue to do business the same way.

Examples of Increasing Essential Complexity

- Increased competition for DSN access means spacecraft must tolerate operating without ground intervention for longer periods.
- Activities and environments have become more complex:
  - → Fly-bys → Orbit insertions → Entry-descent-and-landings → Complex tours and relays of moon systems → Daily surface operations → Precision guided intercepts, sample returns, and extreme data contiguity, …
  - These activities are often “single opportunity” ones upon which mission success is based, so “safe and wait” becomes a mission-ending strategy.
  - Later spacecraft also often carry more science investigations then their predecessors.
- Hardware has become more complex:
  - New power systems (concentrators)
  - New propulsion technologies (ion thrust)
  - “Semi-Reusable” platforms applied to missions with a poor design fit.
  - More complex science instruments
  - Integrated use of science instruments to support engineering functions.
- Past missions teach us to employ new safeguards:
  - Minimize likelihood of unnecessarily powering on new hardware (Voyager experience)
  - Apply separation of concerns within the design.
  - Organize behavior around preserving function rather than applying knee-jerk fault responses.

How Has Fault Protection Software Evolved?

- Fault protection software has evolved in response to the increasing complexity of the activities in which it must operate.
- Unfortunately, problems in recent missions which over-extended Earth-orbiter architectures to support deep space missions suggest that the software has not kept pace.
- A mission usually understands this only after it has already decided on the scope of effort for fault protection.
- This often leads to a painful choice:
  - Accept more development risk by extending the existing software to cover activities that are beyond the scope of those for which it was originally designed.
  - Accept more operational risk by reducing the scope of compliance with the high level requirements.

Topic B. What are the key problems?

What are the Key Problems?

- We have cost overruns, schedule slips, confusion during reviews and operations, and endless debate over the complexity of the fault protection.

- The underlying problems are:
  1. Poorly defined scope for the fault protection.
  2. Mismatch between the requirements and the software solution.
  3. Lack of good development solutions.

Underlying Problem #1
Poorly Defined Scope

- In what ways can scope be poorly defined?
  - Lack of agreed-to fault tolerance policy at the project/program levels.
  - Insufficiently articulated derived requirements during formulation phase
  - Insufficient architectural analysis at a stage early enough to understand the means to meet requirements.
  - Lack of consensus across centers concerning the appropriate scope of fault tolerance.
  - Cold feet on risk as risk-averse project implementers become reluctant to accept the high risk posture put forward by the project proposers.
  - Equivocation concerning definitions of “fault tolerance” and other terms.
  - Others?

Underlying Problem #2
Requirements / Solution Mismatch

- How do we end up with a mismatch between the requirements and the design solution?
  - Insufficient analysis of existing architectures to understand the degree to which they can satisfy the requirements.
  - Adaptation problem due to extending existing architecture instead of “fixing” it.
  - Tunnel vision in which implementers focus about their favorite issues at the expense of others.

Underlying Problems #3
Lack of Good Development Solutions

- In what ways do we lack access to good development solutions?
  - Inherited designs or concepts often have a software architecture in which the existing framework creates an unreasonable implementation and/or unreasonable operations burden on end users.
  - Developers often lack access to good tools to manage the design and to allow early testing. (Lots of “spreadsheets”).
  - Fault protection requires the involvement of the systems and subsystems to which it is applied, but is often begrudgingly supported during its development.
  - Verification and validation, particularly at the system level, occur late because they rely on having a mature system and test infrastructure.

Topic C. What are some solutions?

Solutions for Today's Problems

- To avoid the cost overruns, schedule slips, and debates we need to address the problems:

1. Clearly define and contractually agree to the risk posture and high level requirements that drive the scope of the fault protection.
2. Clearly understand the matching of the software solution to the requirements.
3. Make use of solid software principles and strong architectures to provide robust solutions.

Possible Scope Solutions

- Adopt industry-wide policies and practices that can be applied during the formulation phase of missions.
- Define standard guidelines for review of fault protection requirements in AOs.
- Define standard guidelines for review of fault protection compliance by proposals.
- Publish a standardized dictionary of terms applicable to fault protection.
- Is this a set of NASA-wide fault protection standards?
- Cost/benefit analysis to company bottom lines?

- Other ideas?

Possible Mismatch Solutions

- Promote the use of rigorous analysis with models and architectures to identify disconnects between requirements and design.
- Beef up formal review criteria to confirm that attention and direction of effort is appropriate.
- Publish architectural principles against which designs are be compared.
- Consider contract line items to upgrade existing architectures to meet future needs rather than accepting them as-is.

Possible Development Solutions

- Establish design standards and patterns
- Establish governing or advisory groups
- Work with or establish ongoing working groups
- Develop and (inexpensively) deploy frameworks and architectures that are intended to be reused.
- Develop and deploy tools that enable early modeling and architectural analysis.

- Fault Protection Standards Group?